Volare
Mobile Context-aware Adaptation for the Cloud

Panagiotis Papakos

A dissertation submitted in fulfillment
of the requirements for the degree of
Doctor of Philosophy
University College London

Department of Computer Science
University College London

3 October 2014

Statement

I, Panagiotis Papakos confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Date	**Signature**
03/10/2014	Panagiotis Papakos

Abstract

As the explosive growth in the proliferation and use of mobile devices accelerates, more web service providers move their premises on the Cloud under the Software as a Service (SaaS) service model. Mobile environments present new challenges that Service Discovery methods developed for non-mobile environments cannot address. The requirements a mobile client device will have from internet services may change, even at runtime, due to variable context, which may include hardware resources, environmental variables (like network availability) and user preferences. Binding to a discovered service having QoS levels different from the ones imposed by current context and policy requirements may lead to low application performance, excessive consumption of mobile resources such as battery life and service disruption, especially for long lasting foreground applications like media-streaming, navigation etc. This thesis presents the Volare approach for performing parameter adaptation for service requests to Cloud services, in SaaS architecture. For this purpose, we introduce an adaptive mobile middleware solution that performs context-aware QoS parameter adaptation. When service discovery is initiated, the middleware calculates the optimal service requests QoS levels under the current context, policy requirements and goals and adapts the service request accordingly. At runtime, it can trigger dynamic service rediscovery following significant context changes, to ensure optimal binding. The adaptation logic is built through the characteristics of the declarative domain-specific Volare Adaptation Policy Specification Language (APSL). Key characteristics of this approach include two-level policy support (providing both device specific and application specific adaptation), integration of a User Preferences Model and high behavioral (parameter adaptation) variability, by allowing multiple weighted adaptation rules to influence each QoS variable. The Volare approach supports unanticipated quantitative long term performance goals (LTPGs) with finite horizons. A use case and a proof-of-concept implementation have been developed on cloud service discovery through a cloud service provider, as well as an appropriate case study, which demonstrates significant savings in battery consumption, provider data usage and monetary cost, compared to unadapted QoS service bindings, while consistently avoiding service disruptions caused by QoS levels that the device cannot support. In addition, adaptation policies using the Volare approach tend to increase in size, in a mostly linear fashion, instead of the combinatorial increase of more conventional situation-action approaches.

Acknowledgements

I would like to thank my supervisors, David S Rosenblum and Licia Capra, for their invaluable help and guidance during this research. David's advice and supervision were instrumental in the creation of this work, and the development of my skills as researcher. During our work together he also taught me invaluable teaching and management skills that will undoubtedly be useful for the rest of my career. Licia's feedback provided important insight into the needs and research opportunities in the distributed adaptation-computing domain, and her eye for detail ensured I stayed the course and provided the necessary focus for my work. Thank you both for your patience and unwavering support.

I would also like to thank Emmanuel Letier for his helpful advice and support as my assessor during my initial and transfer examinations. His insightful commentary helped me identify key areas I needed to develop further. He was also an excellent coworker during various teaching posts we worked together, and taught me a lot about other aspects of the academic work.

I would also like to express my thanks to my Thesis Examiners Professors Mark Harman and George Roussos for their invaluable questions and remarks.

I would like to express my gratitude to Arun Mukhija for helping me integrate into the research environment. In addition, I'd like to thank him for all the discussions about further expansions to his research work that helped form the initial basis for what eventually became Volare. The name Volare was inspired by discussions with him. I'd also like to thank my fellow co-workers and researchers, who helped inspire, motivate, and provide feedback through the years, such as Fokion Zervoudakis, Arturo Araujo, Michele Sama, Veerappa Varsha and others.

Gratitude also goes to the faculty, administration and technical staff at the Department of Computer Science at University College London for providing me with the resources and support to develop this thesis.

Finally, I'd like to thank my family and friends for their unwavering encouragement and support.

The Engineering and Sciences Research Council (EPSRC) funded the research for this thesis, under the supervision of David S Rosenblum.

Contents

List of Figures

List of Tables

Abbreviations

The following abbreviations are used throughout the text:

AdaptationM	Adaptation Module
APSL	Adaptation Policy Specification Language
ART	Adaptation Reasoning Technique
BindingM	Binding Module
BV	Behavioral Variant
CAA	Context Aware Adaptation
CAHiD	Context & Adaptation History Database
CAHiM	Context & Adaptation History Module
ContextMonitoringM	Context Monitoring Module
DCAA	Dynamic Context Aware Adaptation
MECC	Mutually Exclusive Criteria Conditions
MESC	Mutually Exclusive Structural Configuration
PD/V	policy-driven-variable (calculated at Policy Execution)
PEVApp	policy execution & Verification Application
PolicyFilesM	Policy Files Manager
ServiceRequestM	Service Request Module
UCP	User Choices Profile
UPI	User Preferences Interface Application
UP	User Preferences
VP	Variation Point
SV	Structural Variant
SVP	Structural Variation Point
SWC	Strategy Weight Coefficients
WAS	Weight Assigning Strategy
WB	weight-based
WBART	Weight-based adaptation reasoning technique

Volare Mobile Context-aware Adaptation for the Cloud

1 Introduction

1.1 The Research Problem

Application development in mobile systems faces the challenges of constrained and/or consumable device resources like battery power, CPU, RAM, storage memory and communication capability, frequent environmental context changes like network availability and bandwidth or location variation, and variable User Preferences concerning desired operation parameters due to mobility. In order to ensure that users meet their non-functional requirements in such scenarios, software services need to be context-aware and adapt to their context, to ensure that the Quality of Service (QoS) levels selected reflect the current needs of the user, which change according to context.

By *context* in this work we mean: (i) device resource and execution-environment parameters (like available battery power level or CPU), (ii) environmental parameters (like: bandwidth or GPS coordinates), and (iii) User Preferences.

Context Awareness is defined in this work as the capability of a system to be aware of the context parameters of the execution environment (see also §2.1 for more detailed reference).

Consequently when mobile devices launch Service Discovery for a web service on the cloud or web, in order to ensure satisfactory operation, a service of the requested content needs to be discovered providing the QoS levels corresponding to the current context and the adaptation logic requirements. Especially for long-lasting services, like media-streaming, navigation, video-calling, tele-conferencing etc., since context conditions may vary during runtime, service levels requested at initial Service Discovery may not be optimal for the new conditions of the device, leading to adverse effects on performance or cost or device resource utilization and overall user satisfaction. In this case dynamic adaptation capability to the new context at runtime has to be supported.

A plethora of research approaches[13][18][30][34] propose general or domain-specific solutions to these challenges, with the state of the art recommending Dynamic Context-Aware Adaptation (DCAA) through a mobile middleware or distributed framework implementing context-awareness, adaptation reasoning and adaptation implementation functionalities. Use of external adaptation logic is recommended for easy updates over the software lifetime, in the form of adaptation policies developed through an appropriate Adaptation Reasoning Technique and based on an Adaptation Policy Specification Language.

There remain certain domain-specific challenges to applying this general approach for implementing DCAA on service discovery by mobile applications that are discussed in the following subsections.

In this work, we are focusing implementation and evaluation on Cloud Services, due to the fact that they are a newer up and coming architecture that seems to be getting higher relevance in the Distributed Computing state of the art [122][121][111]. The work itself is relevant to mobile computing in general, however.

1.2 Challenges for Service Discovery on the Cloud

1.2.1 The Cloud Computing Paradigm

One of the latest emerging distributed computing paradigms is that of Cloud computing [1][4][5][104][106][107], which promises reliable services delivered through next-generation data centers built on resource virtualization technologies. Given the explosive development of cloud computing in recent years, the Software as a Service approach in Cloud computing is becoming increasingly prevalent [106][107]. Cloud computing can offer significant advantages to mobile application developers, by enabling remote data storage, computation capabilities etc., as well as eventual computation off-loading, an area gradually covered by Mobile Cloud Computing (MCC) [97][106].

At the same time, Cloud computing offers to mobile applications accessing to online resources and services in a manner similar to Web Services and Grid Computing, as increasingly the trend among Service Providers is to move their premises on the Cloud due to concrete operational and business advantages [105][109][112].

Scope of this research: *In this work we focus on Cloud Service Request QoS parameter adaptation as pertains to Cloud Service Discovery by mobile applications through application-selected Cloud Service Brokers or Service*

Providers on public Clouds, where alternative services of the same content at different QoS levels are available.

As an extension, our approach may also be utilized on Service Discovery on private or community Clouds (with no cost obligation), or on the Web through Service Brokers or Service Providers, with the cost dimension of adaptation reasoning and of the SR to be ignored as non-relevant.

1.2.2 Service Discovery on the Cloud

Service Discovery on the Cloud – in a Software as a Service business model(see ch2.6.1)- presents to the mobile applications developers its own particular challenges outlined below, in addition to the DCAA challenges for mobile systems already described above, namely:

1. Monetary cost for binding to services provisioning on Public Clouds

Services on Public Clouds present the specific feature [1][4][21][46 - 52] of imposing monetary costs on their client, according to resource usage. Although the pricing policy depends on the CSP or CSB, the cost for use of services on the Cloud typically depends on the resources utilized and on the QoS level of each service among alternative ones of the same content. Consequently, a service request for service on Public Clouds should include the price per resource used as a selection QoS variable and a cost-oriented strategy should be an integral part of a Policy for service discovery and binding on services on the Cloud.

2. Technical challenges on Service Discovery on Public Clouds

Another major issue with Service Discovery on the Cloud is that there is often limited control about the decision-making process of matching a service request with the closest possible service [4][5][105][115][121], see also §2.6.

Additionally, there are currently no industry standard protocols when it comes to advertising Cloud services or QoS service levels. Thus, usually a Cloud Service Provider or Service Broker has its own proprietary protocols. Consequently the Cloud Service Consumer should be directed to relevant for the requested service CSPs or more generally to CSBs [105][21][109][112].

1.3 Challenges on Adaptation Logic and Reasoning

As described above, in addition to the DCAA software, the external adaptation intelligence, i.e. the Adaptation Policy Logic, authored using an Adaptation Reasoning Technique supported by characteristics of the Adaptation Policy Specification Language and of the middleware, dictates the adaptation process.

Typically, the Adaptation Policy Logic is specified through three main tools [18][34][54][56][61]:

- An appropriate domain-specific **Adaptation Policy Specification Language (APSL)** to enable descriptions of policy languages (see also §2.3)
- An **Adaptation Reasoning Technique,** that enables as described below (see also §2.4)

  There are three established adaptation reasoning approaches [18][34], we distinguish as follows: The rule-based technique - known as "Action-based" technique [7][58] implements "condition-action" adaptation and is popular for relatively simple applications/scenarios. The "state-based" techniques - namely the "Goal-based" [74] and the "Utility-based" [6][18] ones - both require complete system state re-evaluation at adaptation time, thus entailing a relatively much more tedious development task and a significant computational burden for the device.
- A **User Preferences Model** that is often integrated in the Policy, setting user preferences through an application-specific User Interface [28][36][83] and/or through dynamic Applications Profiles for User preferences and/or resource requirements [6][15].

We mention briefly below several important adaptation reasoning challenges for the dynamic mobile environment that we claim the adaptation reasoning should support and the Adaptation Policy should satisfy:

1. Support of both device and application specific adaptation

One of the challenges of adaptation in mobile computing is that often the application developer will not know, beforehand, the exact specifications of the platform his application will be running on (various mobile phone models etc.). To ensure efficiency when performing QoS parameter adaptation however, the adaptation policy must take into account the unique capabilities of the mobile device, as well as the specific needs of the application itself. The solution of creating separate policies for each possible device configuration is often unattainable, due to the number of possible devices. Thus, ideally, some sort of collaboration between device-specific and application specific policies is necessary.

2. Support of alternative dynamically selectable and customizable adaptation behaviors

Given the need for the user to The challenge consists in designing a configurable policy-based dynamic User Preferences Model that enables easy on-the-fly selection by the User of policy-based alternative adaptation behaviors on generic crosscutting concerns and thus valid for every application and dynamic customization by the User of the adaptation behavior on predetermined characteristics.

3. Support of Long Term Performance Goals

A unique characteristic of mobile devices is the resource budgets associated with it, either enforced by the service provider (data allocation), the user (monetary budget constraints), or the limitations of the device itself (battery life). Thus, adaptation behavior aimed for such devices needs to be able to take into account such finite resources in its adaptation behavior.

Such “Long Term Performance Goals” (LTPGs) may be the battery power use Mgmt over every battery discharge period, or the monthly cost management of binding on Cloud services or of monthly data volume use.

LTPG monitoring at runtime requires information processing over the temporal horizon, history storage and maintenance capability and adaptation guidance models that are not trivial in mobile applications.

Such policy-based Long Term Performance Goals undertake supervisory tasks and relieve the User from the need to monitor and intervene at Policy operation.

4. Adaptation Logic Building On Selected “Adaptation-Concerns”

We argue that in the adaptation process there are multiple active “**adaptation-concerns”** (targeted adaptation viewpoints) that need to be considered and “represented” at every relevant context instance in the rules selectable at policy execution. These “adaptation-concerns” represent different motivations for adaptation such as: device resource optimal use, satisfactory performance, optimal cost strategy, etc., in order to optimize user satisfaction.

Allowing multiple such adaptation-concerns to be modelled separately in the adaptation policy would allow developers more flexibility in choosing the relative importance of certain adaptation-rules over others as appropriate, and would enable more thorough modeling of adaptation requirements in the policy itself.

The conventional action-based adaptation reasoning technique by imposing a single rule per adaptation-variable selectable at policy execution restricts the synthesis of relevant concerns in one rule per adaptation-variable. It is difficult to provide fine-grained adaptation with just one rule selectable on each adaptation-variable under any contextual situation in the conventional approach. Additionally adaptation tends to be abrupt across “condition” boundaries, where another rule is selected.

An APSL and an Adaptation Reasoning Technique that would permit possibly multiple rules on the same adaptation variable to selectable at policy execution imposing participative weighted contribution to the adaptation choices, is a first challenge at adaptation policy authoring.

5. Adaptation Logic with High Behavioral Variability

Another challenge consists in expressing in the policy richer behavioral (parameter adaptation) variability under each structural configuration, making adaptation behavior more fine-grained and gradual. However, high behavioral variability increases the number of rules required in a combinatorial manner to the number of behavioral variants, thus rendering policy authoring more burdensome and higher the risk of rule faults [34][101], in a typical situation-action approach. This is due to having to define a separate policy for each context situation, which can quickly become untenable.

The challenge is raised for the adaptation logic to enable more fine-grained, gradual adaptation wherever possible, and to express different crosscutting concerns (common for all - or most - applications), imposing richer (higher) behavioral (parameter settings) variability, without undue increase in the number of rules [101].

1.4 Research Hypothesis

In response to the above in §1.1 state of the art CAA design requirements for supporting mobile applications, as well as the challenges for service discovery on the Cloud expressed in §1.2, the adaptation reasoning research challenges mentioned in §1.3 and the resulting Research Question described in §1.4, we come up with the following hypothesis:

Research Hypothesis – *It is possible to support policy-driven adaptation of service request QoS levels in a lightweight, efficient and application transparent manner that provides improved resource consumption and satisfaction of goals and requirements in a mobile service application running in an environment with dynamically changing context.*

1.5 Research Contributions

In response to the aforementioned challenges, the primary contribution of this research project is the “Volare approach”, a client-side policy-based adaptation scheme for implementing in an application-transparent way dynamic context-aware adaptation to the commercial service request (SR) of a mobile application to an application-selected cloud service broker or provider provisioning alternative services of the same content at different QoS levels.

It aims to support the discovery of the most appropriate service found under the current context and the adaptation logic requirements, by adapting the QoS levels of the service request, then evaluating the offered service QoS levels, and either binding to it or adjusting the SR and launching re-discovery.

The Volare client-side approach is composed of three required and integrated constituent components:

- **The adaptation policy specification language (APSL)** that allows the developer to specify the adaptation behavior required.
- **The mobile DCAA support middleware that** implements the adaptation functionality itself.
- **The weight-based adaptation reasoning technique (WBART)** that guides the adaptation policy development process.

The characteristics of each constituent are outlined in §3.4 and are analyzed in detail in chapters 4, 5 and 6 respectively.

These components implement the Volare approach, which has the following innovative characteristics, compared to a more traditional situation-action adaptation approach, in order to better resolve the challenges identified in §1.2:

1. **Two-level Policy Support**

The Volare policy language supports a two-level architecture, using a global and and application level policies concurrently, thus allowing both device and application-specific adaptation of the QoS levels.

2. **Multiple rules selectable on the same adaptation-variable**

Multiple adaptation rules may be selected to adapt the same QoS variable at policy execution and all will participate in the final adapted value via the conflict resolution directives for participative weighted contribution, based rules weight values specified in policy development. This enables the following innovations.

3. **Building the Policy in Adaptation-Strategies**

Determination of the major adaptation-concerns around which the policy will be built, and building it as a set of independent adaptation-strategies, each of which is the collection of policy rules that serve a specific adaptation-concern over the valid context domain. This approach may lead, when there is high behavioral variability, to policy authoring significantly shorter in the number of rules and simpler in testing and verification than an equivalent “action-based” policy (see ch. 5 and §8.3.2 for detailed analysis).

4. The Volare Dynamic User Preferences Model

A policy-based dynamic user preferences model for adaptation logic fine-tuning by the user, allowing a degree of dynamic control to the user over the adaptation behavior without any need for software or policy update.

5. Support of Unanticipated Quantitative Long Term Performance Goals

Volare supports introduction in the policy of unanticipated at middleware design time quantitative long term performance goals (LTPGs) with finite horizons over many binding sessions, allowing for long term resource management.

6. Offered Service Evaluation and SR Adjustment & Rediscovery

The middleware supports offered service QoS terms evaluation on policy-based parameters and hard-coded criteria. If the offered service is not assessed appropriate, then the SR is adjusted and rediscovery is launched until a satisfactory service on QoS terms is discovered and bound to.

7. Support of Hierarchic Multi-cycle Policy Execution

Optional specification of multiple “consecutive execution cycles” at policy execution is supported, thus allowing hierarchic selection of variant(s) at one cycle and of the relevant parameter settings in the next cycle, enabling more concise policy authoring in high behavioural variability scenarios.

A use case and a proof-of-concept implementation have been developed on cloud service discovery through a cloud service provider, as well as a simulation case study based on usage data, demonstrating (Table 8-2) significant monthly savings in battery consumption (average lowest battery level of 18,4% unadapted vs 27,2% in the adapted case, in 10 discharge cycles), provider data usage and monetary cost compared to very shallow adaptation of QoS service bindings, while consistently avoiding service disruptions caused by QoS levels that the device cannot support. The standard reference case presented in Fig. 8.15 achieved, by the end of the monthly period, 95.5% of the credit allowance vs.

105.9% of the very shallow adaptation version (still very near the goal value, under the circumstances) and data use ratio respectively 87.8% vs. 91.1% for a similar very shallow adaptation application, in a typical usage model. In extreme cases of context variations, these results raise dramatically, as seen in chapter 8.

In addition, adaptation policies using the Volare approach tend to increase in size, due to possible context states, in a mostly linear fashion, instead of the combinatorial increase of more conventional situation-action approaches. This leads to smaller adaptation policies in cases as the number of possible context states increases. In our case, with 142 adaptation-rules of the Composite Policy were covered the adaptation of 864 adaptation states.

1.6 Thesis Outline

Chapter 2 presents the current state of the art for the mobile devices on Service Discovery and Binding to services on the Web and the Cloud and Related Literature.

Chapter 3 introduces a Motivating Example demonstrating the research challenges and we outline the ideas and requirements for this research project, highlighting the main innovative characteristics of the Volare project.

Chapter 4 introduces the novel Adaptation Policy Specification Language that is the heart of the weight-based approach, the Conflict Resolution Directives with the Participative Weighted Contribution Procedure and the Consecutive Execution Cycles feature of the Policy Engine that supports hierarchic policy execution.

Chapter 5 presents the novel weight-based adaptation reasoning technique for adaptation policy authoring, as well as the relevant methodological tools specially developed for this purpose, to assist the developer in policy design according to the Volare approach, including Policy Testing & Verification.

Chapter 6 describes the conceptual model for the DCAA middleware and the UPI application for user preference selection and policy customization.

Chapter 7 outlines the implementation, testing and validation issues for the middleware as well as for the adaptation logic that supports policy-based Long Term Performance Goals over finite horizons.

Chapter 8 presents the critical evaluation of this research project as a whole and on each of the three individual constituents of the Volare approach, both qualitatively and quantitatively.

Chapter 9 finally outlines the contributions of this research work in the fields of adaptation reasoning and policy-based DCAA for mobile computing and explores directions for future work.

1.7 Related Publications

1. Papakos P, Capra L and Rosenblum DS. Volare: Context-Aware Adaptive Cloud Service Discovery for Mobile Systems. ARM 2010: 32-38.
2. Papakos P, Rosenblum DS, Mukhija A and Capra L. Volare: Adaptive Web Service Discovery Middleware for Mobile Systems. ECEASST 19 (2009).

2 Background

2.1 Scope of mobile devices

Referring to mobile devices in this work, we mean mobile (handheld) smart-phones, palmtop computers and personal digital assistants (PDAs) that communicate through a wireless or mobile communications network (WLAN, 3G, EDGE, GPRS) and are power supported by their battery. It is further assumed that there is one user per mobile device that sets preferences and choices [46].

Typically, the mobile devices pose three significant limitations to software developers.

Mobile devices have limited hardware resources (CPU, RAM, memory, battery power, storage memory etc) and computation capability, while some of them are consumable over recurring periods.

In addition, mobile devices are subject to frequent environmental context change due to mobility, for instance local network availability or bandwidth or gps coordinates etc., due to movement and/or network load.

Finally, there is the issue of variable user preferences, to ensure user satisfaction during Service Discovery.

2.2 State-of–the-art for Service Discovery by Mobile Applications

These factors lead to the need for developing appropriate software engineering solutions to ensure satisfactory performance for mobile applications launching Service Discovery on internet services, issues discussed in detail in the following subsections.

In particular, applications that require long lasting service binding, which include transfers of large amounts of data or of longer duration, such as data-syncing applications, media-streaming or navigation and video-calling or tele-conferencing applications (like GoogleTalk, Skype or FaceTime) are negatively impacted by fluctuations in available resources and network bandwidth [36][54][69]. In addition, such applications may rapidly deplete the device’s consumable resources, if they continue binding on the initial or nominal QoS levels when not able to support them.

We define below basic concepts to be used throughout this work:

Context - While many definitions of context are available in the literature, the most widely cited one is*: "Context is any information that can be used to characterize the situation of an entity; an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves"* Dey 2001 [57][34].

In this work we distinguish context in: device resource parameters (like available CPU level, storage memory, RAM or battery power), environmental parameters (such as network availability and bandwidth, location, speed, noise, illumination etc.) that may vary at runtime, and user preferences (such as preferences on billing limits, alert settings, minimum QoS settings, etc.) that may also change at runtime.

Context-Awareness – "A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task" Dey 2001 [57].

Context-aware adaptation – we mean the ability of a software component to modify itself in response to context change without user interaction. Mobile applications in the dynamic execution environment have to be context-aware, and able to adapt to context changes [6].

Thus, a major challenge in mobile applications development is **the varying context of mobile systems** that may change at runtime [9][13][54]. The context may include limited or consumable device resources (like available CPU level, storage memory, RAM or battery power), environmental variables (such as network availability and bandwidth, location, speed, noise, illumination etc) that may vary at runtime usually due to mobility, and user preferences (such as billing limits, alert settings, minimum QoS settings) that may also change at runtime.

Extensive research [13][18][30][34][54][56] has identified a series of possible solutions that the current state-of-the-art utilizes to address these issues:

1. **Context-Aware Adaptation at initial Service Discovery**

At a new session of Service Discovery, the Service Request QoS levels need to be specified not at default values but in context-aware manner, in accordance with the current context and the adaptation logic.

2. **Dynamic Context-Aware Adaptation**

The change of context of a mobile device during runtime may also create serious performance problems, as well as inefficient use of device resources. This happens because in such cases of change of device resource levels and/or other parameters like bandwidth, the quality of service (QoS) levels initially requested by the mobile client application may not actually match the current capabilities of the device to receive and process data, resulting delays, performance degradation and user dissatisfaction, device resource wastage and unreliable web service binding.

Extensive research has shown that, especially for long-lasting applications, dynamic context–aware adaptation is necessary to ensure efficiency [18][30][34][36][42].

3. Middleware-based Dynamic Context-Aware Adaptation Support

Dynamic context-aware adaptation may be applied either internally in the application or externally through a DCAA-support middleware. It is shown that the external middleware-based approach makes mobile application development much simpler, while on the same time it can be used for other adaptive applications.

Typically, for policy-based context-aware adaptation of mobile applications a middleware-based architecture is adopted, where the middleware layer provides at least the first two and in some cases all three of the following functionalities [18][34][45][61]:

a) Context-awareness through a Context Monitoring Manager, collecting, aggregating and making the context available to interested components.

b) Adaptation reasoning through an event-based Adaptation Reasoning Manager on the eventual adaptation actions when context change requiring adaptation occurs.

c) Adaptation Implementation, answering to the question: “where to adapt?” [30], i.e. the implementation of adaptation actions selected in response to context change is realized either by the middleware itself as it is often the case of distributed communication adaptation [6][30] or by the active application concerned [7], or by both the middleware and the adaptation [20].

4. External policy-based adaptation logic

The adaptation logic may be either hard-coded in the application or external to the application, in the form of policy. Since today’s software requirements change rapidly, there is a need for easy and frequent updates of the adaptation logic of an adaptive system. Research in the field indicates that policy-based adaptation logic, updatable either at load-time or even better dynamically at runtime, is a most useful feature in the state of the art for context-aware adaptation for mobile systems [6][7] [18] [34][61][80].

The adaptation logic for an adaptive system is developed through the use of two required tools: an Adaptation Policy Specification Language and an Adaptation Reasoning Technique.

The Volare approach conforms to the state-of-the art described above, by providing the three above referenced constituents: a mobile DCAA-support adaptive middleware with the required context-awareness and adaptation reasoning functionalities for SR adaptation and for authoring the adaptation logic, the Volare APSL and the weight-based adaptation reasoning technique.

2.3 Policy Specification Languages

Policy-based systems require a Policy Specification or Description Language to formalize the adaptation policies. A general purpose or domain-specific Adaptation Policy Specification Language for policy authoring is required, defining the rules form after the Event-Condition-Action or simply the Condition-Action model, a rule priority assigning mechanism, the Conflict Resolution Directives [34][12][44][39], keywords and operators.

Several such formal languages have been developed within the last decade for mobile and distributed systems: Jess [35] is a policy description language on Java with an inference engine shell used in Hydrogen [8]. Policy Description Language PDL [37] describes strategies for mapping a series of events into a set of actions. PONDER [12] is a Policy Language that may be used both for management and security policies in distributed systems. The Event Calculus [44] presents a method for transforming both policy and system behavior specifications into a formal notation that is based on event calculus.

Alternatively, semantic Web Service Languages have been developed supporting well defined semantics: KaoS [39] presents a policy-based agent management approach. Rei [38] is a policy language based on deontic concepts.

Finally simpler and shorter domain-specific Adaptation Policy Specification Languages (APSL) to enable policy authoring for mobile or pervasive and ubiquitous policy-based CAA middleware or frameworks have been developed in the last decade, that we will discuss in brief as being closer to our work, such as the APSLs used in CARISMA [6], CHISEL [7], QuAMobile [19], RAM [24], MIMOSA [36], RAINBOW [59], CARE [69] etc.

In the Volare Approach, a domain-specific APSL for the field of Service Discovery has been developed with the aim to support more expressive policy specification. It is endowed with several unique characteristics, like Conflict Resolution Directives allowing possibly multiple rules on the same adaptation-variable to be selected at policy execution and influence the adaptation results through a participative weighted contribution procedure. The Volare APSL is presented in Chapter 4.

2.4 Main Adaptation Reasoning Techniques

Policy-based DCAA systems require adaptation policies externally to the middleware or applications. Three main methodologies have been proposed for the description of such policies [18][34][6][19]]22][26][61].

1. The Action-based Adaptation Reasoning Technique

Situation-action approaches specify exactly what to do in each context situation, using a separate behavioral policy for each contextual configurations. Action-based specifications are undoubtedly the most popular and are used in different domains related to networks and distributed systems [18].

The *action-based* or *situation-action* approach specifies exactly what to do in certain *situations*, requiring the explicit description of each situation as a context sub-domain through a set of predicates and specifying the required adaptation actions (i.e. the rule action) in response to the context changes (i.e. the rule condition) [7][24].

Action-based policies are popular with developers because they offer specific advantages: (a) they are expressive (i.e. easy to write); (b) they do not need to define the possible system states (i.e. the possible system's configurations and behaviors), developing a formal state model of the system at design time, like in the Goal or Utility-based approaches; (c) they give a fast system reaction to context changes, where the system adaptation actions are already defined, without the need to re-evaluate the system state model. However, (a) this approach uses simple rules that fail to catch some dependencies between adaptation and context; (b) defining the specific system reactions to the context changes at each "situation" during the design time may also be difficult in large scale systems with large number of adaptation behaviors; (c) an action-based policy is difficult to be fine-tuned or modified and then a full policy review will be necessary to avoid potential erroneous execution [18][34].

2. The Goal-Based Adaptation Reasoning Technique

“Goal-oriented approaches represent a higher-level form of behavioral specification that establishes performance objectives, leaving the system or the middleware to determine the actions required to achieve those objectives. Goal specifications capture the relations between context and adaptation mechanisms in a concise way” [18].

They specify the possible system’s configurations/behaviors as states. These states are used to build a state-based model for the system’s adaptive behavior, where the transitions between these states are enabled by the context changes. Goal specifications capture the relations between context and adaptation mechanisms in a concise way.

However, goal-oriented approaches also have some drawbacks: (a) they fail to catch dependencies between adaptations and goals, and conflicts between goals; (b) they do not provide any mechanism to compare adaptation actions when several actions can be applied to achieve a goal; (c) when the number of the context variables becomes large, the state explosion problem happened; (d) even if the model does not have the state explosion problem, the enumeration of all possible system states is difficult and may be impossible; (e) comparing to writing the condition-action rules, the building of state based models is difficult; (f) furthermore, computing the required adaptation actions at runtime causes a significant overhead to the system, which affects the system performance [74][75][34].

3. The Utility-Based Adaptation Reasoning Technique

“Utility-based approaches extend goal-oriented approaches. Utility functions ascribe a real-value scalar desirability to system states (i.e. in our case, a state is an application variant). The middleware computes the utilities of variants and selects the variant with the highest utility.” [18]

Utility functions express the rationale for adaptation decisions in a precise way, and are therefore more appropriate than goal policies when adaptation triggers and effects interfere, or when goals are in conflict [6][18][26].

However, they have the drawback of being highly complex to develop when there is a large number of context variables that are used to define the utility functions. Additionally, the utility-based approach has problems similar to the goal-based approach such as the need to enumerate all the possible system states at design time and the runtime overhead where the utility functions are computed at the runtime [34].

4. Evaluation of Adaptation Reasoning Techniques

The Goal-based and the Utility-based techniques typically impose a high computational burden on the restricted capability and resources of the mobile devices, depending on the scenario complication (although CARISMA [6] demonstrated its feasible application for stand-alone middleware solution on mobiles) [34]. Additionally, they require a tedious policy development procedure for calculating a mathematical state-based model for the context-aware adaptation of the system or the service.

The action-based technique is most widely used in relatively simple mobile applications as it is relatively less tedious for the developer and imposes the smallest computational overhead on the system in comparison to the other two. It is typically built by breaking the context domain into contextual "situations" (i.e. context sub-domains), assigning to each situation appropriate adaptation actions through adaptation rules.

"A limitation of action-based approaches is the imposed binary decision logic since each rule may be selected and executed or not. This limitation becomes more obvious in dynamic environments and may lead to low coverage of the context value domain" [34].

Which **Adaptation Reasoning Technique** will be selected for a mobile middleware-based (D)CAA application depends on a cost/benefit analysis based on the adaptation scenario and requirements, the adaptation alternatives available and the adaptation mechanism (parameter adaptation, selecting the most appropriate among a fixed number of alternative configurations or dynamic service composition).

The weight-based adaptation reasoning technique developed by the Volare approach for policy authoring is a rule-based one on the condition-action paradigm and consequently it is comparable to the action-based technique. However, it makes use of the Volare APSL characteristics, allowing at policy execution possibly multiple rules selectable on the same adaptation-variable, thus allowing rules from different "adaptation-concerns" on the same adaptation-variable to influence the adaptation results, depending on their weight value – expressing their relative importance. The weight-based technique is presented in Chapter 5.

2.5 Mobile Middleware/Frameworks on Service Discovery

There has been extended work in the last decade on mobile dynamic context-aware adaptation of mobile applications on service or resource discovery and binding to wireless internet services, some of which are referenced below. These projects use either a middleware deployed on the mobile device or a distributed framework both on the client device and on server(s).

Concerning the object of adaptation implementation, it takes place either: (a) on the middleware or framework components - like HERA [31], design approach on Volare [16-17], Q-CAD [15], ODYSSEY [40], ReMMoC [23], or (b) on the components of the custom-made adaptive application, like: MADAM [18], QuAMobile [19], QuA [22], RAM [24], MUSIC [26], PLASTIC [25].

Specific reference of such mobile DCAA middleware and frameworks and brief comparison to Volare is presented in related work §6.5.

2.6 Cloud Service Discovery by Mobile Applications

Cloud computing enables convenient, on-demand network access to a shared pool of configurable computing resources, such as networks, servers, storage, applications, and services, which the cloud system can rapidly provision and release automatically [109]. Given the explosive development of cloud computing in recent years, it was only a matter of time before mobile device application developers and cloud-service providers began utilizing the new capabilities offered by this approach for service provisioning [1][4].

Although context aware adaptation for mobile devices on the web is a highly researched area, very little research has been done on mobile applications implementing service discovery & binding on the emerging sector of cloud services, despite the trend by service providers to move their premises on the cloud [1][5][21][110-120].

Cloud Service Models

There is particular interest in the commercial applications of cloud computing [5], through alternative service models:

Software as a Service (SaaS) – The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface.

The SaaS layer generally exploits the Service-Oriented Architecture (SOA) technology because SOA and Cloud computing coexist, complement and support each other [109][111].

The Volare project in its current version focuses only on the SaaS service model of cloud computing (i.e. application-level service discovery).

Platform as a Service (PaaS) – The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.

Infrastructure as a Service (IaaS) – The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.

Cloud Deployment Models

Different cloud deployment models are used, following different business models, differentiated on whether they provide access to all interested users or only authorized ones (private or community clouds) [105]:

Public Cloud – The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

Private Cloud – The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units).

Community Cloud – The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations).

Hybrid Cloud – The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Main Actors in the Cloud

In the scope of our work, we can emphasize from the figure 2-1 above by NIST [105], the role of the following main actors in cloud computing:

Figure 2-1 – The Cloud Conceptual Reference Model by NIST [105]

Figure 2-2– Usage Scenario for Cloud Consumer – Provider [105]

The cloud consumer is the principal stakeholder for the cloud computing service. Service discovery on the cloud is implemented by a service request either through a cloud provider or through a cloud broker. The cloud consumer browses the service catalog from a cloud provider, requests the appropriate service, sets up service contracts with the cloud provider, and uses the service. The cloud consumer may be billed for the service provisioned, and needs to arrange payments accordingly.

2.6.1 SaaS Cloud Services

Applications or services offered by means of cloud computing are called cloud services. The user of a cloud service has access to the service through a Web interface or via an API. A benefit of the approach is that it enables clients getting service on a pay-as-you-go basis and selecting cloud services based on the price and other criteria such as QoS. The net benefit for consumers and mobile users in particular, is the ability to receive better services tailored to their current needs [105].

SaaS represents the trend of the future and the most common form of cloud service development. With SaaS, software is deployed over the Internet and delivered to thousands of customers. Using this model, the cloud service provider may license its service to customers through a subscription or a pay-as-you-go model. The service is then accessible using an API [121][5][109].

2.6.2 Commercial Web Service Providers Move in the Cloud

One of the underlying advantages justifying the deployment of commercial web service providers in the cloud is the economy of scale. By making the most of the cloud infrastructure provided by a cloud vendor, a service provider can offer better, cheaper, and more reliable services than is possible within its premises.

The cloud service provider can utilize the full processing and storage resources of the cloud infrastructure if needed. Another advantage is scalability in terms of computing resources, since service providers can scale up when additional resources are required as a result of a rise in the demand for their services. Conversely, they can scale down when the demand for service is decreasing [121][5][104][114].

New software developers no longer require large starting capital to distribute their services. Providers no longer have to commit resources to provide a service that may prove unpopular. Clients no longer need to be concerned about usage spikes, requesting over provisioning, as the cost is calculated by the resource usage. Thus, the overall cost of hosting distributed applications of dynamically changing usage levels decreases dramatically [53]. It also follows that services based on cloud resources are more reliable, as they are able to cope with usage spikes, as cloud resources are assumed to be practically infinite, and instantly scalable. These advantages present particularly useful opportunities for service providers for mobile and non-mobile applications [1][3][21].

As a consequence, SPs operating in the past as commercial web services - move their premises on the cloud, like: FLIXT for video and movies, OnLive or XBOX Live for online gaming, PANDORA for music audio-streaming, etc. [48]. This is the main reason that we selected to focus our project on service discovery by mobiles on public cloud services, that seems to represent the future of the overwhelming majority of commercial application-level services.

2.6.1 Cloud Service Brokerage

The NIST identified in its Cloud Computing Reference Model the cloud broker actor, which is in charge of service intermediation, service aggregation, and service arbitrage [105].

As cloud computing technology matures, cloud services offers are proliferating at an unprecedented pace. As in every business with a delivery model, such as real estate and insurance, cloud services' brokerage is expected to emerge in order to enable organizations to procure cloud services efficiently [121][21]. Indeed, finding the right cloud service is not an easy task for service consumers given the plethora and the variety of cloud services offerings. Dealing with a cloud service provider requires knowledge of its operating environment, the availability of management tools, its security levels and data recovery approaches, and the service terms and conditions. Collecting this information for multiple cloud service providers is likely to be a demanding task that is expensive and time consuming.

A cloud consumer may request service from a cloud broker instead of contacting a cloud provider directly. The cloud broker may create a new service by combining multiple services or by enhancing an existing service. In this example, the actual cloud providers are invisible to the cloud consumer and the cloud consumer interacts directly with the cloud broker [105].

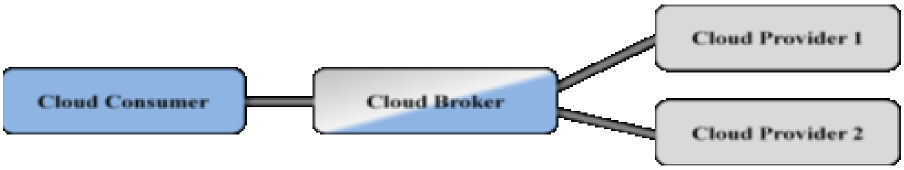


Figure 2-3 - Usage Scenario for Cloud Brokers

Cloud service brokers (CSBs) with their know-how and value-added services will assist service consumers in finding appropriate cloud service offerings, carrying out the QoS Terms negotiation process, monitoring and assessing implementation.

Cloud services, much like web services, are typically accessed using brokers, which will mainly be in charge of the management of the utilization, performance, and delivery of cloud services. CSBs will broker relationships between service consumer and multiple cloud providers [121].

Figure 2-4 – Broker based Cloud service provision [5]

In this figure (2-4) by Buyya, R., Yeo et al [5], we can see the envisioned serviced based market model for future Cloud services. The brokers act as intermidierie betweens the users (1..N) and the plethora of services around them. These brokers allow users to submit a service request, including required QoS levels for that service. They will then proceed to bind the best matching QoS terms with the cloud provider for the requested service and QoS levels [4][5][121]. It should be pointed out that this paradigm is still somewhat theoretical, and has not been applied uniformly to the Cloud industry as of yet.

2.6.2 Cloud Service Discovery by Mobile Applications

Service discovery on the cloud presents to the mobile applications developers its own particular challenges outlined below, in addition to the DCAA challenges for mobile systems already described in §2.2, namely:

1. **Monetary cost of service provisioning on the Cloud**

Cloud services present the specific feature [1][3][4][49][50][51] of imposing monetary costs on their clients - typically on *a pay as you go basis*. This cost for use of services on the cloud, typically depends on the resources utilized and the QoS level at service binding. Consequently, a service request on the cloud should include the cost of binding as a primary selection QoS variable and a cost-based adaptation- strategy should be an integral part of a policy for service discovery and binding on services on the cloud, in contrast to web services.

2. Additional challenges on service discovery on the Cloud

Another major issue with service discovery on the cloud is that it usually passes through service providers or brokers (generally third party services), where there is limited control about the decision making process of matching a service request with the closest possible service.

Additionally, there are currently no industry standard protocols when it comes to advertising cloud services or QoS service levels. Thus, every service provider or broker on the cloud has its proprietary protocols.

There has recently been work on context-aware adaptation of mobile applications [4] with cloud service discovery, focusing on market analysis based adaptation of the request. It focuses on choosing the most efficient service by means of cost-analysis of available services in the market and does not take into consideration the context of the client device.

In [27] an approach is presented for mobile systems, by replicating the whole smart-phone image and running the application code in powerful VM replicas on the cloud for augmented execution, outsourcing the heavy load computation on the *cloud clone*. *CasCap* [66] is a distributed power management framework for mobiles, making use of *crowd-sourced context monitoring*, *functionality off-loading* and *adaptation as a service* on the Cloud on a clone or a proxy, adapting energy-demanding activities in relation to WAN node access and other considerations. The research efforts on Mobile Cloud Computing (MCC) are rapidly developed for mobile applications off-loading part of the computation on the cloud, like CloneCloud [27], CasCap [66], eTime [90], MAUI [96], [97] etc. MCC is outside the scope of the Volare project.

In [67] a context-awareness cloud-based service for profiling mobiles is presented, based on frequency of accessing services on the internet, with activity database storage and management on the cloud.

In [65] a distributed framework for context-aware service provisioning, by dynamically adapting on the provider side the offered cloud service to the mobile user context, through dynamic service composition at the provider side in the form of service. This work is on dynamic service composition on cloud provisioned services, but with a different scope than Volare.

Some new research is published on interactions of mobiles and cloud services. There is also significant interest in optimizing the service provisioning process by development of relevant approaches or broker models for clouds, since currently many cloud providers use proprietary standards for accessing their services, so a third party broker is needed to access multiple different clouds [2][5][121][110-111][114][116][120]. Most of these research efforts aim at optimizing service discovery for cloud services through a broker architecture or brokerage mechanism or service selection component on the cloud.

The Volare approach aims to achieve resource optimization for the mobile device by applying parameter adaptation of the service request on the client-side.

3 Project Overview

This work aims to address the research problem of how to establish an approach that can adapt the service request to ascertain optimal QoS levels to existing or independently designed mobile applications launching service discovery & binding on cloud services.

Extensive research work discussed in Ch. 2 recommends as state-of-the-art for parameter adaptation on mobile applications the approach of middleware-based DCAA adaptation with external adaptation logic. The challenges for the developer to provide middleware-based policy-driven DCAA support to existing or independent mobile applications implementing Service Discovery on the Cloud (CSD) can be seen as multi-fold:

- The general challenges of the resource and capability constrained mobile devices and bandwidth variability referenced already;
- The added adaptation dimensions of the cost of binding and of the Broker-specific QoS advertisement and communication protocols due to service discovery on the Cloud;
- The variable user preferences during runtime due to mobility or change of mood or of objective needs, that were unobserved or trivial in the stable high-resources & high bandwidth non-mobile environment;

The aim of this work is to provide adaptation support to relevant existing or independent applications (that are not custom-made to the middleware), without any recoding and with minimum cost for the application developer, thus improving resource consumption and efficiency in the mobile client and the service provider.

Despite the high expansion rate of cloud computing and the advantages of that it may provide to resource constrained mobile devices, as explained in Chapter 2, service discovery to the cloud has so far been scarcely explored by the research community. Broker based futures approaches are currently in development [121,122], suggesting further interest in expanding the cloud services paradigm.

Due to the interesting research challenges and future impact of services on the cloud to the computing community, along with the increasing number of smart mobile users, that leads us to focus our research project on the domain of service discovery on the cloud for mobile applications.

3.1 Motivating Scenario for Mobile Policy-based Adaptive Service Discovery on the Cloud

The Volare mobile middleware aims to provide policy-based DCAA functionality to independent mobile applications implementing cloud service discovery (CSD) to services by adapting their Service Request (SR). DCAA of the SR by the middleware ensures the most satisfactory Quality of Service (QoS) under the current context and the policy constraints, including binding to the most appropriate and cost-competitive cloud service discovered.

The adaptation should be implemented transparently to the applications without any recoding, simply requiring from the application developer to provide an application-specific adaptation policy file with mostly pre-declared context & QoS variables that includes the rules on the service request QoS variables that will guide adaptation.

3.1.1 Motivating Example – User Set Challenges

We describe below a motivating scenario: Evelin is a smartphone user subscribed to a Mobile Network Service Provider (MNSP) for a monthly contract of a specified maximum monthly data allowance (4 GBs in the case study) downloadable through the use of mobile internet (web and cloud) services. Evelin – or the MNSP – has also installed on the device the Volare DCAA-support middleware, as well as several mobile applications launching service discovery to services on the cloud, like: video-streaming, audio-streaming, navigation, portfolio evaluation, professional news videos, etc., with a specified maximum monthly cost allowance on data volume used for services on the cloud (in the case study: 5£/month).

Jacob, the developer responsible for the firmware and optimization of the mobile device, is aware that cloud services may potentially be operated on it. His interest is to ensure the best operation and most efficient resource allocation for his device, to ensure Evelin's satisfaction. He is aware of the capabilities and limitations of the platform he is working on, and thus wants to ensure that the adaptation behavior for QoS operations on that device reflects the capabilities of the device itself. Thus, he has installed the Volare middleware on the device, and wants to specify a global adaptation policy for it that will influence any future adaptation, since he cannot expect the application developer to be aware of all the intricacies of every possible client device his application will be used on.

Ronald is the developer of a cloud video streaming application. He wants to ensure that his application operates in the most effective manner possible, ensuring optimal performance for Evelin, while minimizing resource consumption from the provider side, which he is likely financially responsible for. He wants to be able to develop a QoS adaptation policy for his specific application that will alter the QoS levels requested, according to the context of the device, which he cannot predict beforehand. In addition, he wants to be able to do this outside of the application code itself, since it may already be developed, or the policy may need to be updated and calibrated. Most importantly, he knows the target devices for his application are running the Volare middleware, but he certainly does not want to develop a separate policy for each device, since he is not aware of the intricacies and limitations of each model. Thus, he wants to focus on developing adaptation behavior for his own application, trusting that the global adaptation policy of the device will ensure that the adaptation behavior will more accurately reflect its capabilities.

Evelin is informed on the default adaptation logic of the Volare middleware, the Long Term Performance Goals (LTPGs) and the customization capability, as well as the alternative generic user preference modes offered by the User Preferences Interface (UPI). Evelin's actions or expectations are presented below in various scenes of the mobile operation for service discovery on the cloud.

Scene 1 **–** *Customizing the default adaptation logic* **–** Evelin initially (and eventually at a later time) customizes the default adaptation logic concerning service discovery on the cloud, including the Long Term Performance Goals values, setting the actual MNSP contract value of 4 MBs/month for the monthly data volume allowance on the internet, and the 5 pounds/month value for the monthly cost allowance for binding to services on the cloud. He reviews other customizable parameters values of the adaptation logic.

Scene 2 **–** *Selecting alternative generic User Preference modes* – Evelin, at runtime of a CSD application, would like to select on-the-fly through the user Preferences Interface or leave selected the currently appropriate user preference mode (like: “Normal” or “HighQuality” or “LowCost” or “SaveBattery”), enforcing the selected policy-based generic real-life adaptation behavior on the active CSD session. For instance, when the highest attainable quality binding is sought, then the “HighQuality” mode is selected – without considering cost or battery level. On the contrary, “LowCost” mode may be selected for services where low QoS is acceptable at low cost. “SaveBattery” mode may be selected when Evelin sees that battery level is below 30% or 40% and expects extensive device use till recharging opportunity.

Scene 3 **–** *User-Transparent Dynamic Rediscovery at Runtime* – When Evelin launches a service request on the cloud, he expects the most appropriate service to be discovered and bound to. If mobility at runtime leads to bandwidth or resource drop, Evelin expects a user-transparent dynamic policy-based evaluation and eventual rediscovery and binding to more appropriate service, without being annoyed by instructions requests or long delays.

Scene 4 **–** *Intelligent Smartphone Operation on LTPGs* – Evelin expects that his only supervision duty, after having reviewed and customized the default settings for the User Choices Profile, should consist in recharging the device when it is at low battery level and occasionally selecting a more appropriate than the default "Normal" user preference mode. Evelin expects his smartphone to operate in an intelligent and user-transparent manner, monitoring context and performance on LTPGs, adapting when required in order to achieve the LTPGs over their horizons, without burdening him with guidance requests and/or options selection for appropriate service QoS levels. It is the Volare's task to achieve each LTPG by the completion of its time horizon.

Scene 5 **–** *Fine-Grained, Gradual Adaptation Behavior* **–** Evelin expects fine-grained, gradual and not abrupt adaptation behavior. For instance, if the monthly expenses rate for cloud services is estimated as overrunning a policy-based estimated threshold, then the middleware should transparently start selecting services gradually cheaper than usual, in order to bring back to normal the expenses rate, keeping costs within target over the monthly period. However, adaptation change level should be in relevance to the expenses rate discrepancy, in a gradual, fine-grained manner.

3.2 Establishing DCAA support for Cloud Service Discovery

As expressed in the state of the art in Ch. 2, Dynamic Context Aware Adaptation (DCAA) for mobile applications launching Service Discovery & Binding to services on the Cloud, require the following three basic constituents of CAA:

a) A Mobile Middleware supporting DCAA for the active application(s);

b) An Adaptation Policy Specification Language for authoring or editing the middleware and the application policies;

c) An Adaptation Reasoning Approach for Adaptation Policy development.

We will examine for each of these three required constituents for DCAA support, the challenges imposed by the research field chosen (Cloud Service Discovery by mobile applications). We additionally impose our own additional usability and development capability needs and we deduce the requirements for the design of each of the above three constituents.

3.3 Motivation for a novel Adaptation Reasoning Approach

In policy-based systems the adaptation policy logic is the driving force for satisfactory adaptation. Consequently, identifying the characteristics that an adaptation policy and the DCAA middleware/framework should support is a major software engineering challenge.

In this work we use the term “policy” or “policy file” to mean the whole set of adaptation-rules for the middleware adaptation or for an application adaptation. We also use the term “adaptation-variable” for each rule “head predicate”.

We restrict ourselves in comparison to the action-based adaptation reasoning approach, since the other two goal-based and utility-based approaches as described in §2.3, entail a heavy development and operational cost that may be justified only for more complicated applications, usually concerning dynamic service composition.

3.3.1 Challenges for a novel Adaptation Reasoning Approach

Critical issues are examined concerning the adaptation reasoning approach for policy design compatible to the Volare APSL innovative features, in order to be able to provide consistent and meaningful optimal (or near-optimal) adaptation results:

1. Policy Structured over Multiple Competing Adaptation-concerns

There exists a multi-faceted challenge on policy development that needs to be addressed. The conventional Action-based adaptation policy uses the “situation - action” approach with the limitation of just one selectable adaptation-rule for each head predicate at any “situation”. However in the adaptation process, there are multiple active, live, **adaptation-concerns** (adaptation interests), that need to be considered and “represented” in the “situation-action” relevant selectable adaptation-rules, for instance:

a) The interest to have the most satisfying application performance for the user under the current context, constraints and user preferences.

b) The interest in optimal use of resources like battery power, RAM, CPU etc.

c) The interest to have service discovery and binding to a service on the Cloud that satisfies the policy-based cost strategy under the current user preference.

d) The interest that service rediscovery and/or rebinding at runtime due to context change should only be allowed conditionally, on a strategy to consider the cost/disruption caused in user satisfaction by a rediscovery/rebinding vs. the benefit of re-adaptation.

We argue that it is difficult on one rule for each adaptation-variable to combine successfully all those adaptation interests, especially since they may vary with context even within the context sub-domain of a contextual “situation”.

2. Making Policy Editing, Modification or Enrichment Easy

In the conventional action-based adaptation reasoning, the policy consists of cautiously designed adaptation rules. If the developer decides to modify the policy on one or more adaptation features, would need to review the whole policy to identify the relevant adaptation-rules that need to be modified, removed or added and make the modification. This happens because the priority settings on adaptation-rules and the shared context-variables conditions make the Policy very complicated for modification and increase the risk of rule inconsistencies or other rule faults at policy execution.

For example consider the case where the adaptation policy has incorporated a specific behavioral attitude, like a cost-based service selection strategy for binding to a service or a battery power management strategy at service discovery on the cloud. At some time, the developer or advanced user decides to replace the specific strategy with a different one. In the conventional action-based approach, this would typically require a full scale effort to review the whole policy, identify adaptation-rules on different context conditions and possibly different head predicates that need to be modified or replaced and the need for full policy verification for danger of inconsistencies or rule faults.

3. Adaptation Policy Fine-tuning on Specific Feature(s) by the User

There is also the challenge of how the user may fine-tune adaptation behavior in an action-based adaptation policy, so that the application/middleware behaves differently on one or more specific feature(s), without intervening in the policy. For instance, the developer or advanced user decides to make the binding cost strategy stricter and/or set a more cautious attitude towards a resource like battery power use.

In that case, it is difficult and tedious for the developer of the action-based policy to identify the appropriate adaptation rules and a policy review is required, due to the fact that there should be only one selectable adaptation rule per QoS variable, and it must represent all relevant adaptation interests, including the possibility for fine-tuning.

4. Fine-grained, gradual Adaptation

A further issue with typical action-based adaptation stems from the fact that it is difficult to implement more gradual adaptation, due to the difficulties in accounting for multiple different context changes happening simultaneously. Thus, policy authors often resort to simply providing static QoS values triggered on specific context events, instead of opting for more gradual adaptation, that may lead to inefficient in final service request QoS levels [34][18]. In addition, at contextual situation boundaries, the adaptation tends to be abrupt, as another rule becomes selected.

5. Policy Optimization through Usage Pattern history

Assuming a single user for each mobile as it is typically the case, it may be possible to attune the initial adaptation policy to the mobile device usage pattern, thus optimizing the adaptation behavior on policy-based Long Term Performance Goals. This is much easier to implement for a policy designed around pre-determined adaptation characteristics, especially with the tool of weight values that may modify the adaptation behavior.

3.3.2 Requirements of a novel Adaptation Reasoning Technique

Thus a novel weight-based adaptation reasoning technique is introduced for policy development compatible to the Volare Adaptation Policy Specification Language (APSL) through eventually multiple adaptation-rules on each adaptation-variable. The requirement is the development of a new, simple, rule-based adaptation reasoning technique for policy development through the APSL, based on the following basic policy design tasks, preferably executed in an iterative pattern:

1. **Develop an adaptation logic that enables multiple adaptation rules per QoS variable, to express multiple adaptation-concerns.**

We argue that the policy would be more expressive, flexible and easy at authoring, editing or fine-tuning, if the developer identified at design-time the scenario major non-overlapping *adaptation-concerns* and defined the policy through adaptation-rules that express each adaptation-concern at every relevant contextual “situation”. Each different adaptation-rule would provide the adaptation action aimed to satisfy the adaptation-concern that it represents under the current context.

At policy execution, the eventually multiple selected adaptation-rules for each head predicate (one for each relevant adaptation-concern and adaptation-variable) the Conflict Resolution should derive the adaptation results through Participative Weighted Contribution of their execution-values and weight values.

Thus we conceptualize an adaptation policy structured around chosen major non-overlapping *adaptation-concerns* that express various adaptation viewpoints (such as performance optimization, resource use optimization, binding cost minimization etc) through multiple rules that affect the same QoS variables or middleware configuration variables. The balance in the adaptation results in the case of eventually multiple execution values is established through the weighted approach.

In addition, if the policy is structured on pre-defined adaptation-concerns, then modifying or replacing all or some from the set of rules on a specific adaptation-concern would be much easier, as they are immediately identified and the inconsistency or highly erroneous adaptation danger is much lower, because there is not just one selectable rule on each adaptation variable.

2. **Define a scenario-specific User Preferences Model.**

   This allows fine-tuning of the adaptation logic by the user for each relevant application and for easy on-the-fly dynamic user preference selection of the appropriate behavior pattern.

3. **Define Long Term Performance Goals.**

   By defining possible long term adaptation goals over finite horizons, we can achieve more efficient resource consumption over the long term. By combining this knowledge with statistical analysis of existing usage patterns, we can enhance the efficiency of the adaptation process, but planning ahead for expected pattering in the resource usage and environmental change.

4. **Define a Weight-Assigning Strategy (WAS)**.

   By developing a system for increasing or decreasing the importance of certain adaptation rules in a multi rule environment, we can achieve higher expressiveness and more gradual adaptation. We can implement this by assigning an appropriate weight function value to every adaptation-rule. The

WAS should facilitate the implementation of the User Preferences Model selected, either in adaptation logic fine-tuning by the user or in the user preference mode selection, imposing a different adaptation behavior.

3.4 Motivation for a novel DCAA Mobile Middleware

3.4.1 Challenges for a novel DCAA Mobile Middleware

When mobile devices are binding to Internet wireless services, context conditions at service discovery and context variation during runtime may lead to changing requirements from client mobile applications with adverse effects on performance, resource utilization and user satisfaction [18][30][34][54]. Variable network availability and constrained device resources may lead to disconnections or resource exhaustion. For this reason the Volare is designed as a client side, not distributed, middleware.

Since the mobile environment cannot offer QoS level guarantees, optimal operation corresponds to a dynamic compromise between what is technically attainable and the user preferences on competing and changing interests, like: performance, resource conservation, cost and disruption minimization etc. For this reason, the Volare approach introduces the User Preferences Interface (UPI) application and dynamic monitoring of the user preferences as described in the Volare scenario in §3.3, making it a crucial characteristic in adaptation and the middleware design and functionality.

Additional challenges for mobile application developers consist in the domain specific difficulties of service discovery on cloud services [4][5][21], namely:

- Service Discovery and binding to services on the cloud passes through Cloud Service Brokers, predominantly commercially based and therefore difficult to work with for research purposes.
- The Service Request Protocol to a Broker presents a significant challenge in the process, as there is still no industry standard protocol available, unlike in the web services domain. Similarly, there is no industry standardized QoS Advertisement Protocol. As such, it is difficult to establish a system where QoS levels from different providers are evaluated.
- Cloud Service Providers, due to the commercial nature of the Cloud [1][21], charge the user in a *pay-as-you-go* directive, according to the Provider resources utilized. Consequently the adaptation logic for binding on a service

on the cloud should necessarily be cost-conscious and a practical User Preferences Model should be supported.

3.4.2 Requirements for a DCAA Mobile Middleware

From our hypothesis in §1.5, the general motivation in §3.1, the scenario in §3.2 and the specific challenges in §3.3, hereby follows a brief description of features that constitute requirements that the Volare middleware should satisfy, based on the functionalities that is required to provide as described in the scenario.

Requirements on the functionality and operation of the middleware:

- Dynamic adaptation at runtime **–** The middleware shall support policy-based dynamic adaptation of the application Service Request, in response to context change at Service Request time and at runtime.
- Adaptive middleware – The middleware should be able to adapt its functionalities to context change to better serve its purpose.
- Application independent and application transparent adaptation **–** The middleware will not require the client application itself to be modified, or to be adaptive or to be custom-made to the middleware to facilitate adaptation.
- Support to the Volare two-level dynamic User Preferences Model, through a User Preferences Interface (UPI) application. The UPI will operate as a context multi-sensor, dispatching to the middleware the current user preferences. The UPI will operate at a generic level, independently to whatever user preference functionality each application may (or may not) have, by specifying short-term and long-term user wishes on performance, resources use, abstract level goals or disruptions due to rediscoveries etc.
- Context & Adaptation History Data Management and support to policy-based Long Term Performance Goals through statistical analysis functions on the context & adaptation history data.

Additional Requirements on the architecture and design:

- Client-side middleware - All components of the middleware should be on the mobile device to cope for robustness against loss of communication capacity due to mobility as well as to reduce the computational burden of a distributed framework and the cost of binding on the cloud. The computational and memory footprint of the middleware should be well within the mobile device capability.

- Middleware architecture in loosely-coupled task-specific modules for easy maintenance, upgrade possibility and extensibility.
- Reusable and platform independent components as much as possible, using an object oriented language for mobiles supported by several platforms, for all components not directly communicating with the OS of the device.
- Events monitoring and management will be implemented by the middleware, triggering policy execution for adaptation according to hard-coded criteria and policy-based threshold parameter values.

3.5 Motivation for a novel Adaptation Policy Specification Language

3.5.1 Challenges for a novel Adaptation Policy Specification Language

As briefly described in §2.4, there are various available domain-specific or general use Policy Description or Specification Languages for policy authoring with particular characteristics. However, in Volare we are interested not only in a simple and clear, declarative Adaptation Policy Specification Language (APSL), but also on some specific original features and we examine below the challenges involved.

Specifically we focus on a characteristic of the Conflict Resolution procedure in the Policy Specification Languages. The common thread in similar adaptation policies is to select a single adaptation rule for each QoS variable based on the context, which in general terms is a very reasonable requirement in order to prevent conflicts and avoid middleware and/or application malfunction**.**

However, if more than one adaptation rules were to be selected and executed providing their execution values but were assigned values of relative importance in the form of weight values, then an appropriate Conflict Resolution and Participative Weighted Contribution Procedure would calculate final adaptation results without inconsistencies.

A major research challenge for Volare is thus the design of such an Adaptation Policy Specification Language, by exploring the requirements and conditions for such an approach to be consistent, enabling authoring of adaptation policy logic with eventually multiple adaptation rules on the same head predicate selectable at policy execution, thus allowing contribution to the adaptation results by competing interests.

Another innovative characteristic sought for the Volare APSL is the support at the same time to two policies independently designed by different developers, for adaptation of the middleware and of the active application service request.

Additionally, we raise the issue that the APSL, optionally at the developer's decision, should support hierarchic policy execution in consecutive cycles to provide for dependencies by specifying at which cycle the rules for each adaptation-variable are allowed to be matched and selected. This APSL feature would allow in the first cycle(s) only rules selection and execution that identify the most appropriate mutually exclusive configuration variant (or variants if more than one sub-configuration exist with alternative variants) and then in subsequent cycle(s) rules selection and execution for the dependent parameters settings, until the maximum number of cycles is reached and all adaptation variables have new values.

3.5.2 Requirements for a novel Adaptation Policy Specification Language

Based on the above challenges, the following requirements are set for the Volare novel, declarative, simple and clear Adaptation Policy Specification Language:

1. **Eventually multiple adaptation-rules on the same adaptation-variable selectable at policy execution**

a) "Allow at policy execution eventually multiple adaptation-rules on the same head predicate to be selected and executed." The Volare APSL – contrary to most policy specification languages and the three main adaptation reasoning approaches described in §2.2 – adopts an innovative, weight-based approach:

b) By Conflict Resolution Directives allowing eventually multiple adaptation-rules on the same head-predicate to be selected and executed, thus providing multiple execution-values, each with a weight value representing its relative importance at adaptation calculation.

2. **Participative Weighted Contribution of the selected adaptation-rules to the adaptation results**

By Participative Weighted Contribution Directives for calculating the adaptation results. More precisely, the Volare APSL introduces two unique Conflict Resolution Directives:

a) “At the “Select Rules” step of the Policy Engine Cycle at policy execution, all the highest priority adaptation rules are selected for execution forming the Agenda, including possibly multiple ones on the same head predicate”. This happens in contrast to conventional on this feature approaches that allow only one adaptation-rule to be selected for each head predicate (adaptation-*variable).*

b) “At the “Execute Rules” step of the Policy Engine Cycle at policy execution, all execution-values of the selected and executed adaptation rules on the same adaptation-variable contribute to the adaptation results through Participative Weighted Contribution Directives: (i) the Weighted-Average Rule for numeric variables and (ii) the Majority Rule on the sums of weights for each non-numeric (Boolean or string type) adaptation-variable”.

3. A weight function value assigned to every adaptation-rule

The APSL should require at policy execution, the existence of a *weight function value* on every adaptation-rule. This weight function value should represent the *relative importance* of the rule in comparison to selectable rules on the same adaptation variable (head predicate) in calculating the final relevant adaptation result by the Participative Weighted Contribution Directives.

4. A simple and clear, declarative Adaptation Policy Specification Language

A simple and clear declarative Adaptation Policy Specification Language (APSL). The policy files can be viewed, printed and edited dynamically.

5. Adaptation-rules of the paradigm: condition-action

Simple rules of the form: If (conditions) – Then (action), neither of the “event-condition-Action”, nor of the “If-Then-Else” model, with “action” a single adaptation-statement assigning to the adaptation-variable (head predicate) at the LHS a value at the RHS of type numeric, Boolean or string and a weight value.

6. Two-level Adaptation Policy support

At Policy execution, a *Composite Policy* made of the middleware and of the active application adaptation policies will constitute the adaptation driving logic, with provisions ensuring the protection of critical adaptation-rules from Inter-Policy Conflict and overriding.

7. Support to multiple Consecutive Execution Cycles (CECs)

Support to developer-defined eventual multi-cycle policy execution, expressing the need for successive hierarchic configuration variant(s) selection or operation profile variants with dependencies, where it is necessary to select first one component or profile/algorithm or configuration or protocol variant before selecting at the next Policy Engine cycle the dependent parameters values.

All these requirements should be covered by the provisions of the Volare Adaptation Policy Specification Language, as will be further described in the features of the APSL in the following paragraphs of Chapter 3 and in the detailed APSL section in Chapter 5.

3.6 The Volare Solution

In this work we introduce the Volare approach for middleware-based DCAA transparent support to independent mobile applications launching service discovery for services on the cloud, for dynamic combined cost, resources and QoS level optimization of the service discovery, according to the operational requirements set in our hypothesis.

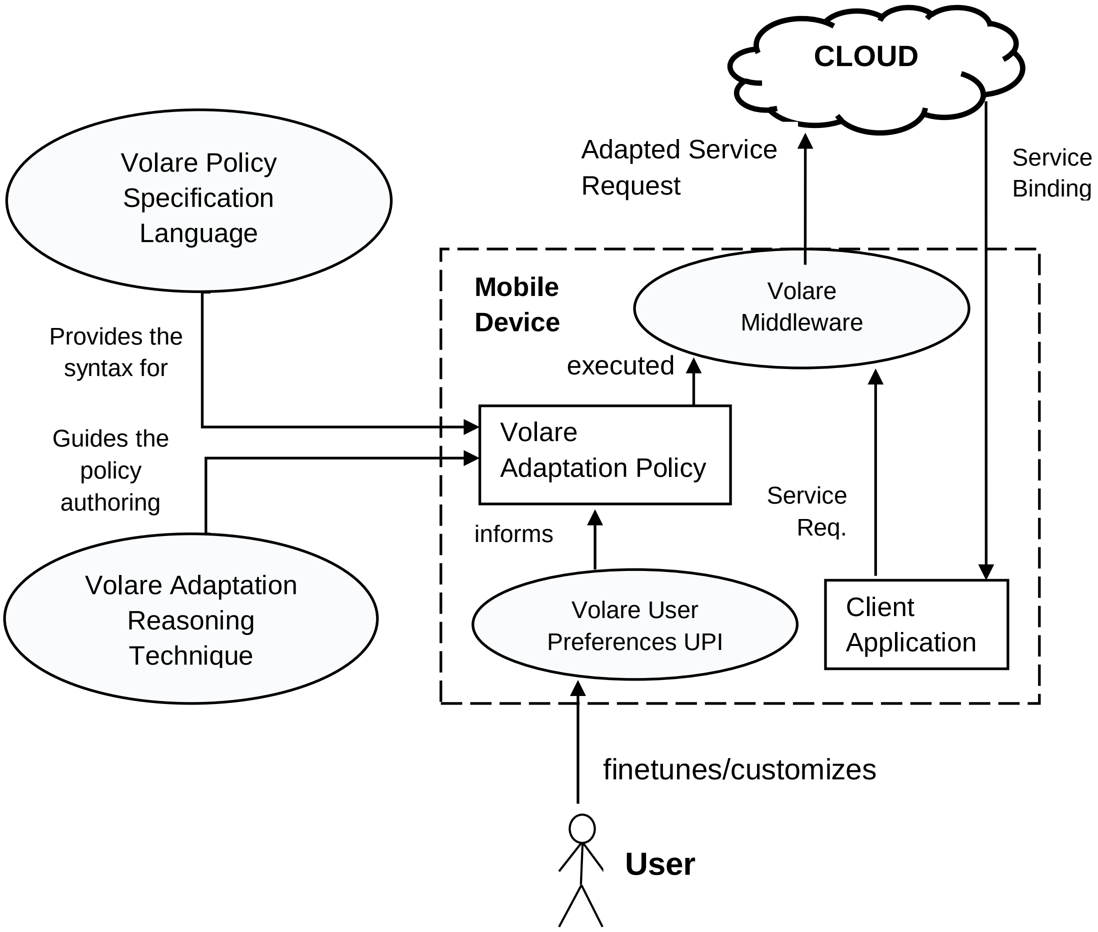


Figure 3-1 – The Volare Solution

As seen in the figure above(3-1), the Volare approach is composed of the following three required constituents:

1. A novel two-level Adaptation Policy Specification Language that enables the developer to define adaptation behavior for individual application, or global adaptation rules, by providing the rules and syntax.
2. A novel Weight-based Adaptation Reasoning Technique for policy development, providing the developer a systematic, consistent methodology for developing those adaptation policies.
3. A dynamic context-aware adaptive mobile middleware design, which implements the adaptation policies written above to perform parameter adaptation an application's cloud service request QoS levels. The middleware thus ensures that the request is properly optimized for the current context of the device, before forwarding it to the service provider.

In addition, the user of the mobile device must be able to influence the adaptation process though a User Preferences Interface.

Our research contributions for each of the above three constituents of the Volare approach are presented in the following paragraphs.

3.6.1 Middleware Operation

When a mobile application launches a service request on the cloud, the middleware intercepts and transparently to the application adapts the QoS levels of the service request according to the current context in a policy-based manner involving the policies for the middleware and the active application and dispatches it to the cloud Broker for service discovery.

Additionally, the middleware configuration is also adapted to the current context to better serve its functionalities, i.e. context monitoring, adaptation reasoning, adaptation implementation and the service discovery & binding functionality.

The Broker returns the QoS for the most appropriate service discovered. The middleware, based on policy-based negotiation limitations, either binds to the discovered service or re-adapts the service request and implements new service discovery.

At runtime, the middleware monitors the context to determine if adaptation is required due to context change according to policy-based criteria. If so, service rediscovery to the same Provider or rebinding to another more suitable Service Provider is implemented, adapting again the QoS request values as well as the middleware configuration variables to the new context.

Volare supports the User Preferences Model through a User Preferences Interface (UPI) application. To achieve optimal cloud service binding Volare attempts to discover the service providing optimal available QoS levels, including the added *binding on the Cloud cost dimension*. Since optimal service in the mobile environment is typically a compromise of what is technically attainable vs. available resources and user preferences, Volare introduces a User Preferences Model enabling dynamic declaration of a short-term user preference mode and of long-term choices that should be supported and satisfied by the middleware.

In this way, the Volare approach can utilize a User Preferences Interface (UPI), through which the user can:

a) Choose or change at runtime one among several alternative pre-defined policy-based user preference modes, thus modifying the adaptation behavior in a policy-based manner as the mode-title indicates;

b) F*ine-tune* or customize the default middleware adaptation behavior for each specific application supported, by setting the long-term User Choices Profile that modifies the default values of middleware adaptation policy parameters within pre-determined "safe limits" through the UPI*.*

3.6.2 A novel Weight-based Adaptation Reasoning Technique

The "weight-based adaptation reasoning technique" for policy development is introduced **–** based on the Volare APSL. Our specific contributions on the conception of the weight-based adaptation reasoning methodology are:

1. Policy structured around predetermined adaptation-concerns

The policy is structured around predetermined complementary and non-overlapping adaptation-concerns representing important qualitative goals that cover the adaptation-concerns space, such as: binding cost optimization, optimal resource use, performance optimization, delays/disruption minimization etc. For every adaptation-concern a group of rules is designed serving it across the whole valid context domain and is called adaptation-strategy.

2. The Weight-Assigning Strategy for weight value assigning to each rule

A Weight-Assigning Strategy assigns an appropriate weight value to each adaptation-rule representing at policy execution the rule's relative importance with respect to the other selected rules on the same adaptation-variable.

3. Integration in the Policy of the Volare Two-level User Preferences Model

Integration in the policy rules of an innovative Volare configurable User Preferences Model of the weight-based methodology, allowing through the UPI selection at runtime of a user preference mode triggering dynamic adaptation to the appropriate adaptation behavior, as well as global policy customization for each application through the User Choices Profile.

4. Weight-based Variant Selection for non-numeric Adaptation-Variables

Another contribution is the weight-based variant selection feature for non-numeric adaptation-variables at policy execution, through the weight values, expressing user preference. This is a unique feature not directly present in conventional action-based approaches, allowing variant selection through the weight values of the selected rules, indicating – under all other similar context conditions – the user preference influence in adaptation results. In this case the sum of weights for each variant is used as a utility function.

5. An Adaptation Policy Authoring Methodology

By developing an appropriate adaptation policy authoring methodology and appropriate testing and verification tools, we can enable the policy developer to easily author policies integrating the above weight-based technique, using the Volare APSL defined below.

3.6.3 A Two-Level Adaptation Policy Specification Language

1. Multiple Rules selectable on the same Head Predicate

Perhaps the most important contribution of our work consists in designing a simple and clear, declarative Adaptation Policy Specification Language (APSL), with unique features like weight value assigning to every adaptation rule built on the situation-action paradigm, so that at policy execution:

a) **Conflict Resolution Directives** allow eventually multiple rules on the same head predicate (the variable to be adapted) – representing different adaptation-concerns – to be selected providing their execution and weight values

b) **Participative Weighted Contribution Directives** enable calculation of the adaptation results without inconsistencies, through application of two common-sense rules on the execution and weight values: the **Weighted-Average Directive** for numeric adaptation variables and the **Majority Directive** on the sum of weights for non-numeric adaptation variables variant selection.

This unique APSL feature permits a much more expressive policy authoring, by allowing eventually multiple adaptation rules to be selected concerning the same QoS variable, contributing their execution values for the adaptation-concern they represent in the policy, at the relative importance assigned through the weight values.

2. Two-level Adaptation Policy Specification Language

The second important contribution for the Volare APSL consists in supporting two-level policies: the global policy for the middleware and the application policy for each application, allowing inter-policy and intra-policy Conflict Resolution, as well as provisions for the protection of critical rules of each policy from being overridden. The application adaptation policy has access to the context and global QoS variables of the global policy.

3.6.4 A dynamic context-aware adaptive mobile Middleware Design

1. DCAA Support to independent Applications on SD on the Cloud

This project includes the creation of a mobile middleware design to enable DCAA support to independent mobile applications that request service provision from the cloud, via the capability to intercept service requests to the cloud in an application transparent manner (without changing the application's code) and dynamically adapting it to better reflect the current context of the device.

2. Middleware Support of Two-level Adaptation Policies

Another original aspect of this work is middleware's capability to allow two different adaptation policies to affect the adaptation behavior. This is achieved by dynamically forming, at service discovery time, a "Composite Policy" from the middleware "global policy", which concerns adaptation logic based on the unique capabilities of the client mobile device, and the active service request "application policy", which is adaptation logic based on the application itself. This allows our adaptation logic to perform adaptation that takes into account both the individual needs of the mobile device and the application itself.

3. Support of Long Term Performance Goals

Predictive functionality allows for Long Term Performance Goals on predetermined characteristics by policy-based criteria through Machine Learning techniques evaluating the context & adaptation history.

3.7 Research Benefits

The benefits of this research work on mobile middleware-based systems to provide policy-based DCAA support to independent applications for the software engineering community beyond the scope of the current project are:

1. The Volare weight-based approach allows imparting DCAA functionality on the Service Request of existing or independent applications in an application transparent manner (without any recoding).

   The application policy development is a small cost for assigning policy-driven middleware-based DCAA functionality to applications with no adaptation capability. The classical approach middleware-based DCAA would require applications custom-made to the middleware or vice-versa.

2. The capability of the Volare APSL to allow multiple adaptation-rules concerning the same adaptation variable to be selected and executed at policy execution under the Participative Weighted Contribution Directives, provides the developer with much greater expressiveness in adaptation policy authoring, than the conventional action-based technique, through the possibility for multiple rules, each expressing a different adaptation-concern.

3. The two-level policy structure, combined with Volare's capability to handle multiple rules concerning the same QoS variable, enables platform specific adaptation in conjunction with application specific adaptation. It allows for the mobile device's firmware developer (via the global policy) to also influence the adaptation process to account for each mobile device's unique resource characteristics. This can increase the efficiency of the adaptation in cases where the application developer cannot predict the platform his application will be run on, which is common in mobile application development.

4. The adaptation-concerns and corresponding adaptation-strategies feature of the Volare approach also allows easy modification or customization of an existing adaptation policy on predetermined adaptation features, or easy substitution of a whole strategy on a predetermined feature with another, or addition of a new strategy selectable under specific conditions. This makes the task of authoring or integrating new policies in an existing policy easier for integrating new user requirements. In addition, it leads to significantly smaller adaptation policies in cases where mutually exclusive context variables must be considered, as it does not require developers to rewrite the entire adaptation policy for each eventuality.

5. The introduction of the Volare Two-level User Preferences Model permits: (a) the selection of a user preference adaptation behavior on the fly and (b) a User Choices Profile for middleware policy customization to each specific application, through a User Preferences Interface application.
6. Volare's weight-based technique capability to smoothly integrate multiple adaptation-rules also allows for fine-grained, gradual, highly expressive adaptation in comparison to the action-based technique.
7. The introduction of policy-based Long Term Performance Goals unanticipated at middleware design time, by the statistical analysis functionality of the middleware on pre-determined features in a policy-based manner of the usage pattern [10].

3.8 Research Scope

The following important topics are referenced in the Thesis but are not studied and are considered outside the frame of this project at this point

1. The use of weight values for uncertainty reasoning

For this project project, on the subject of weight assigning, we do not refer to the issue of uncertainty and fuzzy logic [56][76][34]. Instead, in our work we consider the context information trustworthy and not conflicting, especially since in our scenario we focus on a single mobile user and not on Pervasive Computing multiple context values of different trustworthiness from many sources.

Weight assigning to each adaptation rule at policy execution leads directly to assigning a relative importance to eventually more than one selected adaptation action on the same head predicate, due to different user desirability on each adaptation action originating from a different adaptation-concern.

2. Security Issues

The use of such adaptation policies for security functionality is a possible expansion to our work, however the more fluid nature of our weight-based functionality makes it less predictable than a more conventional action-based design. Thus, it is not a direction we examined thoroughly in this project.

3. Coordinated Adaptation for more than one active Applications

In the present work, assumption is made that only one application has an active service request for service discovery on the cloud at any time. Although this is typically true for foreground applications like video-streaming or navigation requiring the full user's attention, it may not be so for eventual background applications for service discovery on the cloud operating in parallel with a foreground application. The issues of coordinated adaptation and shared resources coordinated management may be the subject of future work.

4. Reflection

This work assumes the ability to inspect and reason about the local context (including resources, such as remaining battery power and bandwidth available) and to expose this information for adaptation reasoning of the service request of applications, allowing intelligent adaptation choices. Although related to dynamic context-aware adaptation, the reflection mechanism is not adopted, since the project aims to the adaptation of the Service Request of independent “black-box” applications on service discovery on the cloud and not adaptation of the applications themselves. Additionally, the middleware itself is self-adaptive.

4 Middleware Architecture, Design & Operation

4.1 Middleware Architecture

As it can be seen in Figure 4-1, the Volare middleware is situated on the client device. That client device then runs an application, which requests a service from the cloud. Volare will intercept the service request and adapt it according to the current context of the device. Volare’s architecture follows a modular approach, consisting of several task-oriented independent interconnected modules.

The middleware modules are as follows:

The Context Monitoring Module monitors the context of the mobile device. If there are significant deviations of the context from policy specified thresholds during runtime, the context monitor alerts Volare to re-evaluate whether the presently bound service satisfies the new requirements of the client based on its new context.

The Service Request Module intercepts the cloud service request sent by the client application to the service. It then forwards that service request to the Adaptation Module. This is a separate module because its implementation is, by necessity, platform dependant.

The Service Binding Module will decide according to the policy adaptation-rules to either send the adapted service request to the broker for rediscovery and rebinding, or to signal renegotiation with the existing cloud provider, if possible.

The Context & Adaptation History Module has the task to record the declared in The Composite Policy parameters data at every recheck period in the Context & Adaptation History Database (CAHiD) and its maintenance and eventual periodic data aggregation.

The Statistical Analysis Module has the task to implement policy-based statistical analysis function on the recorded data and provide statistically inferred information. It supports a number of pre-defined statistical functions like: Sum, Max, Min, Avg etc on a specified parameter data within a specified time period among several pre-defined “periods” of repeated activity, like Daily, Manthly, (battery) Dischargecycle, Session (on the Cloud) etc.

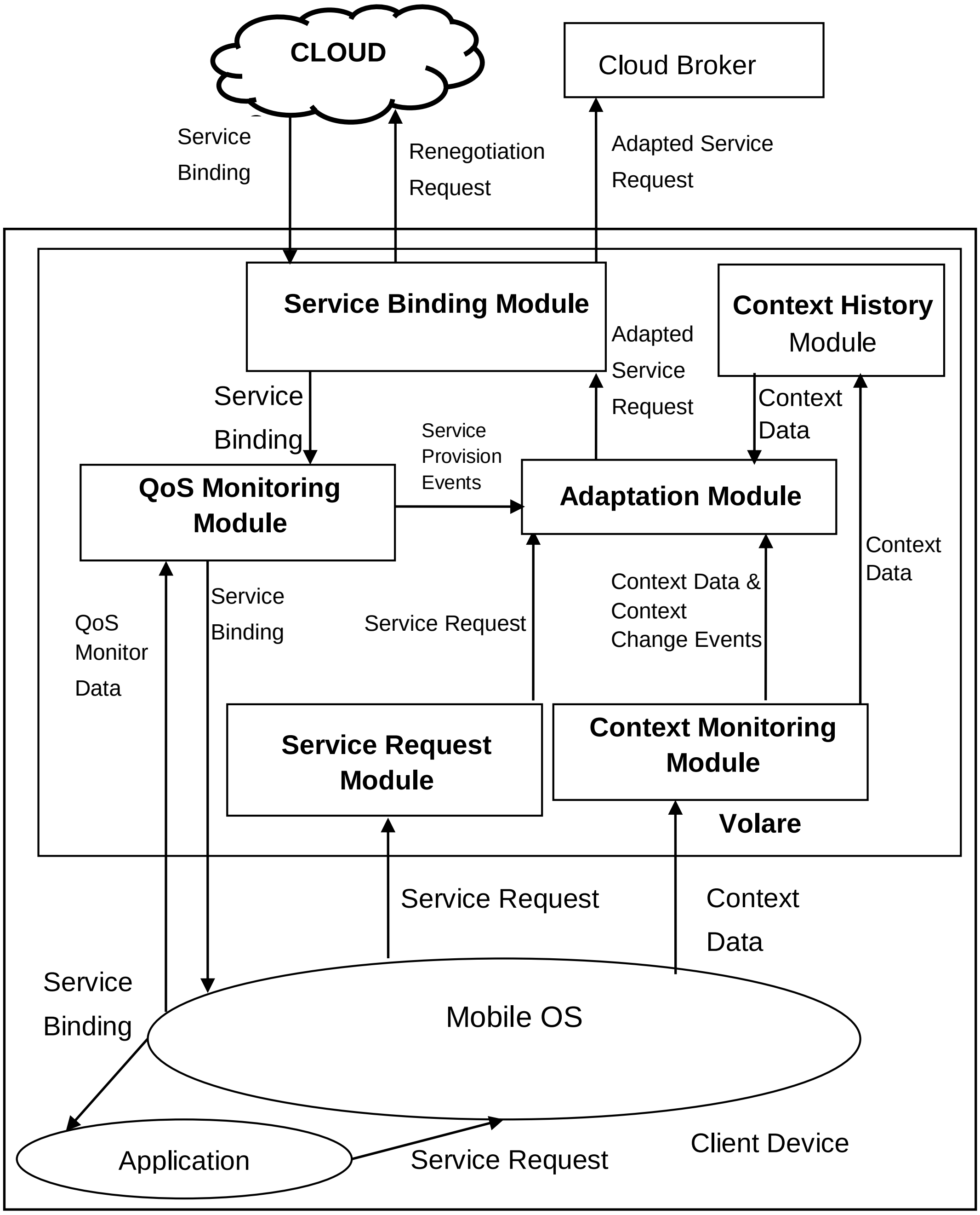


Figure 4-1 – The Volare Middleware Modules

The Adaptation Module handles the adaptation of the service request according to context and resource data or further adaptation based on the results of the initial search. It is then triggered by alerts from the Context and QoS Monitoring modules, to re-evaluate and possibly adapt the service binding at runtime. The Adaptation Module is the core Volare middleware module, charged with the adaptation reasoning and adaptation implementation. It is composed of the following task-specific components:

- The *Event Service Manager (ESM)* handles the middleware events and event driven operation, through a predetermined event-condition-action list. For any predetermined Event Requiring Adaptation (ERA-event), the ARM is notified to initiate adaptation.
- The *Adaptation Reasoning Manager (ARM)* is the middleware component that manages the Policy Engine operation, ensuring the input-output as well as policy execution through the Policy Engine Manager.
- The *Policy Files Manager (PFP)* is responsible for the maintenance of the Policy Files Repository, the loading of requested policy files as well as the parsing and merging them into the *Composite Policy*.
- The *Adaptation Evaluation Manager (AEM)* is responsible for evaluation of the new adaptation data at runtime, according to policy-based criteria, and deciding on launching or cancelling service rediscovery and/or rebinding, if the new adaptation data are not assessed as necessitating rediscovery, so as not to create user annoyance. For instance if the adaptation recommends a small, say 3% increase in bit rate QoS Request, due to some bandwidth increase, this quality improvement may not justify operation disruption and delay for a rediscovery.
- The *Adaptation Implementation Manager* is responsible for dispatching the new adaptation data to the corresponding modules/components for execution, in predetermined sequence.

This architecture enables Volare to be used with already developed applications without modifying the application itself. Instead, the developer only needs to provide a suitable adaptation policy file for the application. Volare achieves this by intercepting the service request after it has been initiated by the client application, and adapts it without the application's direct intervention. The policy is independent of the application itself.

A global and a scenario application policy file have been developed according to the Volare approach in chapter 7, providing the middleware adaptation logic.

4.2 The Middleware User Preferences Interface (UPI)

The UPI is a multi-sensor application connected to the Context Monitoring Module of the middleware and allows the global policy developer configuration of its menu, by assigning to each sensor a user preference context-variable, a range of values that it can take and high & low bounds for modification by the user if it is a numeric sensor.

It includes, in its current version, although this can be modified, two string type sensors for the *user preference mode* context element at 2 levels and at least 24 numeric sensors, that may be declared at the global policy as context-variables of the UPI and are assigned names, default values and allowed modification margins by the developer.

At every recheck period, the values of the UPI are dispatched to the middleware for use in the next adaptation.

Fine-tuning the middleware

The Volare *User Preferences Interface (UPI)* provides the user with the possibility to fine-tune or the middleware policy logic and consequently its adaptation behavior at runtime, at three successive levels of increasing complexity. It is operated by the user easily at the following two levels.

The **User Preference Selection level**, which is by default the active level, at which the user may on the fly select the desired *user preference mode:* “Normal”, “HighQuality”, “LowCost”, “SaveBattery” and eventually also the 2nd level *category*: “Business” and “Personal”. See also Figure 5.2.

The **User Choices Profile (UCP) level,** which allows the user to customize adaptation logic parameters, pre-determined by the global policy developer, defining either the generic UCP for all applications or for a specific application. Every application-specific UCP is saved and is loaded by the policy files Manager when the application launches a SR on the cloud.

The UPI allows the user to customize the values of global policy adaptation parameters, thus fine-tuning the adaptation behavior as follows:

a) It allows the user to modify the default Strategy-weight-coefficients at every user preference, within predefined “safe” limits from about 80% up to 125% of the developer’s default values, always keeping the modified weight coefficient value between 0 and 1. This customization of weight coefficient values, influences the relative importance of the adaptation-strategies and consequently the resulting adaptation behavior, enforcing or weakening an adaptation-strategy rules versus the other adaptation-rules.

b) The user through the UPI may modify the default values for pre-determined by the global policy parameters, as for instance the goal values for the LTPGs, like: monthly data volume or monthly cost of binding allowance.

4.3 Reserved Volare configuration-variables List

This is the list of adaptation-variables reserved as Volare configuration-variables, as well as their default values, that can be accessed and modified through both the global and the application-level policy and adapt the middleware modules to the current context, in addition to the middleware adaptive service of service discovery & binding functionality on the cloud.

However, the Volare APSL provides the keyword: "*overridesAll*", so that global policy adaptation-rules on critical configuration-variables may only be resolved at the global policy level and any adaptation-rules of the application policy on such configuration-variables are overridden. The application policy developer by reading the global policy identifies to which variables may have may access.

Volare Reserved Configuration Variables			
Variable Name	Description	Type	Default
defaultBindingMargin	The default binding margin all QoS variables will have when unspecified	implying percentage	10
recheckRate	The rate at which Volare will recheck the context of the device	integer, in seconds	20
rediscQoSThreshold	The threshold at which the discrepancy of the QoS levels of the existing binding and the calculated new QoS levels based on the current context will initiate service re-discovery and rebinding.	implying percentage	5
renegotiate	Will Volare attempt to renegotiate the request when it finds no satisfying services?	Boolean	True

renegotiationAttempts	Indicates how many attempts will be made to renegotiate the QoS levels required by the client device, in case the discovery engine fails.	Integer	5
discoveryAccuracy	Indicates how much the values of the QoS Offer terms may differ from QoS Request values to be accepted.	implying percentage	5
renegotiationAdjustment	Indicates how much a QoS Request numeric value may be adjusted at renegotiation.	implying percentage	12
rebindingRecheck	Will Volare wait before rechecking after rebinding, in order to avoid rebinding due to sensor spikes?	Boolean	True
rebindingDelay	How long will attempt to wait before rechecking after rebinding.	integer, in ms	10000
defaultPeriodicity	The default monitoring periodicity for any context-variable, for which no periodicity value ids defined	Integer	1
preferredModality	Indicates selection of one among several mutually exclusive variants	string type	

4.4 The Policy Engine Cycle

The Volare custom-made forward chaining Policy Engine operated by the Adaptation Module of the middleware, for adaptation logic authored conforming to the Volare APSL policy file syntax and the WBART, has specific characteristics that are analyzed below.

4.4.1 The Policy Engine Steps at Policy Execution Session

At a policy execution session the Volare middleware Policy Engine operates either in one single or in multiple Policy Engine cycles, depending on policy-based instructions. Each Policy Engine execution session is implemented in three consecutive steps:

1. **Match Rules step**

In the first state, **match adaptation-rules**, the Policy Engine finds all of the adaptation-rules that are satisfied by the current contents of the **Working Memory**. Since the Volare policies adaptation-rules are in the typical condition-action form, this means testing the conditions against the working memory of the current context data. The matching adaptation-rules that are found are all candidates for execution: they are collectively referred to as the conflict set. Note that multiple adaptation-rules on the same head predicate may appear in the conflict set if it matches different subsets of context data.

2. **Select Rules step**

The Policy Engine then passes along the **conflict set** to the second state, select adaptation-rules, applying a selection strategy to determine which adaptation-rules will actually be executed. This select adaptation-rules strategy is based on implied priority assigning on each adaptation-rule through APSL reserved keywords (overrides, yields etc), retaining only the highest priority adaptation-rules on each head predicate (i.e. adaptation-variable). These adaptation-rules will form the **Agenda**.

3. **Execute Rules step**

Finally the Agenda (the selected adaptation-rules of the highest priority) are passed over to the third state, execute adaptation-rules. The Policy Engine starts execution of the selected adaptation-rules, executing sequentially each selected adaptation-statement, deriving an execution-value for the adaptation-variable referenced, along with the values of any referenced attributes (binding margin and weight function value).

The execution-values from all selected adaptation-statements through the Conflict Resolution Directives are evaluated according to the Volare Participative Weighted Contribution mechanism to provide the resolved-value for each adaptation-variable, representing the adaptation results.

4. **Consecutive Execution Cycles at Policy Execution**

The Volare APSL, depending on the global policy file, allows a policy execution session to be executed either in one single or in multiple Policy Engine cycles, thus supporting more complex, hierarchic, consecutive calculation of the adaptation variables values. It allows the system to first select the most appropriate mutually exclusive variant among several competing structural, modality or algorithmic variants and then in the subsequent Policy Engine cycle to calculate the rest of the adaptation variables.

The global policy file declares and defines in the Declarations section the integer type calculation-variable **cyclesMax** that defines the maximum number of Policy Engine cycles. In simple applications when there exists only just one configuration variant in the global policy, then cyclesMax = 1.

However, if there are more than one variants and the Composite Policy includes weight-based selection of the most suitable one (i.e. by the Conflict Resolution Directives through the sum of weights for each variant), then more Policy Engine cycles will be necessary).

At the first cycle, the most suitable variant is selected. At the next Policy Engine cycle, use is made in the Criteria conditions of the known value for the selected structural or algorithmic or modality variant, for instance: *preferredModality = variantX*, thus selecting only the adaptation-statements under the already selected *variantX*.

If the Composite-Policy includes further sub-options, then successively more than two Policy Engine cycles may be necessary, until all adaptation-variables have been calculated in hierarchically successive Policy Engine cycles.

The application developer has either to conform to the maximum number of cycles cyclesMax defined in the global policy or introduce an adaptation-rule with the new **cyclesMax** value and width higher weight value than the default adaptation-rule, with a cycle declaration to be evaluated at the first CEC, so that the configuration-variable **cyclesMax** is adapted.

At the end of all Policy Engine cycles, the Policy Engine eventually provides the calculated resolved-values for each adaptation-variable, representing the adaptation results, to the Adaptation Reasoning Manager (ARM) for implementation, or rejection if the configuration variable: *implementAdaptation* is set to “N”, because the context and calculated QoS values do not differ substantially from the current values, and for storage by the CAHiD.

4.5 The Middleware Operation

A dynamic context-aware adaptive mobile middleware design that enables transparent dynamic adaptation of the service request of independent mobile applications implementing Service Discovery on the cloud, for discovery and binding to services on the cloud ensuring QoS levels that can be supported by the current context. It additionally supports of Long Term Performance Goals, through the Context & Adaptation History Module (CAHiM) for data history maintenance and aggregation and the Statistical Analysis Module that supports pre-determined classes of statistical functions on the recorded usage model, that are needed for the LTPGs management, as described in §3.5.2 and in subsection 6.4.

4.5.1 Middleware Operation when a SR is launched on the Cloud

At initial service discovery – at a service request (SR) on the cloud by a mobile application, the Volare middleware intercepts it, parses and merges the global and relevant application policy to a “Composite Policy” driving the adaptation and adapts the service request initial values of the QoS variables and the middleware configuration-variables to the current context. Then service discovery for the adapted SR is activated through the CSB or CSP for the most appropriate service under the current context. The CSB/CSP identifies a fitting service and submits a Service Offer with the provisioning QoS terms.

The middleware Binding Module implements Service Offer evaluation based on hard-coded mechanism and policy-based parameters and either accepts the offered service and is bound to it or launches rediscovery until a more fitting service is discovered and bound to, as described in §4.7.

During runtime – the middleware monitors the context at every recheck time (initially set to every 30 seconds, except at the “HighQuality” user preference when it is set to 10 s, adapting at any further rediscovery). At every recheck time Composite Policy execution takes place under the new current context, providing the new most appropriate values for the SR QoS variables and for the middleware configuration variables. If the Binding Module reaches decision for rediscovery, based on policy-based threshold parameters, then it launches rediscovery on the new QoS values.

4.5.2 Decision-making Mechanism for Rediscovery at Runtime

A decision-making evaluation mechanism on policy-based threshold parameters is hard-coded in the middleware Binding Module, deciding whether the discrepancy between the last adaptation (and currently in force) QoS variables values from the newly calculated ones is significant enough to necessitate adaptation and rediscovery or not. The evaluation mechanism takes into consideration policy-based threshold parameter values.

At negative evaluation, the Binding Module cancels adaptation of the SR and of the middleware, in order to avoid the nuisance for the user of frequent rediscoveries and consequent delays and changes of performance quality for no or very low advantage gained.

At positive adaptation decision, the Binding Module launches service re-discovery at the newly calculated SR QoS values and the middleware configuration variables are adapted.

The decision-making mechanism for the evaluation of the new service QoS vs. the last adaptation current one, is the same one as for the evaluation of a Service Offer QoS terms vs. the submitted SR QoS terms, and is explained analytically in §4.7.1.

4.6 Adaptation Walkthrough

What follows is a walkthrough of a typical cloud service discovery and adaptation runtime:

1. The client device runs an application which needs to request a service from the Cloud. This may be a Software as a Service (SaaS) or Infrastructure as a Service (IaaS) request.
2. The Volare middleware is active. It intercepts the request, appends relevant context and resource data, as well as the relevant policy for the request.
3. The Volare middleware adapts the service request according to the global and service level adaptation policies, as well as the current context/resource data of the system, as seen in Figure 4-2. It first requests the context data declared in the adaptation policy from the OS. Then the middleware performs the necessary QoS adaptation based on that data, according to the adaptation rules specified in the policy.

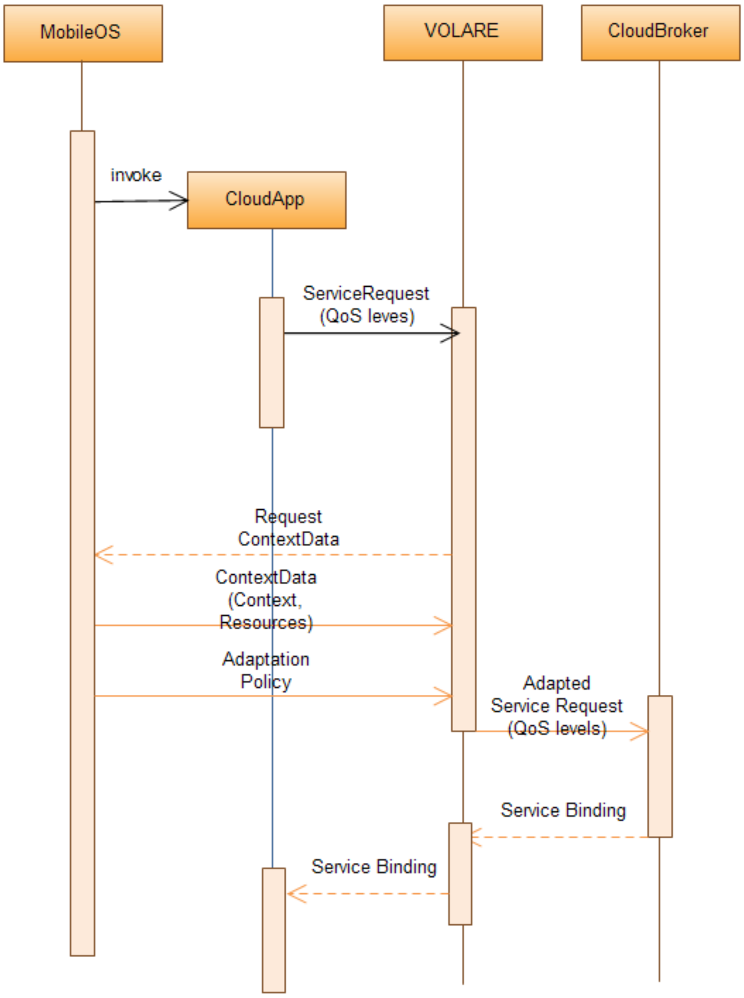


Figure 4-2 - Typical Service Binding

4. The adapted service request will then be sent to the Cloud Service Provider/Broker.
5. As seen in Figure 4-3, if the CSP/CSB fails to produce acceptable results, the Volare middleware will adapt the request further according to the policy files, lowering the QoS levels where appropriate. Then it will send the re-adapted service request to the CPS/CSB.

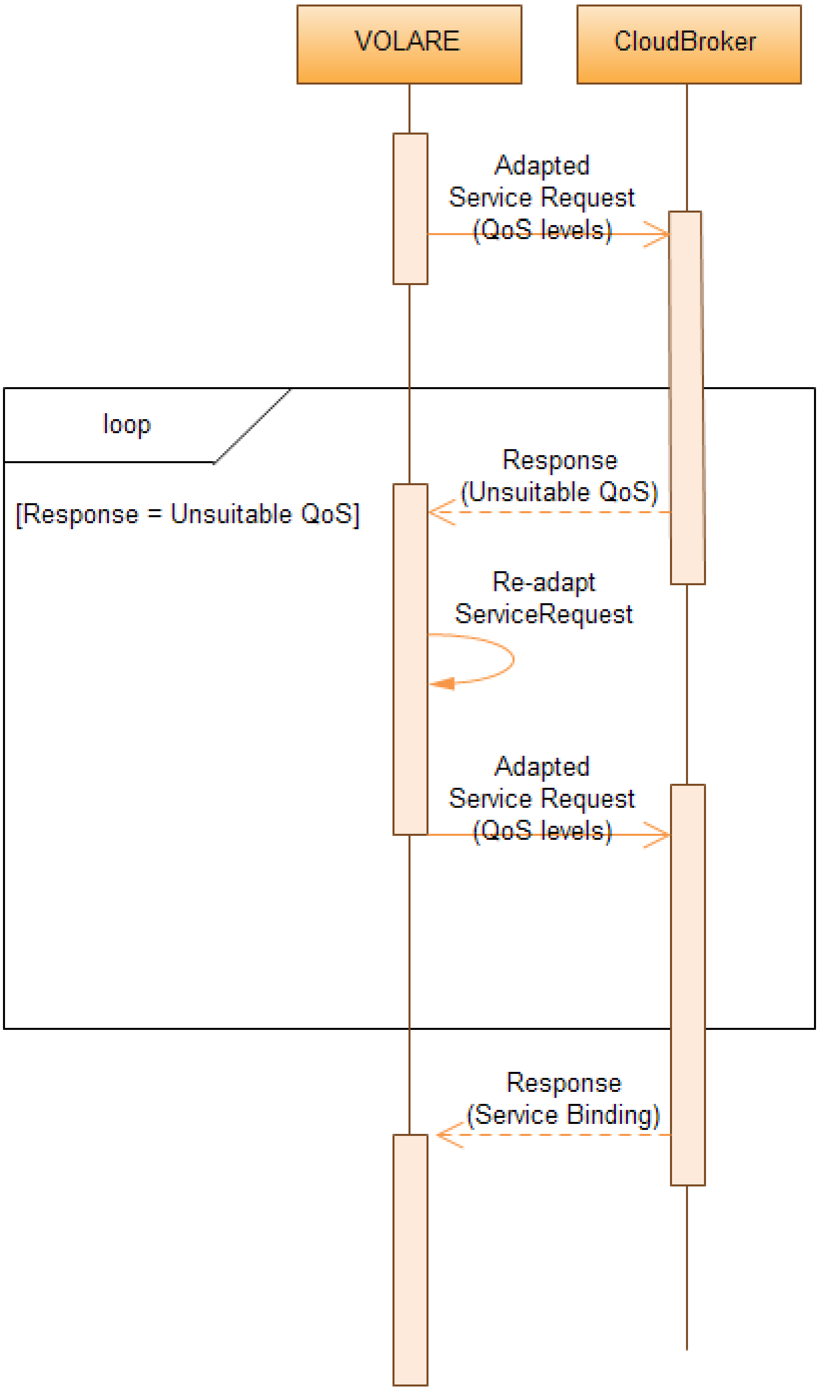

VOLARE
CloudBroker
Adapted Service Request (QoS levels)
loop
[Response = Unsuitable QoS]
Response (Unsuitable QoS)
Re-adapt ServiceRequest
Adapted Service Request (QoS levels)
Response (Service Binding)

Figure 4-3 - Handling QoS matching

6. When finally a fitting service is found Volare will forward it to the mobile OS.

The Volare monitoring service keeps running, and dynamically activates at set intervals or when significant deviations in context and resources happen.

At these intervals, Volare will check that service provided by the current binding satisfies the client, given the current context/resource data which may now be radically different compared to the ones at the time of the binding.

If the service offered by the provider is radically different than the one currently required by the client, the device will attempt to rediscover to a new service with the new requirements, going back to step 2, as seen in Figure 4-4. This rebinding may simply be a new agreement with the same provider for higher/lower QoS levels, which may prove more cost effective, or it may be binding with a more cost effective provider.

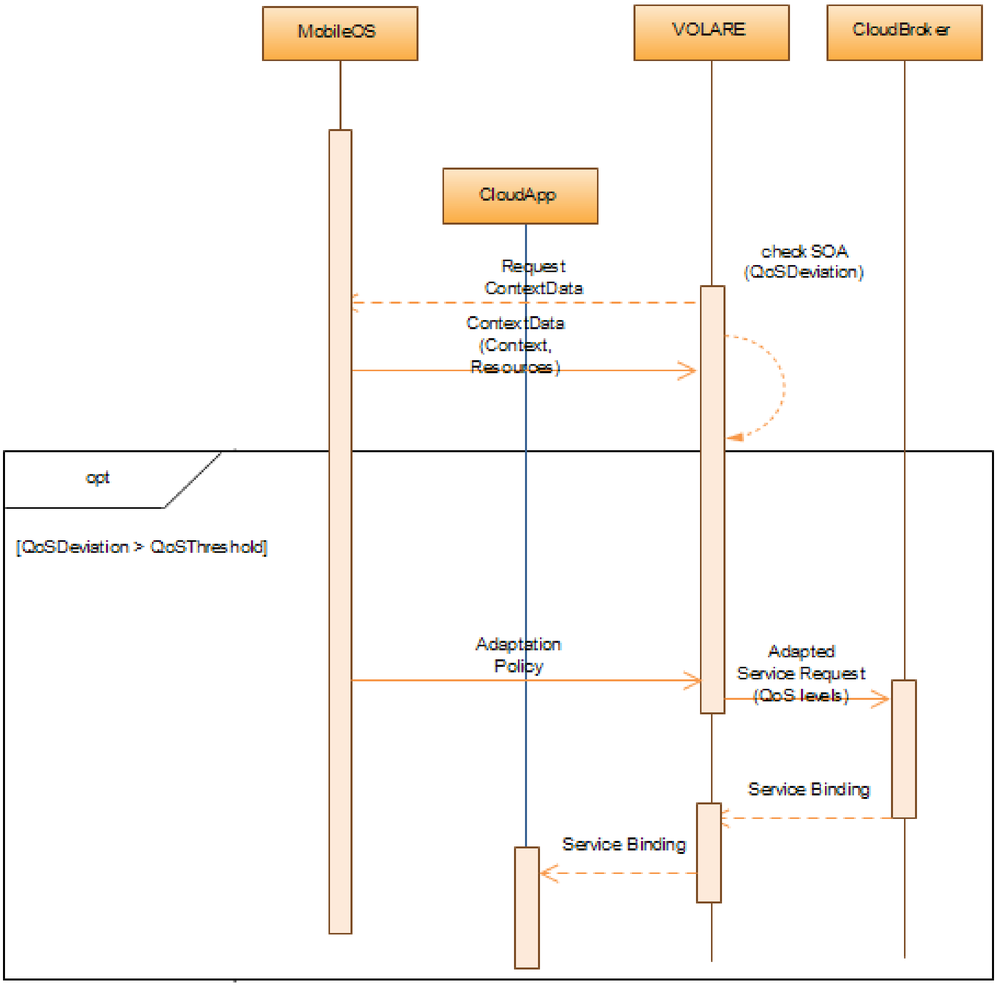


Figure 4-4 - Dynamic QoS level monitoring

In case of service interruption, rediscovery will automatically be activated. In this case, binding with different providers will be given priority.

The thresholds for what is considered a satisfactory service, and when the service should rebind will be defined in a global policy level file.

4.7 Service QoS Evaluation on Policy-based Parameters

The CSB or CSP, in response to a SR by a mobile application, discovers relevant services and after filtering based on the SR QoS Terms returns a *Service Offer* with the most appropriate QoS level values. However as there are multiple QoS terms, filtering and service selection may follow the CSB/CSP specific service selection mechanism.

The middleware Binding Module intercepts the *Service Offer* and evaluates it. Volare has its own policy-based service evaluation parameters – interpreting user preferences and policy goals – that are used by the Volare hard-coded Service Evaluation Algorithm.

At policy execution the following policy-based configuration-variables are provided as parameters for Service Offer Evaluation for acceptance and SR eventual adjustment for rediscovery:

(i) the QoS terms values calculated at policy execution,

(ii) the calculated value for *binding margin* bMi on each QoS term,

(iii) the calculated value for *renegotiationAdjustment* configuration-variable,

(iv) the sign of the binding margin of every QoS variable indicates whether the resolved-value corresponds to an optimal value constituting a lower limit – accepting only values equal or higher (positive binding margin), or corresponds to a higher bound - accepting only lower values (negative binding margin).

4.7.1 Service QoS Evaluation Mechanism

Given one candidate Service Offer by the CSP/CSB, it is evaluated by the Binding Module to be accepted or not. Suppose that the SR has n QoS terms. The Service Request and Service Offer values for the QoS term i, are respectively denoted as: *QoSReqi* and *QoSOffi*, 1 <= i <= n, while the corresponding absolute value of the *binding margin* is denoted as *bMi*.

It is required for each QoS term i value percentage difference not to exceed the binding margin *bMi* plus *rediscQoSThreshold*:

If bMi >= 0 then:

0 <= 100 x (QoSOffi – QoSReqi) / QoSReqi <= bMi + rediscQoSThreshold (1)

If bMi < 0 Then

100 x (QoSReqi – QoSOffi) / QoSReqi >= bMi - rediscQoSThreshold (2)

In order for an offered service to be accepted, the above constraints (1) to (2) should be satisfied for every QoS term value of the Service Offer. The same mechanism is applied when at runtime, the new SR QoS values are evaluated (as Service Offer QoS values) against the last adaptation in force QoS values.

4.7.2 Adjusting the Service Request for new Rediscovery

In the case that the Service Offer is not accepted, we need to adjust the SR QoS terms values – reducing the requested quality of service levels, in order to launch service rediscovery with adjusted QoS Terms. Again, for every QoS term of new SR: QoSReqNewi, we can have a simple formula like:

If bMi >= 0 Then (meaning a QoSReqi is a minimum value)

QoSReqNewi = QoSReqi x (1 + bMi + rediscQoSThreshold + discoveryAccuracy) (1)

If bMi < 0 Then (meaning a QoSReqi is a maximum value)

QoSReqNewi = QoSReqi x (1 + bMi - rediscQoSThreshold + discoveryAccuracy) (2)

4.8 Related Work

There has been extended work in the last decade on mobile dynamic context-aware adaptation for Web Services, some of which are referenced below. There are alternative criteria, through which the different research approaches on mobile policy-based CAA middleware/frameworks may be examined.

Concerning the middleware carrier with (a) stand-alone (client-side) systems, like HERA [31], HYDROGEN [9], CARISMA [6], Q-CAD [15], RAM [24], Volare [16, 17], (b) distributed systems, like DINO [9], CHISEL [7], ReMMoC [23], ODYSSEY [40], Rukzio et al [28] and (c) distributed with deployable components on the client device, like MobiPADS [20], MADAM [18], MUSIC [26], PLASTIC [25], QuAMobile [18, 19], QuA [22].

4.8.1 Stand-alone Mobile Middleware

We shall first consider the mobile client-based middleware/frameworks on CAA that are more comparable to Volare.

HERA [39] is an adaptive Hypermedia Presentation stand-alone mobile policy-based middleware with a specially designed AHA adaptation engine, using an updatable “static” User Preferences Model and the dynamic context from User browsing history, in addition to device and network context.

The Hydrogen [8] approach is a stand-alone mobile context-awareness framework of three - layered architecture. The “Adaptor” layer gets context information from sensors and delivers it to the “Management” layer, where the “ContextServer” stores and can be queried about the context and can share context information with other devices on peer-to-peer basis. Mobile applications are part of the “Application” layer and have access to context by querying the “ContextServer”.

Volare also adopts the client-side attitude, but with a different approach and instead of providing context-awareness services, it provides to independent mobile applications an adaptive functionality of service discovery on the Cloud as well as context-aware adaptation to policy-based compatible applications.

The *Context-Aware Reflective middleware for Mobile Applications* CARISMA [6] was prototyped as a peer-to-peer policy-based dynamic adaptation middleware, supporting the construction of context-aware, adaptive applications. It introduces Application Profiles as the information source for application properties, resource requirements, user preferences/associated QoS requirements and appropriate policies. At runtime, application developers/users may dynamically modify each application profile. At service invocation, the middleware consults the application profile, queries the status of relevant resources and determines the appropriate policy. At adaptation time, making use of utility functions that incorporate user preferences defined at application level, QoS parameters and resource requirements for each policy included in the application profiles, the highest utility variant from a set of alternative implementations of the service requested by the applications, is selected and implemented by the middleware, thus relieving the applications of this burden. Conflicts are resolved through an auction-like microeconomic procedure. CARISMA uses a C-A rule form in the policies without direct program code commands and so does Volare. However, Volare is built on the Service Oriented Architecture and not the peer-to-peer paradigm. Additionally, it provides itself the adaptive functionality for communication services on the Cloud to independent applications. It also differs from CARISMA in the approach adopted for the User Choices/Preferences context element as previously described, which in CARISMA is included in the application profiles as well as the use of utility functions while Volare follows the Weight-based Adaptation-Strategies Approach and the relevant Conflict Resolution mechanism, using weight values for adaptation reasoning through the adaptation-strategies concept.

QuAMobile [19][20] is a lighter version of the QuA [22] core reflective middleware aimed at context-aware mobile computing systems. QuAMobile implements open component architecture with distributed support for extra components deployment. It introduces a QoS-architecture and a distributed reflective meta-level representation of context and services for mobile computing. QoS requirements are specified in the form of utility functions, allowing users to specify their preferences at a high abstraction level. It uses a combined Resource and Context Model, with QoS management for the appropriate adaptation in selecting and implementing an application configuration.

Rukzio et al [28] present a peer-to-peer policy-based middleware for mobile commerce on architecture with a distributed system topology, using RDF for context representation, JESS [35] as policy language and inference engine and the JADE framework as agent-based middleware and agents under the FIPA standard. They also propose a policy editing framework with an interesting general purpose five steps iterative procedure, a UML-extended tool depicting the structure/execution flow of context information as well as a technique to appropriately group policies in modules successively executed. The paper describes a very powerful User Interface requesting user choices step by step at various levels of detail at every application session.

Volare also uses a multiple steps iterative procedure for policy development, specific to the weight-based adaptation reasoning technique on policy design and with specific methodology tools. Volare also considers the dynamic user preferences element as most important, but instead stands for low user intrusion and saves the “long term” User Choices Profile (UCP) for each application, implementing policy-based adaptation accordingly, while providing at runtime the choice among several User Preference behavioral patterns.

The *Q-CAD* project [15] is a context and QoS-aware *local and remote resource discovery and selection* reflective framework for pervasive environments, requiring a shared ontology for context, resource names and characteristics. An *application profile* specifies how the User wished the context should influence resource discovery. Resource discovery is implemented through static *resource descriptors*. At resource discovery, filtering through the application profile context constraints leads to discovered resources pruning. Then selection from the remaining ones is based on the values of a *utility function* for each discovered resource that should be maximized for the most suitable one.

Volare shares similar combination of features like policy files, user preferences and a User Choices Profile for each application, setting the User Choices, weight for utility functions and adapted service requests for resource descriptors. The Q-CAD methodology, although wider in scope than Volare, does not include dynamic adaptation and rebinding of services at runtime.

4.8.2 Distributed Policy-Based DCAA Frameworks for Mobile Systems

Several distributed frameworks, supporting policy-based DCAA on mobile apps, developed the last decade and we discuss some specific features, since they are not directly comparable to Volare:

CHISEL [7] is an open distributed framework for policy-driven context-aware dynamic adaptation in Iguana/J and the distributed reflective framework is built on the ALICE framework with clear separation of concerns. The custom-made policy language supports the event-condition-action rule model. The adaptation rules are of clear, declarative, human readable scripts, with introduction of new rules/events dynamically at runtime, thus covering unanticipated at design time new triggers and adaptation behaviors.

Volare shares with CHISEL features like: the self-adaptive character, the declarative simple form of adaptation rules – which are much simpler in Volare and of the C-A paradigm – and the dynamic support to unanticipated behaviors. However, Volare requires complete policy files from different issuing entities: the global policy file by the middleware developer and the application policy file by the application developer, with dynamic fine-tuning options by the user on the global policy. Volare allows dynamic user preference update instead of introducing adaptation rules at runtime for consistency and procedural reasons.

RAM [24] provides an infrastructure with an adaptation engine for development of adaptive applications by separating the application non-functional concerns in Services & Roles. Two policy levels exist used together at adaptation time: System policies for low-level system rules and application policies with higher level rules. A special purpose language is used for each type of policy. The paper concerns prototype implementation providing a simple proof of concept, with issues to be solved like conflict resolution and consistency guarantees on reconfiguration of components.

Volare also uses adaptation-rules of the C-A model and two level policies, one middleware-specific and one application-specific, which at adaptation time merge operating as one. However each policy file includes all the required high and/or low-level rules for ensuring its specific purposes. Volare also differs from RAM in that it makes use of weight values, as well as in the use and influence of the user preference dynamic context element on adaptation behavior.

DINO [9] presents a distributed infrastructure consisting of a number of brokers for QoS specification and service provider selection in open dynamic environments. It aims at supporting development of service engineering for discovery, selection, binding, delivery, QoS monitoring and dynamic service composition and architecture reconfiguration.

MobiPADS [20] - The Mobile Platform for actively Deployable Service combines context-awareness, dynamic service reconfiguration and mobile agents called *mobilets* in master-slave pairs. It has three tiers: client, proxy and server. It supports not only adaptive behavior but adaptive applications as well. It follows a very different approach than Volare, with deployable components on the mobile.

ODYSSEY [40] is a mobile client – server platform for mobile data access that provides adaptive services to independent concurrent mobile applications in a dynamic context-aware manner based mainly on resource availability and QoS. It is partly implemented on the OS kernel and partly as a middleware. Applications provide a window of “tolerance” on required resource levels, while a “Warden” for each the application provides suitable adaptation levels. Odyssey introduces the concept of “fidelity” to label alternative context versions. It is original but different from Volare in architecture and approach.

ReMMoC [23] - *Reflective Middleware for Mobile Computing* is a reflective middleware platform with two middleware services: service discovery and remote binding, using OpenCOM as its underlying component technology and implemented in separate Component Frameworks (CFs). During runtime, the service discovery framework adapts itself to the mobile network discovery protocol, and implements dynamic reconfiguration to the service composition within the binding CF to allow interoperation with heterogeneous services.

ReMMoC makes use of rule-based policies supporting fixed, predetermined, component compositions, like Volare. Volare differs in architecture, scope and approach, in the use of weight functions and of the User Preferences Model.

MADAM [18] and MUSIC [26] provide a framework for the development of mobile context-aware self-adaptive component-based applications that support dynamic composition through the use of a sophisticated middleware. Both projects follow a model-driven development methodology based on abstract adaptation models and model-to-code transformations. Dynamic adaptation is achieved by plugging into component type different component implementations with the same functional characteristics, or modifying the composition. The middleware on the mobile device uses utility functions based on QoS model to calculate utility scores for each variant under the current context. The most suitable variant is the one with the highest score and it is selected for implementation.

Volare also makes systematic use of the utility concept, but in the Volare approach as “weight”. It does not directly support dynamic service composition or service planning of adaptive applications, offering adaptive services involving only predetermined alternative service implementations and parameter adaptation in its scope of application.

PLASTIC [25] introduces the PLASTIC platform to enable robust distributed lightweight services in B3G networking environments (i.e., environments ggregating the various networking technologies available) through both design-time and run-time development support using a service-centric model. Volare focuses on different issues.

MIMOSA [36] and its core project CARE [69] present an interesting distributed framework that couples a middleware for context-awareness with an intermediary-based architecture for content adaptation on Web Service Provisioning. Although different in scope from Volare, they also raise the question of aggregation of Policies made by different entities (User, Service Provider etc) and set a framework for conflict resolution. MIMOSA also makes use of weight values assigned to each adaptation-rule, but simply used for the purpose of Intra and Inter-Policy Conflict Resolution. CARE and more closely MIMOSA also make use of an innovative multi-feature User Interface for User Preference declaration, for indicating inappropriate adaptation behaviour and suggesting new Policy rules.

The Weight-based adaptation reasoning approach is different. The Volare User Preferences Model allows parameter customization, in a manner similar to MIMOSA. However, the Volare Policy tries to prevent transparently to the User inappropriate adaptation behaviours, setting constraints that moderate User wishes for adaptation inappropriate under the current context, by reasoning on different adaptation-concerns and balancing competing adaptation interests.

4.8.3 Mobile Applications on the Cloud

Finally, there has recently been similar work on context-aware adaptation of mobile applications [4] with cloud service discovery, focusing on market analysis based adaptation of the request. It focuses on choosing the most efficient service by means of cost-analysis of available services in the market and does not take into consideration the context of the client device.

In [27] a different approach is presented for mobile systems, by replicating the whole smart-phone image and running the application code in powerful VM replicas on the cloud for augmented execution, outsourcing the heavy load computation on the *cloud clone*. No specific comments on DCAA are included in the paper.

Some new research is published on interactions of mobiles and cloud services.

In [65] a distributed framework for context-aware service provisioning, by dynamically adapting on the provider side the offered cloud service to the mobile user context, through dynamic service composition at the provider side in the form of service. This work is on dynamic service composition on cloud provisioned services, but with a different scope than Volare.

CasCap [66] is a distributed power management framework for mobiles, making use of *crowd-sourced context monitoring*, *functionality off-loading* and *adaptation as a service* on the cloud on a clone or a proxy, adapting energy-demanding activities in relation to WAN node access and other considerations.

In [67] a context-awareness cloud-based service for profiling mobiles based on frequency of accessing services on the internet, with activity database storage and management on the cloud.

In [68] an action-based modified approach on adaptation reasoning is presented by evaluating the service state-model and specifying explicitly the adaptation behaviour in response to context change. It also distinguishes between independent and dependent adaptation actions, thus reducing an eventual state explosion problem. It introduces a verification procedure on the correctness of the adaptation model with an enabling condition element, while also transforming it to a Petri Net model for design-time verification. The weight-based methodology has also developed static and dynamic verification tools, to prevent failures and identify rule faults. Additionally, at policy execution the adaptation variables values vector (i.e. the new adapted state) is verified for completeness and consistency before implementation, according to hard-coded and policy-based criteria. Volare follows the situation-action paradigm and the state explosion problem is not critical, since no state-model is evaluated at adaptation time. Additionally, Volare safeguards the consistency of matched and selectable adaptation rules by analysing at design time the adaptation into mutually exclusive configuration variants, thus overriding rules belonging to different configurations than the one selected, that would otherwise raise the possibility for inconsistent adaptation actions.

5 The Volare Adaptation Policy Specification Language

5.1 General Principles

This chapter presents the Volare declarative, first order logic, Adaptation Policy Specification Language (APSL) for rule-based policy authoring concerning middleware-based adaptation of the SR on cloud service discovery by mobile applications.

As mentioned in Introduction (ch. 1) and in Project Overview (ch. 3), we set requirements for the APSL in response to challenges ascertained in the preliminary parts of the work. In an effort to deal with those challenges, the Volare Adaptation Policy Specification Language (APSL) was developed which follows the general principles described below:

1. Two-level Adaptation Policy Files

The adaptation adaptation-rules are expressed in **policies** contained in **policy files,** one policy file for the middleware adaptive operation and one for each application service request, which can be easily read, printed, edited and can be updated at runtime.

Since the middleware provides policy-based DCAA support to independent applications, a two level policy approach has been adopted. The **global-level policy** named **global policy** is authorized to globally affect all relevant adaptation for global-variables (i.e. common QoS-variables) and the Volare configuration-variables. The **application policy** for each application is authorized to affect adaptation on application-specific QoS variables expressed through the Service Request as well as on the QoS-Variables pre-defined by the global policy.

2. Adaptation-Rules Form

The adaptation rules in Volare follow the Condition-action paradigm: If "conditions" – Then "action". The "action" part is a value assigning statement to an adaptation-variable, called in this work: *adaptation-statement*.

Consequently, every adaptation policy rule in Volare is distinguished in two parts: the *Criteria* part (i.e. the "If condition statements" part) and the *adaptation-statement* (i.e. the "action" part) of the adaptation-rule, assigning a value (through a value, expression or function variable at the RHS) to the **adaptation-variable** referenced at the LHS of the adaptation-statement, describing the way the variable should be adapted under the current context.

Specific keywords, set at the RHS of an adaptation-statement, assign rule priority to each adaptation-rule, thus imposing the rule priority to depend also on the policy of origin in order to maintain the balance.

3. Policy File Structure

Each policy file consists of three required sections: (a) the **Variables Declarations section**, where the variables are declared, (b) the **Criteria section** which consists of several criteria groups of condition statements, each criteria group corresponding to the specific **Subpolicy** under the same name and **enabling** it when all its **criteria** conditions are satisfied by the current context data, (c) the **Subpolicies section** consisting of a set of "sub-policies" called hereinafter **Subpolicies**, each composed of a number of **adaptation-**statements with common Criteria conditions.

4. Composite Policy at runtime

At policy execution time, the **global** and **application policy** file of the active service request are parsed and merged creating the **Composite-Policy** that drives the adaptation, composed of the three Composite Sections: the Variables *Declaration, the Criteria* and the *Subpolicies* sections, of the global and the application policy corresponding section statements.

5. Composite Policy Execution

At the **Composite-Policy** execution, the **Policy Engine** Cycle operates on the three typical steps:

a) At the **Match Rules step** starts sequential evaluation of each Criteria group with the current context data, to identify the **Matched** Criteria **Subpolicies** that exclusively satisfy the current context. Multiple corresponding Subpolicies from both the global and the application policy for the active service request may be **matched,** resulting in a dynamic list of the adaptation-statements of each **matched Subpolicy**.

b) At the **Select Rules step**, the **Policy Engine** selects all the highest priority adaptation-statements for each adaptation-variable, forming the **Agenda.**

c) At the **Execute Rules step**, every selected adaptation-statement of the **Agenda** that is calculated provides an **execution-value** for the adaptation-variable it is authorized to assign value to.

6. **Multi-cycle Policy Execution Session**

Multi-cycle policy execution is supported by setting the maximum number of Cycles and by assigning to every Criteria conditions group a cycle declaration number specifying at which policy execution cycle it is to be evaluated for matching. This optional feature allows successive hierarchic evaluation of the adaptation parameters values.

7. **Multiple Rules eventually selectable for each Adaptation-Variable at Policy Execution**

The Volare APSL allows eventually multiple adaptation-rules on the same adaptation-variable and expressing different adaptation viewpoints to be matched and selected at policy execution.

8. **Conflict Resolution Directives**

For resolving issues of **conflicting execution-values** provided by more than one selected adaptation-statements corresponding to the same adaptation-variable, the APSL **Conflict Resolution Directives** are implemented by the Policy Engine, as explained in detail in the relevant section below. For this purpose a weight function value is formally required, assigned by the developer to every adaptation-rule, appended at the end of the LHS of each adaptation-statement, indicating the relative importance of the **execution-value** in comparison to the execution-values of the same adaptation-variable.

The basic Conflict Resolution Directives adopted are:

a) When there is only one **execution-value** for an adaptation-variable, then this value is assigned to the variable as the **resolved-value**.

b) When there is more than one **execution-value** for a **numeric variable** from multiple adaptation-statements, the Conflict Resolution ***Participative Weighted Average Directive*** is applied on the **execution-values** of the selected and calculated adaptation-statements to derive the **resolved-value.**

c) When there are more than one **execution-value** concerning **non-numeric** adaptation-variables (such as Boolean or string type variables), then the Conflict Resolution ***Majority Directive on the sum of weights*** is applied, selecting as the **resolved-value** the execution-value with the higher weights sum.

5.2 General Policy Syntax Guidelines

A typical complete Volare policy file is separated into three required sections:

1. Variables Declaration Section

The *Variables Declarations section* contains the declarations of new context and adaptation-variables (global QoS and configuration-variables) of the global policy as well as the QoS adaptation-variables application policy.

2. Criteria Section

The *Criteria section* consists of Criteria conditions groups, one for each Subpolicy, sharing the same name with the respective **Subpolicy** when all the **Criteria** conditions are satisfied by the current context data.

3. Subpolicies Section

The *Subpolicies section* consists of a set of “sub-policies” called hereinafter **Subpolicies**, each composed of a number of **adaptation-statements** having the same Criteria (i.e. “If conditions”).

Figure 5-1 – Typical Policy File Structure

```
Policy Policy-Name{
    Declarations{
        …
                 };
    Criteria{
        [1] Criteria Subpolicy-Name1{
            …
                                   };
        …
            };
    Subpolicies{
        Subpolicy Subpolicy-Name1{
            …
                                     };
        …
                };
};
```

For policy file integrity and consistency and security reasons, each Policy can only be declared once, in one file, and all three relevant sections, i.e. the Declarations, the Subpolicies and the Criteria section must be included in that policy file. This allows global and application-level policy developers to include read-only policy files, in order to avoid tampering by other developers.

A new application-level policy file needs to have a Declarations section that at least includes the declaration of the new service request for the application, as well as the application-specific QoS Variables that may be selected from the global QoS Variables, even if all the other context and adaptation-variables are the generic ones of the global-level policy.

5.2.1 Policy File Syntax

A policy file is a sequence of line statements with characters and symbols conforming to the APSL rules. Starting a policy file or the Declarations, Criteria or Subpolicies section or a Subpolicy or Criteria conditions group of line statements is denoted by the "**{**" just after the last character. Ending any one of the above groups of line statements is denoted by the "**};**" symbols. The end of each line statement in a policy file is denoted by a "**;**" termination symbol. or alternatively with an **"or"** or **"xor"** operand. The only allowed exception is at the end of a line statement denoting the beginning of a new group of line statements, which starts simply by the "**{**" symbol, as shown in the previous figure on the policy file Structure.

The expressions denoting start of a group of line statements are: **Policy Policy-Name{**, **Declarations{**, **Criteria{, Subpolicies{**, **Subpolicy Subpolicy-Name{**, **Criteria Subpolicy-Name{** and **case{**. All these groups of statements should end with a line having the single character: "**}**" followed by the "**;**" statement-termination character.

Similarly, the operand "**or**" or "**xor**" does not take a termination symbol, and constitutes the only word of a line statement.

5.2.2 Keywords and Operands

The following keywords and operands are reserved and used by the Volare Adaptation Policy Specification Language (APSL), starting with a small capital letter:

1. Criteria Operands

and implied operand

When defining the criteria object for a Subpolicy, each line conditional statement in the Subpolicy criteria object is assumed to operate on an **and** operand, unless specified otherwise by the following **case, or** or **xor** operands.

or or **xor** operand

If an **or** or **xor** operand infixed in a **criteria** object is used, then the contents are assumed to share an **and** operand. It is used as a single statement term without termination symbol.

case operand

The "**case"** operand infixed in a **criteria** object, allows to specify **or** or **xor** relationships within a subset of a **Criteria** conditions group that start and finish within brackets {}.

default keyword

The "**default"** keyword in an empty of other conditions Criteria conditions group, signifies that the Criteria should be matched at each policy execution of the assigned Declaration Cycle and the corresponding Subpolicy is also matched at Policy execution.

empty Criteria

An empty "Criteria" is a Criteria conditions group empty of conditions in the form:

[n] Criteria Criteria-NameX{

default;

}; with n E N, where n = 1, 2 3 etc.

Since an *empty Criteria* has no Criteria conditions, it is considered matched at the Policy Engine Cycle = n, activating the same name Subpolicy "Criteria-NameX". An empty "Criteria' is typically used at a **cycleNo** to assign default values to adaptation-variables through the adaptation-statements of the corresponding Subpolicy.

not() operand

The **not( )** operand applied on a Boolean adaptation-variable in a RHS expression of a Criteria condition or of an adaptation-statement, expresses the negation of the Boolean value of the expression.

2. Priority Assigning & Conflict Resolution Keywords

overridesAll keyword

The **overridesAll** keyword can be prefixed at the LHS of a specific adaptation-statement within a Subpolicy, letting it override any other global or application-level matched adaptation-statements for the same variable without the “overridesAll” keyword. The **overridesAll** keyword operates equally on the corresponding binding threshold margin attribute within brackets (see below).

Note 1: In the case of adaptation-statements on the same adaptation-variable which are set to **overridesAll** both in the global and the application-level selected Subpolicies, the global-level policy adaptation-statements takes precedence.

Note 2: If more than one adaptation-statement on the same adaptation-variable is selected with “overridesAll” keyword from the global and the application policy, then at policy execution the Conflict Resolution Directives override all other statements, either from the App Policy or without overridesAll and participative weighted contribution procedure is implemented on the selected A/Sts.

yieldsAll keyword

The **yieldsAll** keyword has the exact opposite functionality from the **overridesAll** keyword, making the specified adaptation-statement(s) only be calculated when there is no other adaptation rule referring to this adaptation-variable without **yieldsAll**, otherwise it is ignored. When used in the global layer, **yieldsAll** will operate in a similar manner. However, if for an adaptation variable several adaptation statements are set to **yieldsAll** in both the global and the application-level, then the application-level adaptation-statements will be executed, not the global-level one(s).

overrides keyword

The **overrides** keyword can be prefixed at the LHS of a specific adaptation-statement within a Subpolicy, letting it override any other global or application-level matched adaptation-statements for the same variable in the Agenda that does not have an **overridesAll** keyword. The **overrides** keyword operates equally on the corresponding binding threshold margin attribute within brackets (see below). If at policy execution, there are adaptation-statements with **overrides** on the same adaptation-variable from both the global and the application policy, then the Conflict Resolution Directives are applied on all A/Sts with **overrides**, based on their weight values. If there is more than one “**overrides”** in adaptation statements on the same adaptation-variable, Conflict Resolution Directives apply as normal.

yields keyword

The **yields** keyword has the exact opposite functionality from the **overrides** keyword, making the specified adaptation-statement(s) only be calculated when there is no other adaptation rule referring to this adaptation-variable without

yields, otherwise it is ignored. The **yields** keyword operates equally on the corresponding binding threshold margin attribute within brackets (see below). If at policy execution, there are adaptation-statements with **yields** on the same adaptation-variable from both the global and the application policy then the Conflict Resolution Directives are applying the participative weighted contribution approach on all A/Sts with **yields**, based on their weight values.

overridesAsUpperLimit or **overridesAsLowerLimit** keyword

One of these keywords prefixed at the LHS of an adaptation-statement for numeric adaptation-variables when encountered at policy execution time, defines an upper or lower limit value for the referenced attribute(s) of this adaptation-variable (also for the binding margin - if a value exists) and not a execution-value. All execution-values on attributes of this adaptation-variable that exceed this limit are modified to conform by the Policy Engine Manager to the limit value and only then the Conflict Resolution Procedure is applied to calculate the resolved-value.

Rule 1: If more than one adaptation-statement with the keyword "**overridesAsUpper (or Lower)Limit**" are encountered for the same numeric adaptation-variable at policy execution time, then the Conflict Resolution procedure adopts the "stronger" (the minimum for Upper and maximum for Lower limit respectively) value as the corresponding Upper or Lower Limit.

Rule 2: When An Upper or Lower Limit value is available for a numeric adaptation-variable at policy execution, then the Conflict Resolution Procedure ensures that each execution-value is first adapted to conform to the corresponding Limit and only then the Weighted Average Rule is applied to the execution-values.

Rule on mutually exclusive keywords: Only one keyword is allowed, prefixed, in an adaptation-statement among the following sixmutually exclusive ones: *yieldsAll*, *yields, overrides, overridesAll, overridesAsUpperLimit, overridesAsLowerLimit.*

cyclesMax keyword

The **cyclesMax** keyword indicates the maximum number of Policy Engine Cycles at a Composite policy execution, and is defined at the global policy file Declarations section by the calculation-variable: **cyclesMax**, in a Declaration statement of the form: **integer CalculationVariables.cyclesMax == 1 (or 2, 3 etc).** Any Criteria group with a cycle declaration higher than the **cyclesMax** will not be executed.

2. Attribute Assigning Operands

binding threshold margin [ ] operand

A float type non negative number implying percentage enclosed in **[ ]** square brackets postfixed at the RHS of an adaptation-rule concerning a numeric QoS-variable is a modifier enabling specification of the **binding threshold margin** for the numeric adaptation-variable concerned. The following rules additionally apply:

Rule 1: A non-numeric QoS-variable or a non QoS-variable does not have a **binding margin** value.

Rule 2: The default threshold margin for numeric QoS-variables not assigned a margin value at the adaptation-rule level is specified in the global-level policy as **defaultBindingMargin**.

Rule 3: The same Conflict Resolution Mechanism that is applied on the **execution-values** for a numeric QoS-variable is also applied on the corresponding *recommended-value(s)* of the **binding threshold margin** attribute of this QoS-variable.

Note 1: If the variable has a **binding threshold margin** of 10, then the service can accept bindings with a discrepancy of 10% from the adapted request value. The higher the **binding threshold margin**, the more leeway the specified QoS variable has to be readapted if a satisfying service is not found, and the more likely it is to be compromised in favour of maintaining another adaptation-variable value with lower threshold margin. This way the developer can specify which QoS metrics can be compromised first.

Note 2: An adaptation-variable not permitted to be negotiated at binding time may be specified with a zero **binding threshold margin [0]** in the adaptation-rule.

weight ( ) operand

Numeric values or expressions or references enclosed in **( )** brackets postfixed at the RHS of an adaptation-rule are called in this work **weights** (also called weight functions) that enable specification of weighted modifiers to be used during Conflict Resolution among execution-values on the same head predicate provided by the selected and executed adaptation-statements originating from both the global or application-level policies.

Rule 1: The minimum allowed **weight** value is **0** and the maximum **weight** value is **1.0** (see Conflict Resolution Directives).

Rule 2: If the **weight** term in an adaptation-rule is omitted, then the default weight value **0.5** is implied.

Note: Adaptation-rules with higher **weights** are more highly considered when there are multiple adaptation-rules regarding the same adaptation-variable and under the same context, across different subpolicies. This enables developers to specify adaptation-rules more or less dominant compared to others.

cycle declaration [ ] operand

The ***cycle declaration*** operand **[ ]** with a positive integer digit is prefixed on each Criteria conditions group starting statement, indicating at policy execution at which Policy Engine Cycle No the Criteria conditions group referenced may be evaluated for matching. For developer-friendly reasons, if the policy file has only Policy Engine cycleNo = 1, then the prefixed "[1]" notation may be omitted.

The notation adopted for the starting statement of any Criteria Conditions group is: **[n] Criteria Criteria-Name{, where n =1 or 2 or 3…**

Note 1: The application policy developer has to comply with the **cyclesMax** value of the global policy, since if it is higher, then the corresponding criteria group will not be executed.

5.3 Volare Adaptation Policy Specification Language Syntax

What follows is a formal definition of the syntax of the Volare policy language using the generic **Extended Backus–Naur Form (EBNF)**, with most important features of EBNF used in this paper being:

Extended BNF	Meaning
Unquoted words	Non-terminal symbol
" ... "	Terminal symbol
' ... '	Terminal symbol
(...)	Brackets
[...]	Optional symbols
{ ... }	Symbols repeated zeroor more times
{ ... }-	Symbols repeated one or more times
=	Defining symbol
;	Rule terminator
|	Alternative
,	Concatenation

/* ...	Row Comment

/* Updated more detailed version using the Thesis terms

PolicyFile ::= PolicyFileTitle, Declarations, Criteria, Subpolicies, “}”;

PolicyFileTitle ::= “Policy “, “Global” | ServiceID, “{“;

ServiceID ::= identifier;

Declarations ::= “Declarations{“, DeclStatements, “}”;

DeclStatements ::= { DeclStatement }-;

/* Short Definition

DeclStatement ::= DataType, Repository, identifier , “ == “, Component, { “.”, identifier }- | expression | number;

/* Alternative more detailed Declaration statements

DeclStatement ::= ContextStatement | QoSStatement | ConfigStatement | CalcStatement | AuxStatement | VariableAssigningStatement;

ContextStatement ::= DataType, ContextVar, identifier, “ == “, “ContextMonitoringM.”, identifier | “UI.” , identifier | ”StatAnaM.”, identifier, “.”, Period, “.” StatisticTerm;

QoSStatement ::= DataType, QoSVar, [“activeRequest.”,] QoSVariable, “ == “, “BindingM.”, identifier;

ConfigStatement ::= DataType, ConfigVar, ConfigVariable, “ == “, Component, “.”, identifier;

CalcStatement ::= DataType, CalcVar, identifier, “ == “, number | expression | identifier;

AuxStatement ::= DataType, AuxVar, identifier, “ == “, “AdaptationM.”, identifier;

VariableAssigningStatement ::= “activeRequest.”, identifier, “ :: “, Repository, “.”, identifier;

Period ::= RecheckCycle | Session | Daily | DischargeCycle | Monthly | OverallHistory;

StatisticTerm ::= Sum | Avg | Std | Max | Min | UpperConfLim;

ConfigVariable ::= “periodicity” | “preferredVariant” | “reNegotiate” | “rebindingDelay” | “rediscQoSThreshold” | “rediscContextThreshold” | “renegotiationAdjustment” | “renegotiationAttempts” | “recheckRate” | “defaultBindingMargin” | “rebindingRecheck”;

DataType ::= { integer | float | percentage | string | Boolean | date | time | functionality }-;

Repository ::= ContextVar | CalcVar | ConfigVar | QoSVar | AuxVar | ServiceRequests;

Component ::= ContextMonitoringM | BindingM | AdaptationM | SRequestM | StatAnalM | CAHiM | MUPI;

Criteria ::= “Criteria{“, CriteriaConditionsGroups, “}”;

CriteriaConditionsGroups ::= { CriteriaConditionsGroup }-;

CriteriaConditionsGroup ::= “[“, CycleDeclaration, “]”, “Criteria “, identifier | “Default”, “{“, Conditions, “}”;

CycleDeclaration ::= integer;

Conditions ::= { Condition }- | { Condition }-, “ or ”, { Condition }- | { Condition }-, “ case{“, { Condition }-, { “ or ”, { Condition }- }-, “}” | “void”;

Condition ::= identifier, Restriction;

Restriction ::= logicSymbol, “ “, identifier | number | expression;

Subpolicies ::= “Subpolicies{ “, { Subpolicy }-, “}”;

Subpolicy ::= “Subpolicy “, identifier | “Default”, {“, { Actions }-, “}”;

Actions ::= { NonQoSAction }- | { QoSAction }-;

QoSAction ::= [Priority,] QoSVariable, “ = “, number | identifier | expression, [“ [“, BindingMargin, “]”,] “ (“, Weight, “)”;

NonQoSAction ::= [Priority,] ConfigVariable | AuxVariable, “ = “, number | identifier | expression, (“, Weight, “)”;

Priority ::= [“overridesAll ” | “overrides “ | “yields “ | “yieldsAll “ | “overridesAsUpperLimit ” | “overridesAsLowerLimit “];

BindingMargin ::= number | expression;

Weight ::= number | expression;

AuxVariable ::= identifier;

logicSymbol ::= ‘ > ‘ | ‘ < ’ | ‘ >= ’ | ‘ <= ’ | ‘ = ’ | “not”;

expression ::= { number | identifier, arithmeticSymbol, | { number | identifier, powerSymbol }-, arithmeticSymbol, expression | expression, [, number,], (,number,) }-;

number ::= integer | integer, “.” , integer;

integer::= num, {num};

arithmeticSymbol ::= ‘-‘ | ‘+’ | ‘*’ | ‘/’;

absSymbol ::= “abs(”;

denialSymbol ::= “not “;

powerSymbol ::= “^”

num ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

identifier ::= char , { char | integer };

char ::= 'E' | 'T' | 'A' | 'O' | 'N' | 'I' | ‘T’ | 'S' | 'R' | 'H' | 'L' | 'D' | 'C' | 'U' | 'P' | 'F' | 'M' | 'N' | 'W ' | 'Y' | 'B' | 'G' | 'V' | 'K' | 'Q' | 'X' | 'J' | 'Z’ | ‘.’ | “_” | “&”;

5.4 The Context & Adaptation Variables Profile

There are six categories of variables defined by the Volare Adaptation Policy Specification Language and six variables Repositories created and used by the Volare middleware:

1. The context-variables

The context-variables (of type float, integer, Boolean or string), the values of which cannot be changed by adaptation and may be distinguished to the “*physical*” context-variables, provided directly by sensors or the computing environment variables (including Boolean context-variables signifying an event or a state of a system component).

The ContextVar Repository includes all the declared context-variables of the global and application policies, either “**physical**” **context-variables** from various context sensors, or **statistic-calculation-variables** by the Statistical Analysis Module.

The “**composite**” **context-variables** equal to an expression including the value of another “physical” context-variable. Abstract-level conceptual or formula-based variables, that may be assigned a value like constants or modifier values used in Criteria statements or adaptation statements may also be declared and then used in the policy files.

A context-variable does not take values from the adaptation-statements of the **Subpolicies** section, but by the corresponding sensor referenced at the RHS of its Declarations statement. The values of the variables referenced from this Repository cannot be altered by a policy. Most of these variables are predefined in the global policy file, but the application-level policy developer may declare new context-variables representing sensor values, or execution context or statistic-calculation-variables when required.

The context-variables are of type numeric (float, integer or percentage) or non-numeric (Boolean, string, time or date) variables.

A context-variable does not take values from the adaptation-statements of the **Subpolicies** section, but by the corresponding sensor referenced at the RHS of its Declarations statement. The values of the variables referenced from this

Repository cannot be altered by a policy. Most of these variables are predefined in the global policy file, but the application-level policy developer may declare new context-variables representing sensor values, or execution context or statistic-calculation-variables when required.

The context-variables are of type numeric (float, integer or percentage) or non-numeric (Boolean, string, time or date) variables.

Note 1: Typically, the values of a numeric context-variable, if it has fixed, known, upper and lower bounds, are expressed as percentages of their maximum value.

2. The calculation-variables

The "**CalcVar" Repository** includes the **calculation-variables** that are declared and are assigned value directly at the Declarations Section statements of the global or application policy, representing constants or expressions using constants or context-variables values or already known history values of adaptation-variables (like current or last adaptation) values.

The calculation-variables are variables (of type integer, float, percentage, Boolean, or string) that are constants, or composite variables from expressions of known value at policy execution. They may include at the RHS expression, context-variables or adaptation-variables of known value (i.e. of previous adaptation) or other calculation-variables. They are used in the Criteria conditions as comparison threshold values or modifiers (multipliers in expressions) or in the adaptation-rules of a Policy as reference values or modifiers at the RHS.

The ContextVar Repository includes all the declared context-variables of the global and application policies, either "**physical**" **context-variables** from various context sensors, or **statistic-calculation-variables** by the Statistical Analysis Module.

The "**composite**" **context-variables** equal to an expression including the value of another "physical" context-variable. Abstract-level conceptual or formula-based variables, that may be assigned a value like constants or modifier values used in Criteria statements or adaptation statements may also be declared and then used in the policy files.

3. QoS-Variables

Each **QoS variable** is a set of two attribute references, the named **QoS variable** and the corresponding **binding margin,** to a general QoS variable name. Any global or application policy adaptation-statement adapting a **QoS variable**, adapts individually both attributes through the same adaptation-statement.

Declarations of **global variables** are allowed only in the global policy. However, the application-level developer may use the **"::"** operator to assign an a**pplication-specific QoS variable** to a **global variable** reference, through a statement in the Declarations section of the application policy file. In this case, the variable at the LHS of the Variable declaration statement substitutes the variable at the RHS of the Variable declaration statement, and all references of that variable are replaced. In this case, the application-specific QoS variable should be of the same units and domain with the global variable. Otherwise, in the **Declarations statement**, the RHS should additionally include the required numeric modifier factor (example in the Declarations section).

Each **global variable** is a set of references to either specific **serviceRequest-Name** variables or general variable names, which will automatically be associated to specific **serviceRequest** "QoS variables" of that name.

The **QoSVar repository** includes all the QoS adaptation-variables that are or may be involved in a Service Request. At the global-level Policy pre-defined QoS adaptation-variables are declared. They are called in this work "**global QoS adaptation-variables**" and are common QoS variables whose values can be altered by global-level or application-level policy statements. Global variables are preset to automatically reference certain common **QoS variable** names, in order to enforce certain variables to be global.

4. The Volare Configuration-Variables

The **ConfigVar repository** includes the reserved names adaptation-variables that directly affect how the Volare middleware operates, called **Volare configuration-variables**. These adaptation-variables are preset and can only take values at policy execution through the global policy adaptation-rules. The list and information on these variables is available in a subsequent paragraph.

The adaptation-variables (of type float, integer, Boolean or string), represent the operation parameters of the middleware or the SR QoS variables. Through their values, assigned at adaptation time, they define the SR and middleware adaptation behavior to a context change.

The adaptation-variables are distinguished by their purpose in:

"**QoS adaptation-variables**", that represent the adapted QoS values for the adapted service request, and in the Middleware reserved

"Configuration adaptation-variables", that provide self-adaptive character to the middleware.

5. The "auxiliary" Variables

The auxiliary-variables are policy -declared variables that are introduced by the global or the application policy for decision-making purposes and take values at policy execution. They may represent Mutually Exclusive Configuration Variants, or virtual parameters that may be assigned values at a Consecutive Execution Cycle (CEC), serving for further decision-making on dependent variables at the next CECs.

The **AuxiliaryVar Repository** includes the **auxiliary** adaptation-variables that are used for purposes of facilitating decision-making at policy execution, like weight-based MESC selection, and are declared and used in adaptation-rules of any Policy and are resolved across both Policies at policy execution.

6. The Service Request Variables

The variables, context, QoS variables or calculation-variables declared and used in an application policy concerning the Service Request.

Most of these variables are pre-determined by the global policy Declarations section.

The ServiceRequests repository includes all the specific application-level policy QoS variables for each service request that Volare may intercept and are represented as adaptation-variables during runtime. Values of the application-specific QoS adaptation variables referenced by **ServiceRequests** may only be altered by adaptation-statements of its specific application's Subpolicies. If however an application-specific QoS variable is assigned to a global variable through a Declarations statement in the application policy file, then it may also be adapted by adaptation-statements on this global variable in the global policy. This repository will automatically include all QoS variables included in a service request. The **ServiceRequests.activeRequest** variable group represents the service request intercepted by the Service Request Manager of the Service Request Module.

At the application policy Variables Declaration section, every ServiceRequests variable should be declared as QoS variable.

5.5 Policy File Structure

The Volare Adaptation Policy Specification Language (APSL) is a two level policy language, the global-level policy for the middleware operation, also called global policy and the application-level policy also called application policy, one for each service request. The global and the application policies for every relevant application, for consistency each is embodied in a policy file edited in plain text form, taking a suitable name.

Each policy file is composed of three required sections – even if one of them is empty (for instance the Declarations section of an application policy): the (**Variables) Declarations** section, the **Criteria** section and the **Subpolicies** section. Each of these sections will be described in subsequent paragraphs.

5.5.1 Variables Declarations

1. Variables Naming Convention

The names of variables or policy files and Subpolicies start from a letter and may only have alphanumeric characters as well as the symbols "-" or "_", without empty spaces or other non-printing characters.

The names of the policy files, Subpolicies or Repositories start with a capital letter.

The names of the context, calculation or adaptation-variables start with a low letter.

Reserved names are the names of the APSL keywords and operands as well as the names of the Volare configuration-variables and start with a small letter.

Each Policy, Subpolicy, Repository or variable name should be unique along both the global and the application policy file. This is especially important to the application policy developer, who should design the application-level policy file with the global policy file in mind. Of course the application may make use of the context, calculation and common-adaptation-variables and of the global QoS variables declared in the global policy. Additionally, the naming conventions of JAVA are followed for the variables names.

2. Variables Declaration Format

Variable declarations may be commonly used in both global and application-level policies, but may be declared in a policy file only once for each variable. In the "Variables Declaration" section of the global-level policy all the context and adaptation-variables (including the global QoS and the Volare configuration-variables) are declared. Note that some of the QoS variables are pre-declared as members of global variable sets by the global-level policy developer. In a similar manner, the context-variables will generally also be declared by the global-level policy developer, and Volare configuration-variables are preset.

The application-level policy developers will generally only have to declare the new application-specific QoS variables in the serviceRequests repository and occasionally context-variables representing new context sensors or statistic-calculation-variables in the ContextVar Repository.

In the Variable Declarations section several generic variable declaration notations are supported, according to each specific case, as explained below.

3. Variable Declaration Notation

The general variable declaration notation is:

typeId Repository-Name.variable-Name == itemReference; **(1)**

where:

typeId may be: float, integer, percentage, Boolean, string, time or date

Repository-Name is the name of one of the six repositories in § 5.2.4

variable-Name is the name of the new variable, starting with a small letter

itemReference may be:

(a) For context-variable declaration, a reference to the middleware module and the specific Sensor that provides the context values, for instance: *ContextMonitoringM.timeNow* or *StatisticalAnalysisM.dataSufficiency*

(b) For statistic-calculation-variable declaration, a reference to the Statistical Analysis Module middleware module and the specific notation that provides the statistic function, namely:

itemReference = StatisticalAnalysisM.parameterID.periodID.statisticTermID **(2)**

where:

parameterID is the variable whose values are evalueated statistically

periodID denotes the period of time over which the values will be evaluated

statisticTermID denotes the statistic function to apply on the selected values.

The Statistical Analysis Module supports several "periods" (repeated activity episodes), like: Recheckcycle, Bindingcycle, Session, Dischargecycle, Daily, Monthly

and several "statisticTerms" (statistic functions) over the values of parameterID concerning theb periodID, like: Sum, Max, Min, Avg, Stdev, UpperConfLim etc.

(c) For QoS adaptation-variable declaration, a reference to the software component and the specific parameter it serves, namely: *BindingM.bitrateQoSReq*

(d) For calculation-variable declaration, the value (for a constant) or the formula/expression that assigns a value to this variable as a function of known entities (i.e. of the working memory data).

(e) For configuration-variable declaration, a reference to the software component and the specific parameter it serves, for instance: *ContextMonitoringM.recheckRate*

(f) For auxiliary-variable declaration, a reference to the software component and the specific parameter it serves, for instance: *AdaptationM.costUseratio*

4. Declaration Notation Assigning a Variable to another Variable

The declaration follows the format below, with eq. Symbol the **"::"**:

Repository-Name1.variable-Name1 == Repository-Name2.variable-Name2;
(3)

or, for already declared variables: variable-Name1 & variable-Name2,

variable-Name1 :: Variable-Name2;
(3a)

where: **variable-Name1 & variable-Name2** are the names of the two variables,

repository-Name is the name of the corresponding variable repositories.

The new variable should be of the same type and units with the corresponding initial variable, Otherwise see Declarations Rule 3, at the RHS of the Declarations statement, the appropriate modifier expression should be added. In this way, a service request QoS variable may be assigned to a global variable.

// Example: Assigning a new name to a variable

ServiceRequests.Request1.qosVariable1 == Global.qosVariable2;

Request1.qosFarenheit == Global.qosCelsius*(qosFarenheit - 32) + 5;

Figure 5-1 – Variable Declarations Example

```
Declarations{
// 1. Context-variables Declaration Statements
percentage ContextVar battery  == ContextMonitoringM.batterySensor;
percentage ContextVar bandwidth == ContextMonitoringM.bandwidthSensor;
string ContextVar userPref == ContextMonitoringM.userPrefSensor;
time ContextVar timeNow == ContextMonitoringM.timeNowSensor;
time ContextVar startTime == ContextMonitoringM.startTimeSensor;
string ContextVar activeRequest == ServiceRequestM.activeRequest;
float ContextVar batteryDischargecycleMax ==
StatisticalAnalysisM.battery.Dischargecycle.Max;
float ContextVar cloudCostMonthly ==
StatisticalAnalysisM.cloudCostRecheckcycle.Monthly.Sum;
// 2. Adaptation-variables Declaration Statements
integer ConfigVar recheckRate == ContextMonitoringM.recheckRate;
Boolean ConfigVar renegotiate == BindingM.renegotiate;
percentage QoSVar bitrateQoSReq == BindingM.bitrateQoSReq;
float QoSVar costQoSReq == BindingM.costQoSReq;
float ServiceRequests.activeRequest.costQoSReq1 == BindingM.costQoSReq1;
percentage ServiceRequests.activeRequest.bitrateQoSReq1==
BindingM.bitrateQoSReq1;
// 3. Calculation-variables Declaration Statements
integer CalculationVar maxBandwidth == 384;
integer CalculationVar cycleStepsMax == 3;
// Monthly Data Allowance by the SP in MBs
float CalculationVar monthlyDataAllowance == 4000;
// Monthly Cost Allowance by the SP for services on the Cloud in pounds
float CalculationVar monthlyCloudCostAllowance == 5;
string CalculationVar myName == “PPapakos”;
float CalculationVar duration == timeNow – startTime;
float CalculationVar rediscoveriesPer5min =  rediscoveries*300/(duration + 1);
// 4. Variable Equivalence Declaration Statement
costQoSReq1 :: costQoSReq;
bitrateQoSReq1 :: bitrateQoSReq;
//5. Configuration-variable Declaration
Integer ConfigVar recheckRate == ContextMonitoringM.recheckRate;
float AuxiliaryVar resourceUseratio == AdaptationM.resourceUseratio;
};
```

5. Variables Declarations Rules

The following Declarations Rules should be observed in every policy file, in order to avoid execution errors:

Declaration Rule 1: If a declared context-variable is referenced at the RHS of a Declarations statement of a calculation-variable, then the referenced context-variable should be already declared.

Declarations Rule 2: If an application-specific QoS variable is assigned to a global QoS variable, then it should have the same units (or be dimensionless) and have the same domain with the reference global variable. If units or scale adjustment is required between the two variables, then it is included in the equivalence declaration statement assigning the second variable to the first.

5.5.1 Adaptation-Rules

According to the Volare APSL, in a policy file (global or application) every adaptation-rule is of the paradigm: **If "conditions" Then "action"**. The "action" of each adaptation-rule in Volare is a value assigning statement to the referenced at the LHS adaptation-variable and is called in this work "adaptation-statement" (A/St).

From all the adaptation-rules with the same "**If conditions**", the "If conditions" are grouped together in the common "**Criteria**" conditions group, while the respective "**actions**" (i.e. adaptation-statements) are grouped together as the corresponding "**Subpolicy**". Thus the Volare adaptation-rules are separated and grouped together in Criteria and Subpolicies sections.

5.5.2 Criteria

This section holds the sets of Criteria that enable the activation of each Subpolicy.

Each Criteria starts within the Criteria section of the policy file, and includes predicates with numeric or non-numeric expressions, denoted as follows:

```
Criteria{
    [n1] Criteria Criteria-Name1{
    Condition Numeric expression  [ = | > | < | >= | <= | <> ]  value |
expression;
    …
                            };
    [n2] Criteria Criteria-Name2{
    Condition non-numeric expression [ = | <> ]  non-numeric value;
        …
                            };
    [n3] Criteria Criteria-Name3{
    Condition Boolean [ = | not ] True/False;
        …
                            };
};
```

5.5.3 Subpolicies

The building block of each Policy Rules section of a policy file is the Subpolicy. Every Subpolicy consists of a number of adaptation-statements under the Subpolicy-name, representing multiple adaptation strategies through the values assigned to the adaptation-variables at policy execution time, when the Criteria conditions group that corresponds to the Subpolicy is satisfied by the current context.

Each Subpolicy lies within the Subpolicies section of the policy file, denoted as follows:

```
Subpolicies{
        Subpolicy Subpolicy-name1{
            Adaptation-statement1;
…
                };
        Subpolicy Subpolicy-name2{
        …
                };
};
```

1. Adaptation-Statements (A/Sts)

The building block of each Subpolicy is the adaptation-statement, through which the value at the RHS is assigned to the adaptation-variable referenced at the LHS.

Through the Volare operands and keywords, an adaptation-rule takes the general form:

[overridesAsUpperLimit | overridesAsLowerLimit | overridesAll | overrides | yields | yieldsAll] Adaptation-Variable-Name = Adaptation-Variable-Value (value | expression) [“(bindingMargin)”] (weight); (1)

Rule 1: An adaptation-statement is valid if it has at least an adaptation-variable name at the LHS and at the RHS a value/expression to be assigned for the adaptation-variable.

Rule 2: When no value is assigned to the **binding-margin** attribute at the RHS of an adaptation-statement concerning a QoS adaptation-variable, then at policy execution time the **Policy Engine Manager (PEM)** assigns the default value provided by the reserved configuration-variable **defaultBindingMargin** in the global-level policy.

2. Priority Assigning to Adaptation Rules

Priority to each adaptation rule is assigned indirectly through keywords, optionally prefixed at the LHS of adaptation-statements, namely the mutually exclusive: “overrides” or “yields” keyword or the lack of it and the policy file of origin. From the matched adaptation rules at any Policy Engine Cycle, only those of the highest priority level are to be selected at the “select Rules” Step of the Policy Engine Cycle.

Volare supports seven (7) implied priority levels on adaptation rules in descending order: *overridesAll* + global policy, *overridesAll* + Application policy, *overrides* + Global policy origin, *overrides* + Application policy origin, no keyword (normal priority level), yields + Application policy origin, *yields* + global policy, *yieldsAll* + Application policy, *yieldsAll* + Global policy origin.

3. Expressions at the adaptation-statement RHS

An adaptation-statement may have at the RHS a numeric or Boolean or string type value or expression, depending on the adaptation-variable **(adaptation-variable)** type:

If the **adaptation-variable** represents a numeric variable, then it may be a possibly repetitive combination of the arithmetic operands (+, -, *, /, ^) acting on numeric entities like constants and/or references to the values of Volare context and/or adaptation numeric variables, providing a numeric value.

If the **adaptation-variable** is of Boolean type, then it should also provide a concrete Boolean value or its negation through the **not()** operand.

If the **adaptation-variable** is of string type, then it should provide a string value.

4. Stored Values Term Definition in use in adaptation-rules

The following definitions concern objects used in adaptation-rule expressions (i.e. in both Criteria conditions and/or adaptation-statements):

new values of the context-variables or of the adaptation-variables, is called the new values of the context-variables derived from the Context Monitoring Module or the newly calculated but not yet implemented values of the adaptation-variables,

current value of a context or an adaptation-variable is called the value of the variable that is still in force,

previous value of a context, QoS or non-QoS adaptation variable is called the corresponding value of the variable at the previous implemented adaptation,

lastAdaptation value of a context or adaptation variable is called the value of the variable that is used at the last adaptation in force implemented adaptation,

unadapted value is called the corresponding value for a QoS variable that belongs to the un-adapted service request launched by the application.

These values may be used in Criteria conditions or expressions in adaptation-statements, like: **bitrateQoSRequest = 0.8*(bitrateQoSReq.unadapted + bitrateQoSReq. current)/2 +**

0.1*(bandwidth.new + bandwidth.current + bandwidth.previous)/3;.

Note 1: The Volare Policy Engine maintains log of the current-values of the context and the adaptation-variables as well as of the previous, the new and the unadapted values as the “working memory” of the Policy Engine, so that they may be used in formulas, in Criteria and in the RHS expressions in adaptation-statements when required.

5. Stored variables values in a RHS expression of adaptation-rules

Rule 1: In any adaptation-statement or Criteria conditions, by convention, whenever at the RHS within an expression context or adaptation-variables are referenced instead of constants, then unless otherwise clearly expressed through the special notation above:

a) the values of the context-variables referenced in the RHS are the **new** values**,**

b) the values of the QoS adaptation-variables that may be referenced in the RHS within an expression are the **unadapted** service request values,

c) the RHS calculated value is the **execution-value** for the adaptation-variable referenced at the LHS of this adaptation-rule.

5.5.4 Application-level Policy

The application-level Policy (one for each application) deals with adapting the QoS levels of specific application service requests for services on the Cloud. Each application-level Policy consists of several Subpolicies that specify different adaptation-rules, suited to different context situations and representing different adaptation strategies.

The application policy**-name** takes the specific service request name such as "**Video-streaming**".

The **Subpolicy-names** are linked to the **Subpolicies** specified in the Policy Rules section with each Subpolicy corresponding to a Criteria conditions group of the same name. At policy execution, when all the Criteria conditions are satisfied, the appropriate Subpolicies are **matched** and their adaptation-statements form the **conflict set** and are to be further evaluated for *selection and execution*.

Each policy file may have at each Consecutive Execution Cycle up to one default Subpolicy, named **Default**. The **Subpolicy Default** – one in each policy file and CEC – adaptation-statements at policy execution are always matched at any context conditions.

1. Criteria for an application-level Policy

In this example, in addition to the Subpolicy Default, the HighQuality Subpolicy can only be matched when the HighPerformance Subpolicy's criteria are also satisfied. In that case at policy execution, both the HighQuality and HighPerformance Subpolicies adaptation-rules are matched and will be considered for selection and execution. At policy execution the Volare Policy Engine Manager evaluates sequentially each Criteria in the Criteria section of the Composite-Policy based on the current context. Once a Criteria conditions group are satisfied, then the Subpolicy and its adaptation-statements are matched and they will be further evaluated for selection and execution. Multiple Subpolicies may be matched at the same time from the global-level and the application-level policies, thus creating a composite-agenda for selection and execution through participative weighted contribution to the adaptation results.

However, in the above example depending on the memory context-variable, it is possible that only HighPerformance and of course any Subpolicy with “default” condition will be matched, at which point the adaptation will be the one specified by the selected adaptation-rules of those two Subpolicies only.

Figure 5-2 - Criteria Section of an application-level Policy File

```
Criteria{
[1] Criteria HighQuality{
          bandwidth > 50;
          bandwidth > 40;
          case{
          userPrefr = “LowCost”;
          or
          userPref = “SaveBattery”;
               };
     };
[2] Criteria HighPerformance{
          bandwidth > 50;
          battery > 50;
          userPref <> userPref.lastAdaptation;
     };
..
};
```

2. Subpolicies

In an application policy the Subpolicies section specifies the actual adaptation behaviour for the service to which the policy corresponds. Which Subpolicies are matched depends on the Criteria section specified above.

Note 1: Only adaptation-variables concerning the service request can be adapted by the middleware, not the internal application behaviour or the application configuration.

Example: Subpolicies section of an Application Policy File

Subpolicies {

Subpolicy DefaultApp{

```
            qosVariable1 = qosVariable1* constant1 * ContextVariable1 (wSgen*0.50);
            qosVariable2 = value2 [15] (wSres*0.25);
            overrides qosVariable3 = qosVariable3* constant2 (wSdisr*0.40);
            overrides qosVariable1 =  value31 (wScost*1.00);
            qosVariable1 = value4 [20] (wSres*0.80);
            yields qosVariable2 = value5 (wSperf*0.50);
                          };
      Subpolicy Subpolicy-name1{
            overrides qosVariable1 = contextVariable1 [12] (wSperf*0.20);
            qosVariable2 = Value5 [15] (wSres*0.20);
            overridesAsLowerLimit qosVariable2 = 0.1*bandwidth [0] (wSgen*0.25);
                                      };
      Subpolicy Subpolicy-name2{
            yields qosVariable1 = qosVariable1* value3 (wSres*0.50);
            qosVariable2 = value4 [35] (wSperf*0.30);
                                       };
      ...
};
```

5.5.5 Global-level Policy

The global-level Policy takes the name “Global” and deals with adapting:

global QoS variables, the behaviour of the Volare middleware and of the adaptive Service Discovery and Binding on the Cloud Functionality offered to independent applications to the current context, through adaptation of the configuration-variables. It consists of the Variables Declaration section, the Criteria section and the Subpolicies section. It consists of several Subpolicies that specify different groups of adaptation-rules, suited to different situations.

Note 1: In the global policy Variables Declarations section, all the global context and configuration-variables are declared, so that the application-level developer may only need to declare only application-specific context or application-specific QoS adaptation-variables.

1. Criteria of the Global Policy

Figure 5-3 – Global Policy Criteria

```
Criteria{
    [1] Criteria Subpolicy-Name1{
        battery > batteryLevelHigh;
        bandwidth < bandwidthLevelHigh;
 userPref = “Normal”;
        or
        battery > 40;
        bandwidth > bandwidthLevelMedium;
 userPref = “HighBirtrate”;
                    };
    [2] Criteria Subpolicy-name2{
        battery < 40;
        case{
            userPref <> “Normal”;
            or
            userPref <> “HighQuality”;
            };
        };
    ...
};
```

The criteria for the global-level policy are structured identically to the application-level one, only the policy name is “**Global”.** Same rules that apply to application-level policy criteria also apply to global-level policy criteria.

2. Subpolicies of the Global Policy

The major differences between an application-level and a global-level policy are:

The global policy manages the Volare middleware configuration variables, affecting its behaviour and the SD & Binding Functionality. If no global-level policy is present, Volare will simply not adapt its behaviour. At the same time, the global policy assigns default values to the global QoS adaptation-variables.

The application policy may only affect the SR QoS adaptation-variables.

Note 1: As it can be seen, Volare is capable of modifying the middleware behaviour (self-adaptation), as well as adapting global QoS variables and their attributes (the binding threshold margin) and is able to adapt to the current context the parameters of the adaptive Service Discovery and Binding Functionality.

Figure 5-4 - Global Policy Synoptic Example

```
Policy Global{
Subpolicies{
    Subpolicy Default{
            overrides qosVariable1 = qosVar1* value1  [20] (wScost*0.25);
            qosVariable2 = value2 [15] (wSgen*0.30);
            VolareSetting1 = value3 * contextVariable2 (wSdisr*0.40);
            overrides VolareSetting2 = booleanValue1 (wSres*0.30);
            yields qosVariable1 = value4 (wSperf*0.20);
                    };
        Subpolicy Subpolicy-name1{
            yields qosVariable1 = qosVariable1* value4 [15] (wSres*1.00);
            qosVariable2 = value [35] (wScost*0.20);
        overridesAsUpperLimit VolareSetting1 = value51 (wSperf*0.80);
            VolareSetting2 = booleanValue2 (wSgen*0.50);
                   };
        …
};
```

Note 2: As defined in the application-level policy specification language, keywords such as **overrides** and **yields** have different priorities when in the global-level policy. Specifically, **overrides** in the global-level policy will always take precedence over the application layer. On the other hand, **yields** will always yield in favour of the application layer specification.

5.6 Conflict Resolution Mechanism

When more than one **execution-value** is provided for an adaptation-variable, then we have a conflicting set of **execution-values.** A conflict prevention strategy at policy execution is enabled by the Volare middleware through the following rules and constraints:

The global variables may be common between the application-level and global-level policy.

A conflict resolution methodology is also structured and implemented by Volare, based on the principles, approach and rules described below.

There is a significant possibility of intra-policy conflict as well as of inter-policy conflict between application and global-level policies, caused whenever there is more than one execution-value for an adaptation-variable provided by the selected and executed adaptation-statements of the Composite Policy.

The Volare policy specification language provides to the developer the following powerful techniques, supporting the conflict resolution strategy in calculating the final **resolved-value**.

5.6.1 The Volare Conflict Resolution Directives

Once the appropriate rules are matched from the Composite-Policy based on the "matched Criteria", we are left with the Matched-Composite-Policy, which includes all the appropriate rules with Criteria conditions satisfying the current context data, but without the necessary conflict resolution.

The next step is to execute the Conflict Resolution Directives (CRDs), leading to the final list of adaptation rules to be executed, the Agenda. The following Conflict Resolution Directives are applied, in sequence at policy execution in order to calculate the **resolved-values** of the adaptation-variables:

CRD1. At the "Select Rules" step of the Policy Engine out of the Matched-Composite-Policy adaptation-rules, the Conflict Resolution Directives for rule selection select for each head-predicate, only the highest priority adaptation-rules.

The **priority level** that is derived for each A/St by the prefixed (or missing) keyword and the policy file of origin is as follows:

1 "overridesAll" + global policy

2 "overridesAll" + Application policy

3 "overrides"

4 no keyword

5 "yields"

6 "yieldsAll" + Application policy

7 "yieldsAll" + Global policy

0 "overridesAsUpper (or Lower) Limit

All the lower priority adaptation-rules on the same head predicate are ignored and all the highest priority adaptation-rules selected for each adaptation-variable form the **Agenda**.

CRD2. At the “Execute Rules” step of the Policy Engine, all the selected (one or more) adaptation-rules of the **Agenda** on the same head predicate (adaptation-variable) are sequentially calculated and their **execution-values** are provided for “participative weighted contribution to the adaptation results”.

CRD3. At the “Execute Rules” step of the Policy Engine all adaptation-statements on numeric adaptation-variables with the “**overridesAsUpper(or Lower)Limit**” are executed. For each such numeric adaptation-variable, the strictest limit value is retained as the **resolved-limit-value** as follows:

On **UpperLimit**, the smallest value is retained as the resolved-limit-value;

On **LowerLimit**, the highest value is retained as the resolved-limit-value.

Before the “participative weighted contribution” calculation is implemented, for each adaptation-variable with an Upper or Lower-limit-value, all execution-values are modified to conform to the appropriate limit value(s).

CRD4. If at the “Execute Rules” step of the Policy Engine there is only one **execution-value** for an adaptation-variable in the **Agenda**, then this value is adopted as the **resolved-value** for this adaptation-variable, after it is modified to conform to any eventual Upper or Lower-limit-value.

CRD5. If at the “Execute Rules” step of the Policy Engine more than one **execution-values** are derived concerning the same numeric **adaptation-variable**, then the **resolved-value** of the numeric adaptation-variable is calculated as **the weighted average** of all relevant numeric **execution-values**, after they are modified to conform to any eventual Upper or Lower-limit-value for this adaptation-variable.

CRD6. If at the “Execute Rules” step of the Policy Engine more than one **execution-values** are derived concerning the same non-numeric (Boolean or string type) adaptation-variable, then:

The optimal **resolved-value** is the execution-value with the **maximum sum of weights** (Majority Rule on the sum of weight values).

If the weight sums of the two conflicting values are equal, then no application-level adaptation is performed for that adaptation-variable and the **current** value remains.

CRD7. If the sum of weight values of all the selected adaptation-rules on an adaptation-variable is equal to zero, then the next highest priority adaptation-rules are selected and executed.

5.6.2 Participative Weighted Contribution Directives

1. Participative Weighted Contribution Over Numeric Variables

For numeric variables, when an adaptation-variable ***x*** is common among n selected and executed Subpolicies and ***x_i*** represents each of the n **execution-values**, with a weight ***wi*,** the adaptation will calculate as **resolved-value *x*** the simple weighted average of:

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}, \quad 1 <= i <= n \text{ and } i, n \in N \qquad (1)$$

where n is the number of selected Subpolicies where there is a **execution-value** for the adaptation-variable ***x*** and where *xi* denotes the adaptation-variable **execution-values** and ***wi*** the weights.

If the adaptation numeric variable is of type integer, then the equation (1) becomes:

resolved-value *int*(*x*), where $x = \Sigma w_i * x_i / \Sigma w_{i,}$ i = 1 to n, n Є N (1a)

2. Participative Weighted Contribution Over Non-Numeric Variables

When conflict exists on more than one **execution-value** for a Boolean type adaptation-variable, the value with the highest sum of weights will be used.

When the two different **execution-values** have equal sums of weights, then no adaptation is done for that adaptation-variable and the existing value remains.

In case of inter-policy conflict between two or more **execution-values** of a non-numeric adaptation-variable with the **overrides** keyword, then precedence takes the global over the application-level policy value.

If two or more execution-value**s** from adaptation-rules in the same policy have an **overrides** keyword, then the higher weight value becomes the **resolved-value**.

5.7 Policy Files Maintenance

To facilitate the updating of policy files, the Volare APSL allows the developer to write in plain text simple policy updatefiles with the name of the target policy file but ending in "**.upd**". The Policy Update File follows the same structure as a normal policy file, referencing only the section(s) concerned by the changes. The following two keywords are available to specify adding or removing a declaration statement or a Criteria conditions group and/or a Subpolicy or replacing a whole section with a new one: **add/remove{ }** keywords

The update inserts the new text lines that are new or to be removed under the corresponding section title, within the appropriate command: **add** or **remove{ };.**

A Subpolicy or a Criteria group is added and/or removed as a whole. For instance:

Figure 5-5 – Policy File Update Example: Policy-Name.upd

```
Policy Policy-Name{
Declarations{
remove{
float CalculationVar wG13 = 0,17;
                  };
add{
float CalculationVar wG13 = wG17*bandwidth*0.10;
};
Criteria{
add(or remove){
[1] Criteria G1{
...
                      };
                      };
};
Subpolicies{
add(or remove){
Subpolicy G1{
...
                    };
                    };
            };
};.
```

The Policy Files Manager, when notified for a new policy file update, implements Policy File Maintenance. Policy file security issues are ensured through the use of an appropriate ID & password, by the Policy Files Manager.

With the use of the keyword **remove** over each section name of the policy file, even the whole old Policy statements are removed and the new **added** statements allow whole Policy replacement by an updated one.

5.8 Related Work

We will refer mainly to custom-made APSLs for authoring of obligation policies for DCAA for mobile middleware and applications, like CARISMA [6], CHISEL [7], RAINBOW [59], RAM [24], CARE [69], MIMOSA [36].

A general purpose or domain-specific Adaptation Policy Specification Language for Policy authoring is required for policy-based systems, defining the rules model after the Event - Condition-action or simply the Condition-action model, a rule priority assigning mechanism, Conflict Resolution Directives [34][12][44][39], keywords and operators and the rules form.

CARISMA [6], using the Condition-action model, has introduced a micro-economic auction-type Conflict Resolution mechanism, using dynamically updatable application profiles for setting User needs and application resource requirements. CHISEL [7] with rules after the Event Condition Action model, introduces dynamically Policy rules for unanticipated situations.

Additional requirements are needed if adaptation-rules from more than one entity are considered at policy execution.

Few APSLs have some of the features that Volare APSL is introducing. The use of weight values in adaptation-rules has been used in Goal and Utility-based projects, but in different concept, in order to evaluate the utility of a state out of different adaptation dimensions [84]. CARE [69] makes use of a weight value in every adaptation-rule for the purpose of assigning priority value to each rule. The Volare APSL is the only one in our knowledge in the mobile DCAA research field that allows multiple rules to be selected at policy execution and a Participative Weighted-Contribution mechanism.

Concerning the object of adaptation implementation: (a) on the middleware or framework components, like CARISMA [6], HERA [31], Volare [17], Q-CAD [15], ODYSSEY [40], ReMMoC [23] or (b) on the components of the custom-made adaptive application, like: QuAMobile [18], QuA [22], RAM [24], MADAM [18], MUSIC [26], PLASTIC [25], or on both like MobiPADS [20].

MIMOSA [36] and its predecessor CARE [69], although of a different scope to Volare, present a distributed middleware that implements Conflict Resolution on context Profiles and Policies by different (User and Service Provider Operator). In this feature, of aggregating multiple Profiles and Policies and their Conflict Resolution, Volare also includes this functionality but with totally different mechanism.

6 Weight-Based Adaptation Reasoning Technique & Methodology

In chapter 4 the unique or unusual developer-visible features of the Volare APSL were presented – that are also supported by the middleware – with most important ones: (i) possibly multiple rules selectable at any context instance on the same adaptation-variable – each with a weight function value, (ii) two policy support with inter-policy conflict resolution, (iii) hierarchic multi-cycle policy execution, (iv) notation for statistical inference based on the usage data on policy -specified parameters.

In chapter 3 "Project Overview" the motivation and requirements have been discussed for a novel rule-based adaptation reasoning technique (ART), which employs the Volare APSL features supported by the DCAA middleware leading to specific advantages on adaptation policy authoring in comparison to the action-based technique, like:

- Application-transparent adaptation according to the current context and the adaptation logic requirements and goals of the SR on CSD by a mobile application independent to the middleware, without any code change and with minimal burden on the application policy developer,
- Adaptation logic balancing the goals and constraints of all three stakeholders: the middleware policy developer, the application policy developer and the user.
- High behavioural (parameter adaptation) variability with quasi-linear instead of combinatorial increase to the number of rules, in comparison to an equivalent action-based policy.
- Capability for dynamic fine-tuning of the adaptation logic by the user without policy updates by integrating a configurable User Preferences Model.
- Easy introduction in the adaptation logic of unanticipated at middleware design time quantitative Long Term Performance Goals (LTPGs) over finite horizons.
- Fine-grained, gradual adaptation.

Since a main and distinctive feature of the Volare approach for rule-based adaptation policy development consists in assigning a weight value to every adaptation-rule, denoting the relative importance of its execution-value at policy execution, the proposed ART for policy design & implementation is called in this work: ***weight-based adaptation reasoning technique*** or shortly ***weight-based***

technique – abbreviated as ***WBART*** – to distinguish it from the action-based, goal-based or utility-based ARTs.

Reference is made in this work for comparison to policies designed in the action-based adaptation reasoning technique – which is also rule-based – since the goal-based and the utility-based techniques, as described in subsection §2.3, follow a different and more burdensome approach for policy development and for middleware operation requiring at policy execution the evaluation of all the system states, and are recommended mostly for complicated scenarios with dynamic service composition, which are outside the frame of this work.

The following subchapters describe an outline of the weight-based technique and the relevant methodology that has been developed to assist the policy developer in policy authoring.

6.1 The Weight-Based Adaptation Reasoning Technique

The three established adaptation reasoning techniques (ARTs) described in §2.3 are all based on the principle of a single adaptation-rule that may be selected at policy execution for each adaptation-variable.

Note that every adaptation-rule selectable at policy execution at any contextual situation instance represents a complex adaptation decision, as it needs to represent multiple adaptation interests. It is argued that each such “situation-action” conventional adaptation-rule is equivalent in adaptation results to a set of multiple selectable adaptation-rules built in the weight-based technique and stemming from different targeted adaptation reasoning viewpoints that in this work are called *adaptation-concerns.* Thus at each contextual “situation” where the action-based policy would have only one adaptation-rule selectable for each head predicate, a weight-based equivalent policy may have more than one, with each adaptation-rule serving the goals of an *adaptation-concern*.

In this way the weight-based technique for the development of an expressive adaptation policy, aims to specify at any considered contextual situation the required adaptation actions by developing instead of just one, possibly multiple adaptation-rules expressing every associated *adaptation-concern* through possible the same adaptation-variables.

This main and unique feature on purpose introduced in the Volare APSL, consisting in allowing possibly multiple selectable adaptation-rules per adaptation-variable serving different *adaptation-concerns*, lays the foundation of the WBART through specification of the adaptation actions at each context sub-domain (i.e. the adaptation-rules) separately for each associated *adaptation-concern*. Of course for simple adaptation actions representing only one *adaptation-concern*, one adaptation-rule may be sufficient at any context sub-domain.

6.1.1 Weight-Based Adaptation Concepts

The following definitions are necessary to describe the Volare weight-based technique on policy development for middleware-driven dynamic context-aware adaptation of the SR for CSD by mobile applications.

Remark: *Compound words that are defined and used in this work with a specific interpretation are spelled with a hyphen between them. For instance, statistic-calculation-variables are a specific category of calculation-variables declared in a policy according to the APSL notation for policy-based statistic inference on usage data.*

Context & Adaptation Profile *of a mobile scenario for middleware-driven DCAA is called in this work the context, adaptation and intermediate variables declared and used in the Policy (context-variables, QoS adaptation-variables, middleware configuration-variables, calculation-variables, etc.) and their data structure.*

Adaptation Space *of the middleware is called in this work the set of adaptation objectives concerning the middleware scope, for the given context & adaptation profile and middleware expected operation.*

Adaptation-Concern*, also abbreviated as* **adaptation-concern,** *is called in this work every one of the developer selected major minimally overlapping specific adaptation reasoning viewpoints, each with a feasible adaptation objective in conformity to the scenario context & adaptation profile. These adaptation-concerns are interpreted to cover the whole scenario adaptation space, called the* ***Adaptation-Concerns Model****.*

Note*: As an example, CPU optimization is a perfectly reasonable adaptation reasoning viewpoint. However, if the scenario context & adaptation profile does not monitor or control CPU, then it does not represent a feasible adaptation objective and should not be defined as an adaptation-concern but it can be represented by the nearest in scope Resource Use Optimization adaptation-concern.*

Adaptation-Strategy*, also abbreviated as* **adaptation-strategy,** *is called in this work the collection of adaptation-rules within a policy file that serve a specific* ***adaptation-concern*** *over the whole valid context domain.*

***Note**: The adaptation-concern and adaptation-strategy concepts are different but in a one-to-one mapping. The adaptation-concern is a viewpoint of interest to the developer with a feasible adaptation objective. The adaptation-strategy is the entity that represents the collection of adaptation-rules in the policy file that serve this adaptation-concern over the valid context domain.*

***Weight function** value of every adaptation-rule is named in this work a non-negative value less or equal to 1, representing the relative importance that the execution-value of the adaptation-rule carries at policy execution to the adaptation results, according to the APSL participative weighted contribution (PWC) procedure, as described in §5.6.*

***Variation Point (VP)** of the considered adaptable system, is called in this work every specific concern of structural or algorithmic or operation mode or parameter adaptation character for which more than one mutually exclusive alternative sets of actions are executable at adaptation at a context instance, while retaining the same functional properties.*

***Structural or Algorithmic Variation Point (SAVP)** of the considered adaptable system is called in this work every VP of structural or algorithmic character (i.e. where the alternative mutually exclusive alternative sets of actions are active-on-demand components or algorithms).*

***Behavioral Variation Point (BVP)** of the considered adaptable system is called in this work every VP on behavioral settings (parameter) adaptation for a given MESC.*

***Variants of a VP** of the considered adaptable system are called in this work the mutually exclusive alternative sets of adaptation actions concerning a VP that are executable at adaptation at a context instance. The variants of a VP are expressed in a weight-based policy by alternative mutually exclusive Criteria - Subpolicies within an adaptation-strategy. Depending on the character of a VP, its variants are called Structural or Algorithmic Variants (SAVs) or Behavioral Variants (BVs).*

***Mutually Exclusive System Configuration (MESC)** is called in this work every alternative configuration of the considered adaptable system with a different structural or algorithmic variant.*

6.1.2 The Generic Adaptation-Concerns & Strategies Model

The generic Adaptation-concerns & Strategies Model is described below, developed by the weight-based methodology, covering the adaptation space with the following selected five basic adaptation-concerns:

1. Generic Operational Constraints Adaptation-Concern

This adaptation-concern includes the adaptation interests concerning generic constraints and global invariants as well as default values for parameter adaptation. The respective adaptation-strategy is denoted as ***Sgen***. It concerns generic adaptation actions independent of the user preference mode.

Practically, the middleware should be able to operate at a basic level without optimization or fine-grained adaptation or integrated user preferences – only with the Generic Operational Constraints adaptation-strategy rules.

2. Performance Optimization Adaptation-Concern

It concerns adaptation interests aiming at optimal performance specifying QoS requirements according to the selected user preference mode. The respective adaptation-strategy is denoted as ***Sperf***.

3. Resource Use Optimization Adaptation-Concern

It concerns adaptation interests aiming at resource use optimization, like battery power, CPU, RAM, storage memory. The respective adaptation-strategy is denoted as ***Sres***.

4. Cost Optimization Adaptation-Concern

It concerns adaptation interests related to optimization on cost-related parameters or goals, like the cost of binding to the cloud, respecting the credit allowances set by the CSPs, respecting the *priceMax* policy ceiling value for cloud services, etc. The respective adaptation-strategy is denoted as ***Scost***.

5. Disruption Minimization Adaptation-Concern

It concerns adaptation interests for minimizing operation disruption at runtime by delays, rediscoveries or frequent change of QoS due to bandwidth drops or not unnecessary rediscovery/rebinding etc. The respective adaptation-strategy is denoted as ***Sdisr***.

6.1.3 Importance of the Adaptation-Concerns Model

Once the scenario relevant adaptation-concerns are selected, in the depth that is of interest to the scenario, the corresponding adaptation-strategies are also automatically defined by their adaptation-concerns. In Table 6-1 the generic

Adaptation-concerns Model is depicted with the corresponding adaptation-strategies, which is used at policy authoring of the case study.

1. Reasons for defining the Adaptation-concerns Mode

The determination of the scenario Adaptation-concerns & Strategies Model plays an important role in policy design in the WBART, for the following reasons:

Table 6-1 – The Generic Adaptation-Concerns & Strategies Model

Adaptation-Concern	Corresponding Adaptation-Strategy
Generic Operational Constraints	Generic Operational Constraints Adaptation-Strategy
Performance Quality	Performance Optimization Adaptation-Strategy
Resource Use Optimization	Resource Use Optimization Adaptation-Strategy
Cost Optimization	Cost Optimization Adaptation-Strategy
Disruption & Delays Minimization	Disruption Minimization Adaptation-Strategy

- Since possibly multiple adaptation-rules on the same adaptation-variable may be selected at policy execution, it makes clear for which different specific concern every rule stands for, although more than one different rules may specify action on the same adaptation-variable.
- It determines the major adaptation reasoning viewpoints around which the global and each application policy will be structured, and on which viewpoints it can be most easily fine-tuned by the user or later modified or enriched.
- Since in the weight-based technique policy development is implemented as a synthesis of overlapping independent policies (the adaptation-strategies), each serving an adaptation-concern, determining the focal viewpoint for every adaptation-strategy is crucial for policy development.
- It allows seamless integration of the User Preferences Model in the policy, since the weight values of all adaptation-rules in every adaptation-strategy,

vary uniformly with the user preference mode. For instance by setting the user preference mode from “Normal” to “LowCost”, the execution-values of the selectable adaptation-rules of the Cost Optimization adaptation-strategy take higher importance at policy execution, influencing the final adaptation results.

2. **Adding New Adaptation-Concerns to the Generic Model**

Each adaptation-concern of the generic Adaptation-Concerns & Strategies Model may be extended, at the global policy developer’s discretion if the Context & Adaptation Profile supports it, in more specific adaptation-concerns thus permitting more fine-grained adaptation space segmentation. For instance the Resource Use Optimization adaptation-concern may be analyzed in the adaptation-concern for RAM and for CPU and for battery use optimization, if the scenario Context & Adaptation Profile supports feasible adaptation objectives. The advantage for each additional adaptation-concern & adaptation-strategy would be the introduction of an additional set of strategy-weight-coefficient (SWC) values, one for each user preference mode, which would allow more fine-grained adaptation at the cost of additional SWCs at the UPI configuration.

Note: *It should be noted that shrinking the generic model to less than five adaptation-concerns is also allowed, if the developer so wishes. At the extreme, the model may be represented by a single adaptation-concern covering the whole adaptation space and in this case the policy imitates an action-based policy, with the policy being a single adaptation-strategy with a single strategy weight coefficient, thus eliminating partly the fine-tuning capability.*

3. **Introducing in the Policy the Adaptation-Concerns & Strategies Model**

The Adaptation-Concerns & Strategies Model is defined at the global policy design stage through comments in the policy and has to be respected by all application policies. It is indirectly expressed through the strategy-weight-coefficients that are introduced at the configuration of the User Preference Interface and through the weight functions of every adaptation-rule, as defined by the recommended Weight Assigning Strategy described in §6.3, in the form:

(weight function) = (wSstrat * wRule);

like: (weight) = (wSgen * 0.80). However, different forms of the weight function may be adopted, as extensively mentioned in §6.3.

6.1.4 Weight-Based Adaptation Reasoning

The fundamental characteristics of the weight-based adaptation reasoning are presented and discussed below:

1. Policy Structured Around Selected Adaptation-Concerns

To every *adaptation-concern* corresponds an *adaptation-strategy*, i.e. a collection of the adaptation-rules serving the relevant adaptation-concern across the valid context domain. Every adaptation-strategy in the policy is built as an independent policy representing an adaptation-concern. At policy execution, the APSL participative weighted contribution directives (PWC procedure) derive the adaptation results according to the weight- and execution-values of all selected adaptation-rules from different adaptation-strategies. An adaptation policy in the WBART may thus be considered as a synthesis of overlapping independent policies, the *adaptation-strategies,* through adaptation-rules over the valid context domain, with each adaptation-strategy serving a specific adaptation-concern.

2. Multiple Rules Possibly Selectable at policy execution

The most innovative feature of the Volare APSL lies on a different approach on the characteristic of conflict resolution directives, by setting the terms and conditions to allow at policy execution possibly more than one matched adaptation-rules of the highest priority and on the same head predicate (adaptation-variable) to be selected and executed. The Volare APSL establishes, for all selected and executed adaptation-rules on the same adaptation-variable, participative weighted contribution procedure to the adaptation results, applying the weighted average rule for the numeric variables or the majority rule for Boolean or string type variables. Thus from possibly multiple execution-values on the same adaptation-variable at policy execution, only one resolved-value will be chosen as adaptation choice per adaptation-variable, either of numeric or of non-numeric type.

Special precautions are taken at policy development so that the possibly multiple selectable adaptation-rules on the same adaptation-variable at policy execution will not lead to inconsistencies, as is analyzed in the next paragraphs.

3. Rules Specifying Variant Selection

This thesis focuses on the difference between the two paradigms at policy execution: the **action-based adaptation reasoning with single selectable rule** vs. the **weight-based adaptation reasoning with possibly multiple selectable rules per adaptation-variable**. In both cases, a single resolved value out of the single or multiple selected rules for every adaptation-variable will be provided. The great difference between the action-based and the weight-based technique in string or Boolean variant selection needs to be noted, typically representing a structural or algorithmic or operation mode variant selection process.

a) Action-based variant selection

In an action-based policy the selectable rule for each adaptation-variable will be matched and then selected and executed by its priority and its predicates that are satisfied by the current context, in a policy cautiously designed to have a single rule selectable per adaptation-variable at any context instance. If additionally, the rule represents a structural variant that requires also parameter adaptation, then the parameter adaptation settings may be assigned through rules on the relevant adaptation-variables with the same predicates and again a single rule for each adaptation-variable will be selected for parameter adaptation.

b) Weight-based variant selection

In a weight-based policy it is possible to have more than one adaptation-rules of the same priority and with predicates satisfied by the current context that are selected on the same adaptation-variable. Then the participative weighted contribution directives (PWC procedure), through the weight- and execution-values for each adaptation-variable, will select the resolved-value.

This is the **weight-based variant selection paradigm** – which differs from the action-based one – by allowing all rules expressing mutually exclusive feasible variants under the current context – to be selected through priority level and predicates. Then through the APSL PWC procedure an additional weight-based selection level is introduced, reflecting the current user preferences through the weight values of all the selected rules on the same adaptation-variable.

Weight values in a weight-based policy may change with the current user preference mode. Consequently, the weight-based variant selection technique introduces a more subtle selection of the most *user-desirable* variant among feasible ones under the current context, allowing the current user preference to influence – not deterministically – the adaptation choice.

This characteristic is useful in cases where more than one of several mutually exclusive variants are technically acceptable (feasible) under the current context and fine-grained decision-making is required for final selection on user preferences through the weight values.

c) Adaptation Reasoning Differences at Specifying Variant Selection

In the action-based policy, the developer would introduce additional selection predicates, so that only a single most appropriate variant will be selected with design time criteria.

In the weight-based policy, the developer does not need to specify in such detail the predicates for the variant selection adaptation-rules. It is only needed to specify the predicates allowing the feasible variants to be selected. Then at policy execution, the rules for the one or more than one feasible variant will be selected and the PWC procedure will select as most appropriate to the current user preferences, the one with higher sum of weights.

6.1.5 Specifying Mutually Exclusive Variants

The weight-based methodology for describing 2 or 3 or eventually n mutually exclusive variants of a VP, suggests specifying the n-1 variants through mutually exclusive Criteria conditions and leave as default (in the default Criteria-Subpolicy) the n_{th} variant (the one requiring the most complicated criteria conditions) to be selected by default when all other variants are not selected.

At any context instance the adaptation-rule representing the n_{th} variant is by default selected, and possibly another adaptation-rule if its Criteria conditions are satisfied. Two cases are possible:

a. **Numeric Variant Selection –** If the associated adaptation-variable representing the nth variant is numeric, then the adaptation-rule should have a lower priority level than the rules with the other variants, so that it will not be selected unless it is the only rule matched for this adaptation-variable.

b. **Non-numeric Variant Selection** – If the associated adaptation-variable representing the nth variant is of Boolean or string type, then it is sufficient for the default variant adaptation-rule to have the same priority level but lower weight value than the adaptation-rules of the other variants, since through the PWC procedure, the higher weight value adaptation-rule will determine the variant selected.

By using the weight-based variant selection, the developer may prevent intensive predicate specification for all adaptation-rules, and at the same time make sure that in any case only one of the mutually exclusive variants will be selected, as in the global policy example:

```
 [2] Criteria G2_DEFAULT{
         Default;
 };
// 2nd CEC: At High Battery Use Ratio, high attainable values allowed for QoSvars
```

```
  [2] Criteria G2_BATTERYLTPG_HIGH{
         batteryRefRate > 100;
battery >= 50
         or
         userPref = "HighQuality";
  };
  // 2nd CEC: At Very Low Battery level, abrubtly reduced attainable values for QoSVars
  [2] Criteria G2_BATTERYLTPG_VLOW{
         battery < batteryEmergencyLevel;
         userPref <> "HighQuality";
         or
         battery < 1.5 * batteryEmergencyLevel;
         batteryRefRate > 100;
        userPref <> "HighQuality";
  };
    [2] Subpolicy G2_DEFAULT{
          batteryLTPGVariant = "LowBat" (wSres*0.20);
    };
```

6.1.6 Ensuring Policy Consistency

Eventual selection of incompatible adaptation-rule(s) on structural or parameter adaptation may lead to severe inconsistencies. This danger risks to be accentuated due to the weight-based mechanism for SV variant selection through the multiple selectable rules that the WBART introduces. The solution to this problem lies on the principles established for consistency by the weight-based technique at policy design in order to eliminate the danger of inconsistent adaptation-rules selected and executed, exploiting the specially developed **multi-cycle policy execution** feature of the Volare APSL:

1. Pre-determined adaptation-variables

The adaptation-variables declared and used in adaptation policies conforming to the Volare approach are only: (i) the pre-determined middleware configuration-variables, (ii) the global QoS variables common for all service requests, and (iii) the application-specific QoS variables of the service request. No direct programming commands are allowed.

2. Participative Weighted Contribution Directives (PWC procedure)

By “adaptation-rule selection and execution” at policy execution in the Volare approach, it is meant that each selected adaptation-rule will provide its execution-value on its adaptation-variable. The resolved value for every adaptation-variable is established of all derived execution-values by the middleware Policy Engine according to the APSL conflict resolution PWC procedure, as described in §6.5.

Resolved-value selection according the PWC procedure for a string or Boolean type adaptation-variable representing a structural/algorithmic or operation mode SV, will be the execution-value with the higher sum of weights. Resolved-value selection according to the PWC procedure for a numeric variable will be the weighted average of the execution-values.

3. Directives on structural adaptation in the weight-based technique

The following two directives are applied for specifying adaptation-rules on structural adaptation in the weight-based technique:

a. The adaptation-rules specifying structural or algorithmic or operation-mode variant selection are to be evaluated for matching, selection and execution at the first CEC at each multi-cycle policy execution. If there are dependencies (structural sub-variants) then the appropriate adaptation-rules for structural sub-variant selection may be specified at the next CEC.

b. Adaptation-rules on a Structural VP for variant selection may be specified in two ways: Firstly, by mutually exclusive Criteria conditions or priority levels so that only a single rule is selected indicating the selected variant, as in the action-based policy. Alternatively, multiple adaptation-rules representing variants of a SVP that are valid under the current context may be selected at policy execution. In this case, weight-based selection of the most appropriate SV will be implemented by the Policy Engine according to the PWC procedure, based on the weight values of the selected adaptation-rules under the current user preference mode.

4. Directive on parameter adaptation in the weight-based technique

Structural, algorithmic or “operation mode” variant selection at policy execution typically requires the appropriate parameter adaptation through relevant rules specifying parameter settings. However, in a weight-based policy with multiple selectable rules at policy execution, the selected rules for the usually numeric variables that specify parameter adaptation may also represent the different alternative feasible variants whose predicates were satisfied by the current context. Consequently, at policy execution not all selected rules on behavioral (parameter) adaptation may be compatible with the variant to be chosen, since they may represent mutually exclusive variants, thus raising the danger for incompatible rules selection.

The following directive is applied for specifying adaptation-rules on parameter adaptation in the weight-based technique:

a. Adaptation-rules specifying behavioral (parameter) adaptation on a structural/algorithmic or operation-regime variant, should be assigned a cycle-declaration subsequent to the one(s) at which the SV(s) are chosen.
b. Adaptation-rules specifying behavioral (parameter) adaptation on a structural/algorithmic or operation-regime variant are to be specified with additional variant-specific or MESC-specific predicate, determining to which variant(s) or MESC(s) are compatible. This requirement ensures that these may be matched, selected and executed at policy execution, only after the associated SV or MESC has been chosen in the previous CEC.

In this way at the systematically established by the weight-based technique multi-cycle policy execution, first the most appropriate SV(s) will be selected under the current context. Then at the next CEC(s) only the behavioral (parameter) adaptation rules compatible to the already selected in the previous CEC(s) MESC will be matched, selected and then executed, providing the parameter settings through the PWCP on each relevant adaptation-variable. If the policy has no structural or algorithmic or operation-mode variant, but only a single configuration and there are no dependencies between the adaptation-variables, then the weight-based policy can be designed for execution in a single CEC.

6.1.7 The WBART Methodology - Generic Models & Procedures

The Volare approach in order to facilitate the development of adaptation policy logic, has developed a methodology and makes available several policy authoring methodological tools**,** like generic models, procedures and an offline testing & verification Policy Engine Simulator and associated tools specially designed to assist the adaptation policy developer. These policy development tools are referenced below in the order they are usually used in conceptual design and policy development.

1. Generic Models Assisting Policy Design & Implementation

Four generic models relevant to the mobile scenario have been designed, each on a different aspect of policy development. Every one of these models can be adopted or be adapted (modified, enriched or truncated) to the current scenario.

They provide - to a certain extent - generic conceptual design for the adaptation policy logic in appropriate textual form, together with the global policy file generic declarations part, which can be easily adapted by the developer to a modified scenario case. These generic models or procedures are:

a. The generic “Context & Adaptation Profile”, which is described in the case study in §7.3.
b. The generic “Adaptation-concerns & Adaptation Strategies Model”, already described above in §6.1.2.
c. The configurable “User Preferences Model”, described in §6.4.
d. The generic LTPG Design Model, that is described in §6.5.

2. Generic Procedures Assisting Policy Design & Implementation

Similarly, methodological procedures for adaptation policy authoring according to the weight-based paradigm have been developed to assist easy modification of the generic models and authoring of the global or application policy file. These generic or procedures are:

a. The recommended Weight Assigning Strategy (WAS), that is described in §6.3.
b. The User Preferences Model configuration procedure described in §6.4.
c. The WBART synoptic Policy Authoring Procedure, described in §6.7.

3. Testing & Verification Tools Assisting Policy Design & Implementation

The simulated policy execution & Verification application “PEVapp” has been developed for offline (of the mobile device) for testing, verification and evaluation of the adaptation policy by simulated policy execution on automatically generated test suites. It is described in §6.6 and in Appendix B, while the PEVApp User Guide is in Appendix E.

The methodological policy development tools referenced above are roefly described in the following paragraphs. All the above tools provide assistance for the development of the adaptation policy logic but are recommended, not obligatory.

Based on the APSL innovative features described in the previous chapter, the WBART characteristics on policy design for scenarios of mobile middleware DCAA of the SR on the cloud by mobile applications are presented below, different to established research practice [18][22][34][56][61].

6.2 Two Level Policy Support

The adaptation logic for guiding dynamic context-aware adaptation of a SR for a cloud service by a mobile application in the Volare approach is based on two policy files. Firstly the global-level policy file, installed with the middleware and specifying adaptation-rules for the middleware components adaptation as well as global (common for all or most SRs) QoS variables. Secondly the (unanticipated at middleware design time) application-level policy file, specific for every application, that manages the adaptation of the SR application-specific QoS variables and may also affect the middleware adaptation.

The middleware, at an active SR on the cloud by a mobile application having an application policy file stored in the Policy Files Directory, automatically parses and merges the global and the application policy file creating a “Composite Policy” that drives the adaptation.

These two policy files, the global and the application policy file for the active application, are typically authored by different entities. The adaptation-rules of the active “Composite Policy” are expected to operate in common, in conformity and without inconsistencies at every CSD session of the respective application.

6.2.1 Influencing adaptation by both policies

As a consequence a major task of the policy developers involved consists first in assigning to the adaptation-rules appropriate priority levels for inter-policy conflict resolution and selection of the appropriate adaptation-rules, as well as appropriate weight values for the PWC procedure to derive the final adaptation choices on possibly multiple selected adaptation-rules on the same adaptation-variable.

Additionally it is expected for each of the two adaptation policy files to influence the adaptation choices on adaptation-variables of common interest, while avoiding the danger of inconsistencies.

One way to avoid eventual inconsistencies might be to separate the control of the global policy on the middleware configuration-variables and of the application policy on the SR QoS-variables. This is allowed, and can be established by the application developer optionally assigning higher priorities on the adaptation-rules for the SR QoS-variables, while the global policy developer may use the “overridesAll” keyword on the adaptation-rules concerning the middleware configuration-variables. However it is considered a severe loss in the adaptation behaviour management by the adaptation logic, if this separation of adaptation “jurisdiction” takes place unnecessarily, as the application policy developer may have an influential role concerning also the middleware adaptation, while the global policy developer may set operation goals and seek them through adaptation of the global QoS-variables. Additionally, it is expected that the application policy through appropriate adaptation-rules will assist at achieving the established LTPGs and will conform to the User Preferences Model specified by the global policy.

The Volare approach recommends that the global policy includes adaptation-rules on the global QoS variables, common for all (or most SRs), and that the application policy establishes adaptation-rules on the middleware configuration-variables of interest to the SR adaptation. Consequently the WBART employs the APSL characteristics at developing the adaptation-rules, in three ways:

a. Enabling inter-policy conflict resolution for adaptation-rules on common “global” QoS variables and middleware configuration variables across the two policies assigning appropriate priority levels through keywords.
b. Enabling each policy to safeguard the normal range on its proprietary adaptation-variables, overriding more restricting or extreme settings by the other policy, by making use of the *overridesAsUpperLimit* or *overridesAsLowerLimit* keyword to set upper/lower limit values ensuring effective operation. In this way, the global policy can safeguard a configuration-variable range against unreasonable increase or decrease by an application policy statement, and the same is true for the application policy concerning the global QoS variables.
c. Enabling participative weighted contribution to the adaptation choices through the weight values for the possibly multiple selected rules on same adaptation-variables, thus allowing joint adaptation influence across both policies.

6.2.2 Facilitating the Application Policy Authoring Task

The weight-based technique declares all context-variables, the middleware configuration-variables and the global QoS-variables as well as the required for common LTPGs intermediate variables at the global policy.

The application policy developer has to focus only on declaring the application-specific variables, while having direct access to use any declared variable of the global policy for adaptation-rule development, thus facilitating its task without re-declaration.

6.3 The Weight-Assigning Strategy (WAS)

The Volare APSL requires a weight function value for each adaptation-rule, representing its relative importance in comparison to the other selectable adaptation-rules on the same adaptation-variable under the current context, expressing also the user preference on the adaptation-strategy importance. This means that at changing user preference mode, the relative importance of all adaptation-rules of an adaptation-strategy may change significantly in comparison to the other strategies.

The Volare APSL does not specify the form of the weight function, which is up to the developer and the scenario. Two approaches are open to the developer for assigning a weight value to every adaptation-rule, described below.

6.3.1 Manual Weight-Assigning

The manual weight assigning allows the developer of the policy file to evaluate and assign the appropriate weight value or expression to each adaptation-rule within the "()" symbol postfixed at the RHS of every adaptation-statement.

Example: ***bitrateQoSReq = 0.8*bandwidth (0.85);***

Note*: However, if the developer inserts the weight value of every adaptation-rule as numeric value instead of formula as described below, the possibility for fine-tuning the policy through the User Preferences Interface and the strategy-specific structure for reviewing a policy file are eliminated.*

6.3.2 Recommended Weight Assigning Strategy (WAS)

The WBART provides a generic WAS for assigning a weight function to each adaptation-rule of the policy based on reasoning and scenario-specific considerations, such as: the relative importance of the adaptation-strategy to which the adaptation-rule belongs, the adaptation-variable "role" in the adaptation-strategy and the current user preference mode. The basic relation for the weight function value of an adaptation-rule is given below by:

rule weight value =

strategy-weight-coefficient x rule-weight-coefficient (1)

*Example: bitrateQoSReq = 0.8*bandwidth (wSgen * 0.50);*

The Volare generic *Weight Assigning Strategy (WAS)* assigns to each adaptation-rule a weight function that is the product of several weight coefficients, each representing a non-negative value less or equal to 1, as explained below.

1. The Strategy Weight Coefficients (SWCs)

The *strategy weight coefficient* (SWC) denoted as **wSstrat** (0 <= wSstart <= 1) for each adaptation-strategy is a value/expression denoting its relative importance in comparison to the others under each user preference mode. Default SWCs values are assigned by the global policy developer for every user preference mode at the UPI configuration.

2. The Strategy Weight Coefficient Modifiers

The *strategy weight coefficients* default values may be customized by the user at the User Choices Profile (UCP) for each application by the *strategy weight coefficient modifiers* denoted as ***uSstrat*.** They take non-negative values that the user may assign through the UPI when defining the **User Choices Profile (UCP)** for each specific application, within margins defined by the developer. The UPI dispatches to the middleware at runtime the user-set values that modify the default values at the current user preference mode:

$wSstrat = wSstrat_{default} * uSstrat$ ***(2),*** where:

wSstrat denotes the current strategy weight coefficient,

wSstratdef denotes the default strategy weight coefficient value, initially set by the developer at the UPI configuration,

uSstrat denotes the respective strategy weight coefficient modifier value (with default value 1), representing a non-negative value number set by the user, within upper & lower limits set by the developer in order to preserve the relative importance for every adaptation-strategy at each user preference.

The SWCs are declared as float type context-variables at the global policy and their current value under the selected user preference mode is provided by the UPI. For example: ***float ContextVar wScost == UPI.wScostdef *uSstrat;***

Consequently, equation (2) becomes for each Adaptation-strategy:

wSgen = Sgen * uSgen **(2a)**

wSperf = Sperf * uSperf **(2b)**

wSres = Sres * uSres **(2c)**

wScost = Scost * uScost **(2d)**

wSdisr = Sdisr * uSdisr **(2e)**

Table 6-2 – Case Study Default Strategy Weight Coefficient (SWC) Values

User Preference ➔	Normal	SaveBattery	LowCost	HighQuality
Adaptation-strategy	N	S	L	H
Adaptation-strategy Sgen Generic Operat.	1.00	1.00	1.00	1.00
Adaptation-strategy Sperf Performance	0.80	0.50	0.50	0.80
Adaptation-strategy Scost Cost	0.50	0.50	0.80	0.20
Adaptation-strategy Sres Resource	0.50	0.80	0.50	0.20
Adaptation-strategy Sdisr Runtime Disruption	0.40	0.40	0.40	0.10

At each user preference mode, through the formulas (2a) to (2e), the UPI dispatches to the middleware as numeric context-variables the current Strategy Weight Coefficients values.

3. The Rule Weight Coefficients

The *Rule Weight Coefficient* denoted as **wcRule** is a non-negative value/expression/function <= 1, independent of the user preference mode, assigned by the developer and expressing the relative importance of the "operating role" of the adaptation-rule in comparison to the other selectable adaptation-rules on the same adaptation-variable, since evidently not all the adaptation-rules in the policy on an adaptation-variable are of equal importance. This value is deduced by the developer by considering issues such as: the context sub-domain that the adaptation-rule represents (i.e. the Criteria conditions of the adaptation-rule), the adaptation-rule implied priority level (through keywords and policy of origin) and the "operating role" of the adaptation-rule.

4. The final Rule Weight Function

Based on the above analysis, the weight function value assigned to every adaptation-rule is the product of the three above described weight coefficients, as specified by relation (1) above. The UPI provides to the middleware the current *Strategy Weight Coefficient* value, thus:

$$\mathbf{wSstrat = wSstrat_{def} * uSstrat} \qquad \mathbf{(2)}$$

The overall weight equation for the adaptation-rule weight **wRule** becomes:

$$\mathbf{wRule = wSstrat * wcRule} \qquad \mathbf{(3)}$$

Equation (3) is the final equation used in each RHS parenthesis of every adaptation-rule, with the **wcRule** (adaptation-rule weight coefficient) preferably assigned by the developer in numeric form, like in the example adaptation-statement of Sgen:

bitrateQoSRequest = 0.7 * bandwidth [20] (wSgen * 0.40);

Note that the developer may add additional factors to the weight function for every adaptation-rule. A variable-weight-coefficient may be added, to signify the relative importance of the specific adaptation-variable to all the rules in an adaptation-strategy that may take a different value in another adaptation-strategy. For instance, bitrateQoSReq may have a different relative importance in rules of the Sperf than I rules of the Scost adaptation-strategy. Similarly, the developer may specify the rule weight coefficient context-dependent.

Remark: *In the recommended Weight Assigning Strategy, the weight coefficients are not context-dependent, at the exception of the SWCs that are user preference mode dependent. Consequently, the weight values express the user's preference on the relative importance of every adaptation-strategy.*

6.4 The Volare Configurable User Preferences Model

As attested by numerous research works [6][41][36][45][69] and surveys [13][61] commented in Ch. 2, satisfactory adaptation of a DCAA mobile middleware to context change requires incorporation of user preferences – even at runtime. In fact, on resource constrained mobile devices with variation in network availability and bandwidth, there can be no QoS guarantee but a compromise between what is attainable and what the user preferences are at QoS or indirectly at resources or cost. User preferences are needed, to indicate which QoS dimensions – which in a wider sense may be and are expressed by the weight-based methodology as *adaptation-strategies* – have higher desirability for the user.

The Volare User Preferences Model refers to the adaptation of the SR and of the middleware configuration and is generic enough to express user preferences on generic quantitative crosscutting concerns – common for all supported applications – such as: resource management, performance level desirability, cost of binding, disruption minimization or binding cost strategy.

The WBART integrates dynamically in the adaptation logic the user preferences through a *User Preferences Interface (UPI)* application operating as a virtual multi-sensor, informing the Context Monitoring Module of the middleware on the current user preferences.

The Volare middleware acknowledges the need for dynamically declared user preferences and allows the user to alter user preferences at runtime at two different dimensions of increasing complexity through the UPI:

a. The short-term ***User Preference Mode*** selection, even at runtime – since change of user preference triggers adaptation – for establishing the policy-based adaptation behavior corresponding to the selected Mode.

b. The ***User Choices*** for each application, denoting the global policy customization encoded in a **User Choices Profile (UCP)** for each application. Through the UPI the middleware provides the opportunity to the user, to modify the default values on pre-determined at policy design time parameters – within authorized percentage limits (usually between 80% – 125% of the default value).

6.4.1 The User Preference Modes

The User Preferences Interface (UPI) application allows the user to easily select the desired user preference among several mutually exclusive ones, representing real-life alternatives on crosscutting concerns, valid for every application. As a typical example for the current scenario, the following four user preference modes have been defined:

UP Mode: HighQuality or **Normal** or **LowCost** or **SaveBattery**

The adaptation behavior description of each user preference mode of the case study global policy is given below in Table 6-3, as provided by the generic model of user preference modes by the WBART methodology.

The user preference mode influences the adaptation behavior in two ways: Firstly, through the Weight Assigning Strategy, since the SWC table (see Table 6-2 above) provides different default SWC values at each user preference mode, indicating a change in the relative importance and consequently the contribution of each adaptation-strategy and by consequence of their selectable adaptation-rules to the adaptation results.

Table 6-3 – The Generic User Preference Modes

User Preference: Normal (N)

Guideline: The Normal mode is the default user preference mode and allows full use of the resources, with bitrate = 0.8*unadapted value. Set strategy-weight-coefficients at almost equal weight coefficients at the SWCT.

User Preference: LowCost (C)

Guideline: The Cost Optimization adaptation-strategy becomes dominant:

a) by a reduction of the cost preference value on binding price of the “Normal” value;

b) by increasing the relative importance of the strategy-weight-coefficient Scost at the SWCT, making it dominant.

User Preference Mode: SaveBattery (S)

Guideline: The Resource Optimization Adaptation-strategy becomes dominant, reducing power consumption in anticipation of extended mobile use before recharging by:

a) reducing the technically attainable bitrate QoS request of the “Normal” value;

b) by increasing the relative importance of the adaptation-strategy Sres through appropriate value at the SWCT and making it dominant.

User Preference: HighQuality (H)

Guideline: The Performance Optimization adaptation-strategy Sperf becomes dominant, ignoring eventual low resource, cost or disruption levels, like battery power level or higher than normal cost of binding:

a) by setting the QoS Request parameters values at the maximum attainable levels;

b) by setting the Sperf strategy-weight-coefficient dominant at the SWCT.

Secondly, as a context element in the Criteria predicates, since at each user preference mode there are typically adaptation-rules specific to it (if it is a relevant context for the adaptation-strategy) that represent the recommended adaptation behavior. Changing user *preference mode* through the UPI even at runtime establishes a different policy-based behavioral pattern, activating policy execution and adaptation influenced by the user preference specific adaptation-rules under the current context. Not all adaptation-rules depend on the user preference mode.

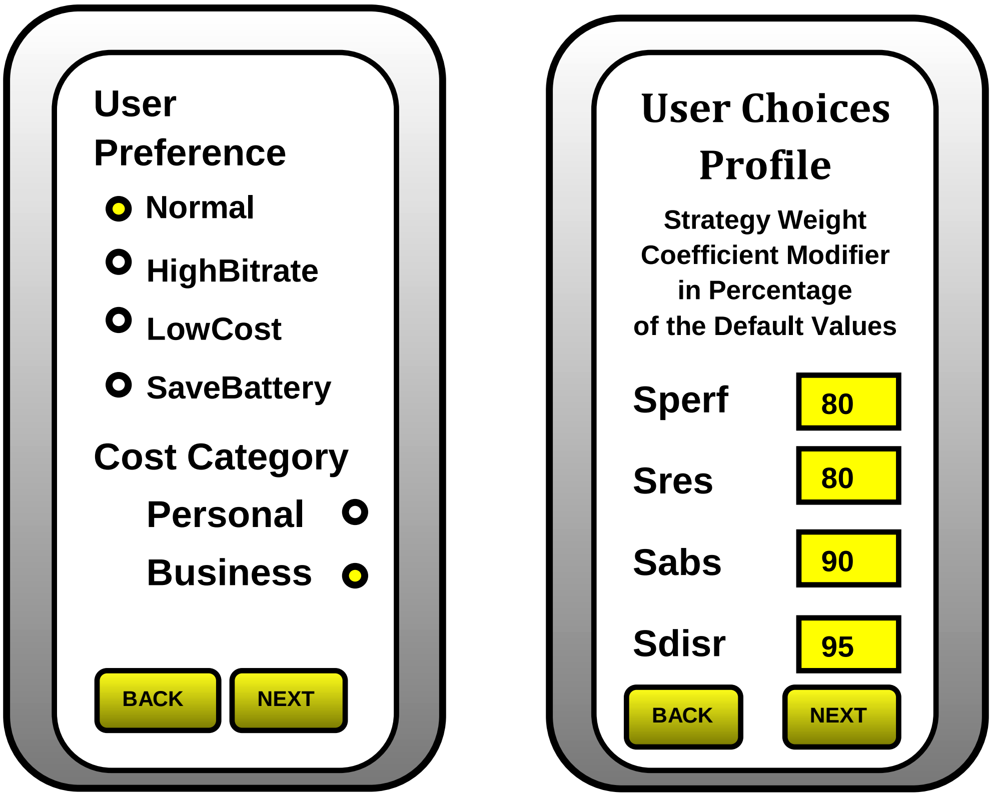
Remark: *In Volare the User Preferences Model follows a reserved approach on the frequency and degree of active user intervention, since the middleware never actively requests user intervention for decision-making in contrast to other work like [28][36][69].*

6.4.2 The Scenario User Choices Profile

In adaptation logic, the adaptation reasoning – when, how and how much to adapt – is defined by certain parameters, either c*omparison parameters* in the Criteria predicates – specifying when to adapt, or in the adaptation-rule adaptation actions as *action parameters*, specifying the measure of the adaptation action for each selected adaptation-rule.

In Volare the User Preferences Interface (UPI) is a configurable by the global policy developer multi-sensor application with a number of numeric and string variables, that to each one may be assigned a name, a default value and a variation range, so that the User may easily insert her preferences at will.

Figure 6-1 – Sample User Interface screen for Policy Customization



User Preference

● Normal

○ HighBitrate

○ LowCost

○ SaveBattery

Cost Category

Personal ○

Business ●

Figure 6-1a

User Choices Profile

Strategy Weight Coefficient Modifier in Percentage of the Default Values

Sperf 80

Sres 80

Sabs 90

Sdisr 95

Figure 6-1b

Through the UPI the middleware provides the opportunity to the user on pre-determined at policy design time numeric *comparison, action or weight coefficient* *parameters*, to modify the default values – within authorized percentage limits (usually between 80% – 125% of the default value).

The UPI dispatches at every monitoring *recheck period* the current sensor values set by the user to the Context Monitoring Module, corresponding to policy - declared UPI-based context-variables.

In the case study scenario example the user (see Fig. 6-1b) customizes the default values of the Strategy Weight Coefficient (SWC) of the adaptation-strategies at every user preference mode. In the above figures, the four strategies are Spref, Sres, Sabs and Sdist. The initial default values of the customizable parameters such as the SWC are set by the global policy developer at the User Preferences Interface configuration (or at the global policy as calculation-variables).

Based on the above, the User Choices Profile (UCP) should include default values and modifiers for at least the four following adaptation logic parameters:

a. The default Strategy Weight Coefficients for each user preference mode, with upper/lower bounds for fine-tuning by the user. The case study default SWC values are presented in Table 6-2 above.

b. The global policy LTPG goal values, since these values are user-specific. For instance, the user will need to insert her own monthly data volume allowance (4000 MBs/month) or the monthly credit allowance (5 pounds/month) for services on the cloud, as specified in the contract with the MNSP. Similarly, customizable constants related to LTPGs metrics, may also be configured.

c. The global policy developer configures the UPI setting the customizable parameters, like selected *comparison* or *action* parameter values defining the “High” or “Low” or “Very low” threshold level of a context-variable or metric at which the adaptation behavior should change. For instance: take no action for a LTPG on a consumable resource, as long its level is higher than 80% of the budget. On the contrary, take strict actions if the resource comes at very low level (say <20%).

In this way the user may create her customized User Choices Profile (UCP) for the adaptation behaviour of the middleware and the SR of each application. The customized UCP for each application is saved by the UPI and is automatically recalled – instead of the default UCP – at a service request of the relevant application to customize the UPI default settings and consequently provide the customized values to the adaptation logic parameters.

6.4.3 Optional Extensions of the User Preferences Model

It should be noted that the Volare configurable User Preferences Model is scenario-specific, configurable and may include at the same time multiple levels of mutually exclusive or concurrent user preference options, as follows:

UP Feature: Business or Personal (on different cost/performance requirements);

UP Functional Feature: Wi-Fi or GSM (on different functional requirements);

UP Disability Feature: Video + Audio or Audio only or Audio + low QoS Video (on different QoS requirements).

Thus, the current user preference mode may be a vector of preferences on more than one level of mutually exclusive variants. In these cases, the user selects the desired variant for each user preference feature. At policy execution, each selected context element influences which adaptation-rules will be matched to be further evaluated form selection and execution. For instance the “Business” feature may indicate a different binding cost strategy (company expenses vs. personal expenses) and different QoS settings, that will be matched under the context element User Preference: “Normal” and the option: “Business”. Each level of user preference mode features is declared as a string type context-variable. The UPI dispatches the current user preference vector of feature value(s) to the Context Monitoring Module dynamically.

6.5 Quantitative Long Term Performance Goals (LTPGs)

It is important to distinguish between two types of adaptation concerning the adaptation reasoning temporal horizon dimension:

a. **Short Term Performance Goals** are called in this work policy-based performance goals in consideration of only the context parameters current values or values of the current application session (in the scenario: session of service discovery and binding on cloud services). At a new session, adaptation reasoning restarts based on the current context without any sense of continuity, but simply on policy-based: *If current conditions … Then actions.*

b. **Long Term Performance Goals**, abbreviated as **LTPGs**, are called in this work *quantitative performance goals* on repeated activity cycles managed by the adaptation policy that span over a period of many application sessions, until a temporal pre-determined limit or otherwise until a predefined parameter limit has been exceeded.

This second adaptation approach is more challenging and more “intelligent”, demanding more on abstract-level policy logic. It may concern optimization over a consumable resource, like optimizing battery power within the battery-recharging schedule, or keeping binding cost on services on the cloud within a monthly renewable (or additive) credit allowance. In more general terms it may concern optimization or respect of constraints on a performance metric over a time period (week, month, etc.) or until an increasing or decreasing parameter value limit is reached.

6.5.1 Introducing Long Term Performance Goals in the Policy

A main feature of the Volare approach and the WBART methodology consists in creating the capability for introduction and use in the policy of quantitative LTPGs over finite horizons that supersede the duration of a cloud service discovery (CSD) session.

Adaptation policy with Long Term Performance Goals (LTPGs) depends on the current context not only within a session as in short term adaptation, but also insistently over the whole Long Term Performance Goal horizon time. Consequently, it requires capability from the Policy Specification Language to define metrics required for monitoring performance and guiding the adaptation based on usage data that span at least over one LTPG horizon context & adaptation history. Adaptation through LTPGs differs significantly from typical short-term condition-action adaptation based on the current context of the active CSD session, since they concern a full temporal trajectory (sequence of decisions on successive condition-action occurrences) and they are assessed for the degree of success or failure at the end of their finite horizon.

LTPGs are expressed through adaptation-rules of the global or the application policies or of both. In the case study both cases are supported, either of individual LTPGs in only one policy or LTPGs served by adaptation-rules across both policies.

6.5.2 Functional Requirements to Support LTPGs

The design and use of LPTGs in the adaptation policy requires support by the APSL and the mobile middleware, on the following non-trivial characteristics:

1. **Declaration capability for new unanticipated variables**

Declaration capability for new unanticipated calculation- or statistic-calculation-variables supported by the APSL is required, so that the developer may declare new metrics representing cumulative LTPG performance within the horizon duration so far, as expressions of declared context- or adaptation-variables or statistical inference on their usage data.

2. Middleware capability for context & adaptation history recording

Middleware capability for context & adaptation history maintenance, by recording at regular specified intervals the values of all declared (and consequently monitored or evaluated) context or adaptation or intermediate variables, in the device Context & Adaptation History Database (CAHiD) and maintaining it. The Volare middleware includes the Context & Adaptation History Module (CAHiM) that is charged with usage history database management, aggregation and maintenance (see Ch. 4 on the middleware implementation).

3. Middleware capability for statistic inference on declared parameter

Middleware capability for statistic inference on declared parameter stored on the usage data.

The Volare APSL provides notation for declaration of statistic-calculation-variables, supported by the middleware Statistical Analysis Module, in the form: **typeID CalcVar parameterID == parameterID.PeriodID.StatisticTermID**, on numeric declared parameters and the periods and statistic terms supported by the middleware, (see ch.5.4).

6.5.3 The LTPG Design Model

Every quantitative LTPG is considered as a consumable real or virtual resource that is gradually depleted but should preferably not be totally consumed till the end of each horizon.

Adaptation of an LTPG is considered as a behavioural adaptation (parameter adaptation) VP with different behavioural variants selectable under different context conditions to assist achieving the goal by the end of the horizon. LTPG performance monitoring has to be instituted, with the current context “augmented” to include LTPG-related monitoring metrics at any point in time within the LTPG horizon. The LTPG control strategy consists in defining LTPG resource availability levels, at each of which a different behavioural variant representing adaptation-rules on the control variables will be matched and selected specifying the appropriate adaptation through the following LTPG design model, at each of the two Policy Authoring stages: the Policy Design and the Policy Development stage.

1. Policy Design Stage

For an LTPG to be mapped through adaptation-rules in the policy, at the preparatory Policy Design Stage it is required by the developer to analyse, define and declare in the policy the following parameters:

a) Assign the LTPG to an Adaptation-Strategy

This task concerns the classification of the LTPG in the most related adaptation-concern & strategy, so that its adaptation-rules will have the associated strategy-weight-coefficient.

b) Define & Declare the LTPG related Parameters

LTPG related Parameters – The main parameters that typically characterize a quantitative LTPG need to be identified and declared, like:

horizonDuration: the horizon duration (fixed or variable), for instance the discharge period for battery LTPG, or the monthly period for the Credit or Data Volume LTPGs. This value is indirectly defined by selecting the pre-defined statistic "Period" that corresponds to the **horizonDuration**.

resourceAllowance: the max value of the LTPG resource that is not to be exceeded. Typically this parameter is either declared as a constant calculation-variable or as a user-customizable UPI context-variable.

resourceVLowLevel: threshold parameter representing the very low level of the LTPG resource at which emergency adaptation-rules are matched and selected for adaptation, in order to keep LTPG performance on track.

c) Define & Declare LTPG Monitoring Metrics

LTPG Monitoring Metrics – The LTPG performance over its horizon may be monitored by several metrics that are defined and declared as calculation or statistic-calculation-variables by the developer and are used in the policy for decision-making purposes. The weight-based methodology suggests the following six main metrics, measuring both the resource use and the time passed within the LTPG horizon:

a. The "durationTillNow" Metric

durationTillNow: the time duration from the beginning of the current horizon period till now, in appropriate units.

b. The "resourceUsed" Metric

resourceUsed: the resource used within the **durationTillNow** time duration from the beginning of the current horizon period till now, in appropriate units.

c. The "Resource Use Ratio" Metric

The **resourceUseratio** evaluates the considered real or virtual resource use from the beginning of the current horizon till now, vs. the whole horizon allowance or prediction (implied percentage type):

resourceUseratio = 100 x resource used / resourceAllowance (1)

d. The "LTPG Duration Ratio" Metric

The **durationRatio** is required to indicate the time interval so far within the known or estimated horizon duration (implied percentage type):

durationRatio = 100 x duration till now / horizonDuration (2)

The above two ratios represent "criticality" levels on LTPG performance for the achievement of an LTPG, in the sense that the higher their values are the more difficult it is to correct by adaptation any deviations from the goal sought. As "criticality" indicators may be used in the adaptation-rules, for instance in order to decide (in the predicates) or to specify (in the adaptation-statements) the magnitude of corrective adaptation actions.

e. The "Resource Reference Rate" Metric

The model defines as **resourceRefRate** the implied percentage type metric:

resourceRefRate =

100 x (resource used / duration till now) * (horizonDuration / resourceAllowance) (3)

2. Policy Development (Adaptation-Rules Authoring) Stage

Once the LTPG related parameters and monitoring metrics have been defined in the previous Policy Design Stage, the LTPG control strategy lies on determining the following elements:

- Define the LTPG sufficient condition(s), that if it is respected then the LTPG is achieved by the end of the horizon time
- Identify control adaptation-variables
- Define LTPG resource availability levels
- Define a behavioral variant at each availability level (typically at the "Low" or "VLow" level), in the form of a Criteria-Subpolicy pair with adaptation-rule(s) on the control variable(s), imposing the required adaptation.

Consequently at the Policy Development Stage (adaptation-rules authoring), the following procedural successive actions need to be followed in order to successfully map an LTPG in adaptation-rules:

a) Define the LTPG Sufficient Condition

One or more quantitative constraints have to be defined that should represent a sufficient condition of “correct operation” of the system over the remaining finite horizon time interval concerning the LTPG, which if respected by the end of the LTPG horizon, then the goal will be achieved.

The LTPG Sufficient Condition Constraint – For an unknown usage pattern, the LTPG design model adopts the following “naïve” constraint:

resource used / time till now <= resourceAllowance / horizonDuration (4)

⇔

(resource used / time till now) / (resourceAllowance / horizonDuration) (4a), which is an empirical sufficient condition when lacking usage pattern information for achieving the LTPG by the end of the horizon. The constraint (4a), substituting for monitoring metrics (3), becomes:

resourceRefRate <= 100 (5)

Remark: *Note that constraint (4a) or (5) is cumulative over the LTPG horizon duration, in the sense that even if for a time interval within the horizon duration the metric* ***resourceRefRate*** *exceeds the limit, if later the resource use rate is reduced – for instance by appropriate adaptation actions – then it may again cumulatively satisfy by the end of the horizon duration the upper limit constraint and the LTPG will be achieved.*

b) Define LTPG Resource Availability Levels

Based on the above metrics, parameters and the sufficient condition (5), define different LTPG resource availability levels – by comparing the current monitoring metrics values against threshold parameter values – with each level requiring different adaptation actions. Practically, instituting up to 3 levels of availability levels for behavioural adaptation for every quantitative LTPG may often be considered satisfactory – in the sense that it usually ensures both goal fulfilment and satisfactory adaptation behaviour, in the following manner:

- At High resource availability – with *resourceUseratio* and *resourceRefRate* within acceptable limits – no restrictions need to be imposed on the control variable(s)

- At Low resource availability level – when monitoring metrics like: *resourceUseratio* or the consumption rate *resourceRefRate* exceed set limits – gradual restrictions are imposed in proportion to the discrepancy of the current performance metric from the set level value
- At Very Low availability level – indicated by reaching *resourceUseratio* =< *resourceEmergency* level – more abrupt adaptation actions are to be specified, that may need to lower substantially the quality of the requested cloud service in order to achieve the goal by the end of the horizon.

This step constitutes the design of a LTPG resource availability level assessment algorithm, classifying the current instance in one of the selected levels.

In §7.4, a segment of the case study global policy file presents a generic LTPG availability levels classification algorithm, introduced through appropriate adaptation-rules at the 2nd CEC of the global policy file rules section.

c) Identify the LTPG Control Variable(s)

Control Variable(s) – Control variables are adaptation-variables, either middleware configuration-variables or QoS-variables, whose adaptation may influence the resource consumption of the resource considered. For LTPGs set in the global policy, it may be that the global QoS-variable(s) may not be effective at managing the LTPG. In this case adaptation-rules in every application policy should be specified for fulfilling the LTPG at the end of its horizon.

For instance, for the Monthly Credit Allowance Management LTPG, the costQoSReq global QoS-variable may be an appropriate control variable. For battery or data volume management the bitrate QoS variable of an application policy may be a control variable.

d) Define the Behavioural Adaptation Variants for the LTPG

In the weight-based methodology every quantitative LTPG is considered as a behavioural VP with several mutually exclusive behavioural variants, each expressing a collection of adaptation-rules matched and selected when its common predicates are satisfied, for influencing the required adaptation to keep the LTPG on track.

A simple and practical procedure adopted by PAM for LTPG control consists in classifying the availability of the real or virtual resource representing the LTPG, in typically three (or eventually more than three if more fine-grained analysis is required) **resource availability** classes as “**High”, “Low”** or “**Very Low”** (abbreviated as **VLow**), depending on predicates established through the monitoring metrics. At each LTPG resource availability level established through the related adaptation-rules in the 2nd CEC, the corresponding behavioral variant of the LTPG VP includes the adaptation-rules that specify the required adaptation. This takes place at the 3rd CEC, with a BV corresponding to each resource availability level.

e) Prefixed vs. Usage-based Adaptation on LTPGs

Two adaptation viewpoints are distinguished in this work on the support of policy-based quantitative Long Term Performance Goals:

1. LTPG Control by Prefixed Metrics & Threshold Parameters

At the LTPG management by prefixed metrics & threshold parameters, performance metrics and decision-making for each LTPG are influenced by the context through prefixed common sense but empirical estimates on the usage evolution within the time horizon, like the date-based criterion. The date-based criterion, expresses the prior belief that the activity will continue using the specified resource or budget till the end of the LTPG current period, proportionally to the number of days still to come, i.e.:

> *Date-based criterion = (days till now / expected days in the period) x (budget used / total budget) (1).*

It is a common-sense measure for decision-making on LTPG control without any usage data and the case study makes systematic use of it.

2. LTPG Control by Usage-based Metrics & Threshold Parameters

At the usage-based adaptation on the contrary, statistical analysis of the usage model may provide insights in the future system behaviour and permit a better estimation on the course of action to achieve the LTPGs, thus allowing smoother and less abrupt adaptation (and resultant operation) to achieve the LTPG.

LTPG performance through consideration of the device usage model may improve the policy performance assessment on goal achievement without undue constraining other uses. However, it requires specifying at the policy level statistical inference parameters based on the usage model and sufficiency of historic data.

> *The usage-based criterion expresses the prediction estimate for the rest of the current LTPG period as function of the usage model statistics.*

Since typically there is one user for each mobile device, with personal habits, favourite activities on the device use etc., the usage model may well provide useful information in optimizing an initial arbitrary estimation for an LTPG.

6.5.4 Usage-Based Policy Self-Optimization

In the previous paragraphs an LTPG design model of typically three resource availability levels (“High”, “Low”, “VLow”) is described, defined by LTPG metrics taking values within predetermined sub-domains, that presents simplicity in establishing the LTPG adaptation and practicality in achieving the goal. Typically the threshold value **resourceVLowLevel** (for instance 20% of the resourve level) is used referring to the **resourceRatio,** which in combination with the other defined LTPG monitoring metrics assess the current resource availability level. Then at the current availability level the appropriate behavioral variant is activated imposing restrictive adaptation actions.

The collection of adaptation-rules (Criteria-Subpolicy pairs) in the adaptation logic that constitute adaptation reasoning on pre-determined threshold parameter values is called in this work: *control-layer*. However, managing a LTPG over its horizon with unknown usage pattern and stochastic variation through a prefixed “control layer” of adaptation-rules, may not be effective, especially since it refers to a sequence of adaptation actions on successive CSD sessions.

This is the reason that a “supervisory layer” of adaptation-rules is required, so that in case that the LTPG is not achieved, appropriate corrective actions are taken on the “control layer” to improve its effectiveness, making it stricter or more relaxed, depending on the LTPG results.

This way, the adaptation logic may be enriched with capability to automatically evaluate its past behavior on LTPG horizons scale and verify if each goal has been fulfilled or missed and even evaluate how effectively this was done and take corrective actions at the beginning of a new LTPG horizon.

The collection of adaptation-rules that selects alternative algorithms or adapts threshold parameters on existing algorithms on the basis of horizon scale results is called in this work: *supervisory-layer* (or *change management layer*). It serves to establish automated self-optimization of the *control-layer* adaptation-rules in the adaptation logic by modifying threshold parameters, through policy-based *supervisory-layer* adaptation-rules designed to reason and act on a slower time scale than the *control-layer*, on horizon time scale usage data.

The Volare approach, establishing usage data recording and statistical analysis-support by the middleware, allows the developer to introduce in the adaptation logic this *supervisory-layer* of policy self-optimization.

In the case study this self-optimization of the adaptation logic is demonstrated by *supervisory-level* adaptation-rules that are matched and selected at any policy execution after the end of an LTPG horizon, verifying if the associated LTPG has been achieved or not. Then, on the basis of this evaluation, different algorithms and threshold parameters values are introduced in the *control-layer* adaptation-rules for the new time horizon. As this *supervisory-layer* evaluation may concern algorithmic alternatives, as different or modified algorithms are activated, the relevant adaptation-rules are assigned to the first CEC.

A simple application of *supervisory-layer* adaptation-rules in the case study global policy consists in monitoring success or failure of the LTPG at the end of the last horizon and adapting the *resourceVlowLevel* (initially set at 10% - at which "VLow" level restrictions are imposed at the control variable(s)), thus enlarging the "Low" and "VLow" availability levels range and reducing the "High" level range, enabling restrictive behavioral variants to control adaptation. Additionally, the *actionCoeff* parameter, influencing the action value for the control variable(s) may also be adapted accordingly. Of course this is only a simple demonstration of the capability offered by the Volare approach to the policy developers, as applied in the case study in chapter 7.

6.5.5 Limitations in the Use of LTPGs in the Adaptation Policy

Two main limitations are set in the current Volare version for introduction of quantitative LTPGs in the policy. Firstly, in the current Volare version, the finite horizon of a LTPG should coincide with one of the time periods supported by Statistical Analysis Module of the middleware. Secondly, all LTPG monitoring and decision-support metrics that are based on statistical inference should be in the form of a statistic term from the ones supported by the middleware.

If these two requirements are satisfied, the APSL and the middleware support the declaration and calculation/retrieval of unanticipated at middleware design time statistic-calculation-variables or of declared calculation-variables referencing them for metrics monitoring LTPG performance during each episode.

6.6 Policy Testing & Verification

The methodology developed concerning the weight-based adaptation reasoning technique has designed a detailed and extensive testing, verification & evaluation process for the adaptation logic, described in detail in Appendix C. It makes use of a specially designed simulated policy execution & verification application named PEVApp for offline automated generation and execution of test suites and the verification and evaluation of the simulated dynamic results.

As part of its functionality, the middleware keeps record of the context & adaptation data at each monitoring row in the Context & Adaptation History Database (CAHiD). The CAHiD data may be extracted and used independently of the middleware for verification, evaluation or validation purposes.

In the following paragraphs the dynamic testing and verification techniques for the middleware and the adaptation policy are outlined, as well as the strategy for test suites generation on each technique and the adequacy criteria. Based on the context data & results derived by the offline simulated dynamic testing, a rules fault detection algorithms have been designed for detecting policy logic "irregularities" that do not stop the program flow but constitute rule faults.

The phases for the adaptation logic (policy) testing, verification & eventual evaluation are described in the following paragraphs.

1. Automated Policy Syntactic Correctness Verification

Every policy is first verified on syntactic correctness through an initial policy syntax evaluation by the relevant Syntactic Correctness Verification Tool, verifying basic data-flow testing prerequisites, like: no variable used but undeclared, no variable declared but unused, variable declared twice, Criteria non-corresponding to Subpolicies, etc. Such errors detection, including typing errors, helps avoid a lot of troubleshooting when editing a new or updating an existing policy.

2. Automated Offline Test Suite Generation & Execution

Test suites on the developed policy are automatically derived on developer-selected options, based on the predicate coverage strategy [71][73]. These test suites may be automatically enriched, based on the domain testing strategy [72] concerning the context sub-domain boundary values, where may be higher probability for rule faults, with tentative adequacy criterion 100%.

Automated repeated execution of the test suites is implemented offline by the Policy Editing & Verification Assistant tool. Test cases with failures are recorded for debugging, while the dynamic data are stored for evaluation.

3. Automated Analysis of Results by Fault Detection Algorithms

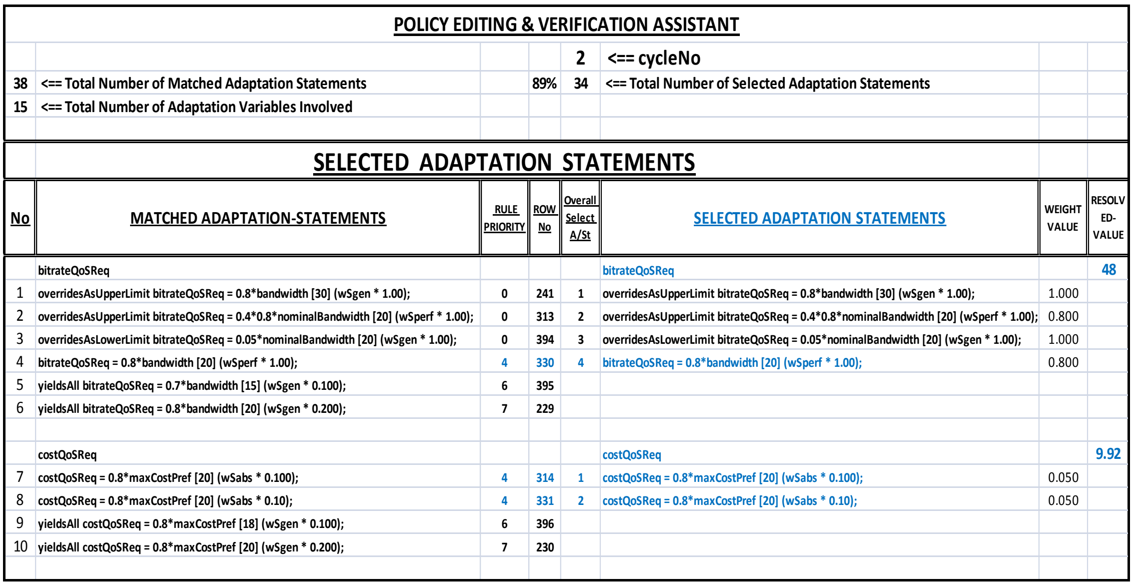
The real extracted usage data from the mobile or the test suite execution data are evaluated for rule faults detection that are not evident as test case failures. Sama et al [55] have published an approach for static verification of the adaptation logic of CAAAs. However, the foundation of the approach in [55] is based on the consistency algorithm, which is not relevant in Volare since multiple adaptation-rules on one head predicate may frequently be selected in the Volare APSL-compatible policies.

Volare shares the basic thinking with [55] on rules fault detection and has developed rule fault patterns identification and detection algorithms, adjusted to the Volare APSL compatible adaptation policy for detecting faults and anomalies, based not on static analysis as in [55], but on the analysis of the context & adaptation real or simulated execution data through Policy Editing & Verification Assistant tool. The faults detected are recorded in a separate output sheet, so that the developer/tester may evaluate them and take corrective actions.

4. Policy Editing & Performance Evaluation

PEVApp provides specific tools like the Policy Editing & Verification Assistant, assisting the policy developer at evaluating policy performance through automated charts on simulated policy execution dynamic results on test suites. Additionally, it assists the developer at analysing at each context instance the adaptation-rules matched & selected and the weight-based driven adaptation choices. The provided semi-automated tables and charts assist the developer in policy authoring.

Table 6-4 – A view of the Policy Editing & Verification Assistant Tool



POLICY EDITING & VERIFICATION ASSISTANT

				2	<== cycleNo		
38	<== Total Number of Matched Adaptation Statements		89%	34	<== Total Number of Selected Adaptation Statements		
15	<== Total Number of Adaptation Variables Involved						

SELECTED ADAPTATION STATEMENTS

No	MATCHED ADAPTATION-STATEMENTS	RULE PRIORITY	ROW No	Overall Select A/St	SELECTED ADAPTATION STATEMENTS	WEIGHT VALUE	RESOLVED-VALUE
	bitrateQoSReq				bitrateQoSReq		48
1	overridesAsUpperLimit bitrateQoSReq = 0.8*bandwidth [30] (wSgen * 1.00);	0	241	1	overridesAsUpperLimit bitrateQoSReq = 0.8*bandwidth [30] (wSgen * 1.00);	1.000	
2	overridesAsUpperLimit bitrateQoSReq = 0.4*0.8*nominalBandwidth [20] (wSperf * 1.00);	0	313	2	overridesAsUpperLimit bitrateQoSReq = 0.4*0.8*nominalBandwidth [20] (wSperf * 1.00);	0.800	
3	overridesAsLowerLimit bitrateQoSReq = 0.05*nominalBandwidth [20] (wSgen * 1.00);	0	394	3	overridesAsLowerLimit bitrateQoSReq = 0.05*nominalBandwidth [20] (wSgen * 1.00);	1.000	
4	bitrateQoSReq = 0.8*bandwidth [20] (wSperf * 1.00);	4	330	4	bitrateQoSReq = 0.8*bandwidth [20] (wSperf * 1.00);	0.800	
5	yieldsAll bitrateQoSReq = 0.7*bandwidth [15] (wSgen * 0.100);	6	395				
6	yieldsAll bitrateQoSReq = 0.8*bandwidth [20] (wSgen * 0.200);	7	229				
	costQoSReq				costQoSReq		9.92
7	costQoSReq = 0.8*maxCostPref [20] (wSabs * 0.100);	4	314	1	costQoSReq = 0.8*maxCostPref [20] (wSabs * 0.100);	0.050	
8	costQoSReq = 0.8*maxCostPref [20] (wSabs * 0.10);	4	331	2	costQoSReq = 0.8*maxCostPref [20] (wSabs * 0.10);	0.050	
9	yieldsAll costQoSReq = 0.8*maxCostPref [18] (wSgen * 0.100);	6	396				
10	yieldsAll costQoSReq = 0.8*maxCostPref [20] (wSgen * 0.200);	7	230				

5. Automated Metamorphic Relations Verification

In this work use is made of **Metamorphic Relations (MR),** that are expected relations not in the input/output data of a test case but between sets of input and output data of the test suite results [70].

Table 6-5 – Automated Composite Policy Verification Tool

AUTOMATED DYNAMIC TESTING & FAULT DETECTION VERIFICATION			
CUMULATED RESULTS EVALUATION			15/01/2013
ON COMPOSITE POLICY:	G1A1 - APP1		
	Test Cases		
Adequacy Criteria	Evaluated	Total	%
Predicate Coverage	2800	2900	96.6
Domain Testing Boundary Values Coverage	400	420	95.2
I. Dynamic Testing Failures Assessment			
Dynamic Testing Failures Assessment	2920	2920	100.0
II. Dynamic Testing Fault Detection	Failed	Evaluated	%
Dead Criteria	0	26	0.0%
Dead Conditions	3	72	4.2%
Dead Rules	3	121	2.5%
Rules Unmatched	3	121	2.5%
Dead Adaptation Actions (Non-numeric)	1	4	25.0
Rules with Dead Adaptation Action	4	121	3.3%
Unreachable States	4	121	3.3
III. Metamorphic Relations Verification	Failed	Evaluated	%
Criterion: HighQuality >= Normal >= (LowCost, SaveBattery)			
bitrateQoSReq	0	2920	0.0%
costQoSReQ	0	2920	0.0%

The MR idea is implemented for checking appropriately sorted context & adaptation data derived from the simulated dynamic testing on their conformity to predetermined scenario-specific MR. If violation of the MR is identified then a behavioural fault is detected. Such Metamorphic Relations have been identified in this work and are used for policy verification, especially w.r.t. the user preference context element, where at the same or similar other context data, some adaptation-variables like *bitrateQoSReq* and *costQoSReq* should be decreasing when moving from “HighQuality” user preference to “Normal” and then to “LowCost” or “SaveBattery”.

6. Simulation-based Policy Evaluation on LTPGs

An important use of the simulated policy execution PEVApp application consists in designing test suites or in using real usage data offline, and evaluate policy performance at achieving the LTPGs by the end of their horizons, by modifying policy parameters. Through the automated chart-making on the offline simulated dynamic results, the developer may have an evaluation of the policy performance before installing it on the device.

In Table 6-4 is depicted the Overall Testing & Verification Results Table on the real global policy, detailing the rule fault patterns and detected errors, as provided by the Automated Testing & Verification Tool on the test suites generated by the PEVApp application.

6.7 The Weight-Based Policy Authoring Methodology

Based on the above mentioned innovative features, the weight-based adaptation reasoning technique has developed the ***Policy Authoring Methodology (PAM)***, to assist the global or application-policy developers to design, develop, test & verify the adaptation policy for implementing middleware-based dynamic context-aware adaptation on the active SR for cloud service discovery by a mobile application.

The three Policy Development Stages

Based on the conceptual viewpoints described in §6.1 to §6.6, policy design & implementation in the weight-base technique proceeds through the following three conceptual successive stages:

a. Policy Design Stage (Stage 1)

Analyze System – Identify Requirements & Goals – Declare Variables

b. Policy Rules Development Stage (Stage 2)

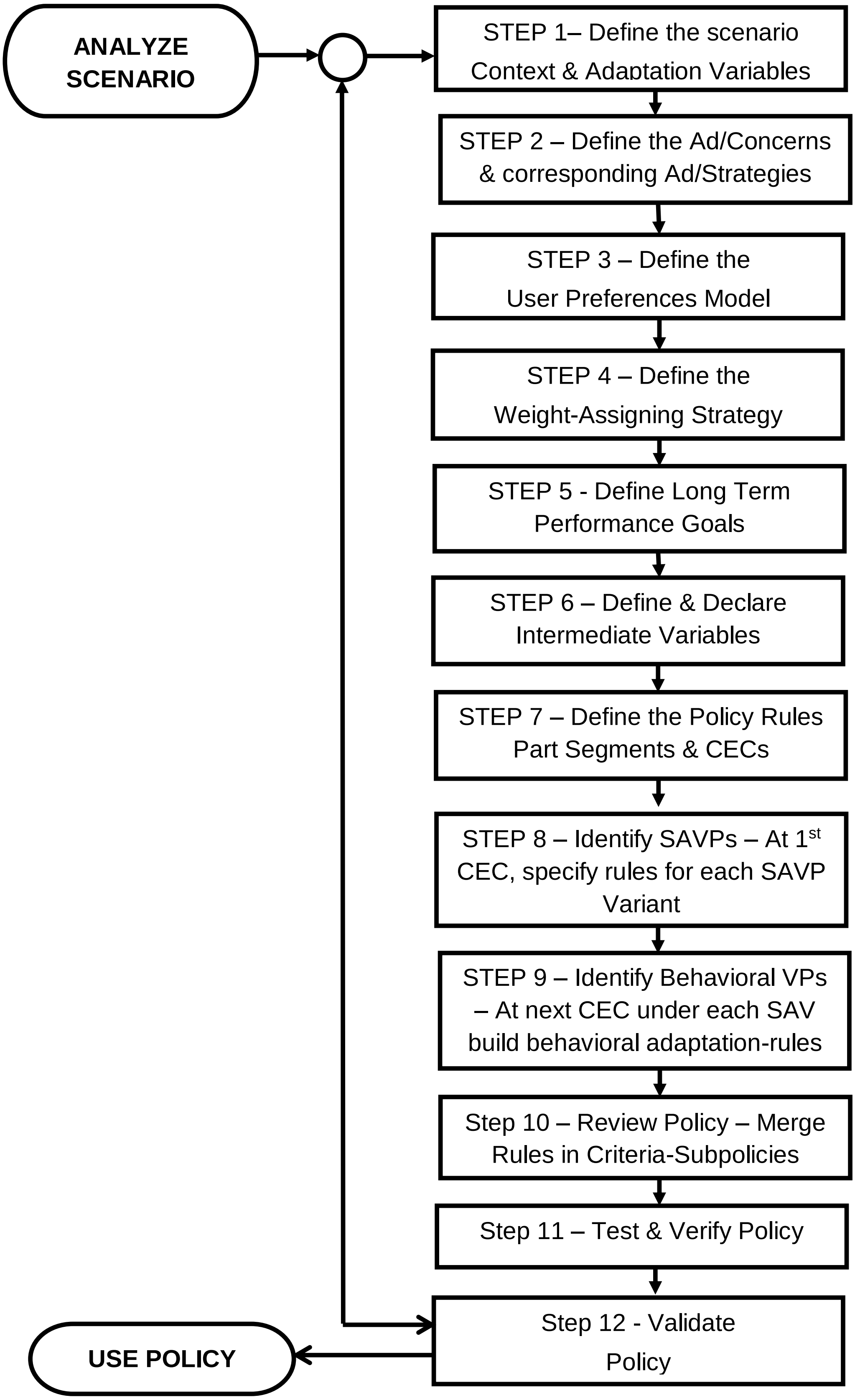
Specify the Adaptation-Rules for Structural/Algorithmic and for Behavioral Adaptation

c. Policy Testing & Verification Stage (Stage 3)

Test, Verify & Validate the Policy.

Each of the three stages includes successive steps that need to be followed – typically in an iterative manner, as new elements derived at subsequent steps may impose re-evaluation of decisions made at a previous step – in order to finally develop a global or application-level adaptation policy for middleware-driven SR DCAA in the weight-based technique. The steps are shown synoptically in the following Figure 6-2.

Figure 6-2 – POLICY DEVELOPMENT FLOW DIAGRAM


ANALYZE SCENARIO
STEP 1– Define the scenario Context & Adaptation Variables
STEP 2 – Define the Ad/Concerns & corresponding Ad/Strategies
STEP 3 – Define the User Preferences Model
STEP 4 – Define the Weight-Assigning Strategy
STEP 5 - Define Long Term Performance Goals
STEP 6 – Define & Declare Intermediate Variables
STEP 7 – Define the Policy Rules Part Segments & CECs
STEP 8 – Identify SAVPs – At 1st CEC, specify rules for each SAVP Variant
STEP 9 – Identify Behavioral VPs – At next CEC under each SAV build behavioral adaptation-rules
Step 10 – Review Policy – Merge Rules in Criteria-Subpolicies
Step 11 – Test & Verify Policy
Step 12 - Validate Policy
USE POLICY

In chapter 7 at the case study implementation, this procedure will be analytically applied step by step.

6.7.1 Stage 1 – Policy Design Stage

Analyze System – Identify Requirements & Goals – Declare Variables

The Policy Design Stage represents the adaptive system analysis and policy authoring preparatory activity, where requirements are identified, goals are set concerning various viewpoints of the adaptation process, and the Variables Declaration policy section is specified.

Step 1. Define the Context and Adaptation Variables

Context & Adaptation Variables in this work is called the set of policy file context- and adaptation-variables and their data structure. These variables are predetermined for the middleware or for each SR and are specified by declaration statements at the Declarations section of the policy file.

The context-variables, defined at middleware design time, represent: (i) the middleware-supported monitoring context sensors and (ii) the UPI user-customizable user preference context-variables as in Step 5 below is analytically described.

The adaptation-variables are distinguished in:

a. the middleware configuration-variables, that specify adaptation of the middleware components as described in §4.3 and are predetermined at middleware design;
b. the QoS-variables of the active application SR, which are application-specific, except of the global QoS-variables that are common for all applications and are defined in the global policy (in the case study: *costQoSReq*).

The generic Context & Adaptation Profile for the case study global policy is described in §7.3.

Step 2. Define the Adaptation-concerns & corresponding Strategies

In the weight-based technique the scenario adaptation space is distinguished conceptually in several major ***adaptation-concerns***. The global policy developer defines the adaptation-concerns of interest to the scenario as major conceptual targeted adaptation viewpoints for designing the adaptation behavior.

The generic Adaptation-concerns & Adaptation Strategies Model described in §6.1 with the most generic concerns of common use may be adopted, or it may be modified, if adding new adaptation-concerns pays in finer-grained adaptation, since this action would increase the number of strategy weight coefficients.

Step 3. Specify the Weight Assigning Strategy

The weight-based methodology provides the recommended Weight Assigning Strategy (WAS) described in §6.3, through the weight function value through the default format: **weight function = wSstrat * uSstrat * wcRule** (or a developer-modified one), specifying the factors default values:

a. The default ***wSstrat*** *strategy weight coefficients* (SWCs) for each user preference
b. The default ***uSstrat*** *strategy-weight-coefficient-modifiers* set to 1 – allowing for future user customization of the SWCs
c. The rule weight coefficient **wcRule** depending on the specific role of each rule in the adaptation-strategy.

According to the recommended WAS procedure, when configuring the User Preferences Interface to the scenario the SWCs are specified for each adaptation-strategy and under each user preference mode.

The rule-weight-coefficient **wcRule** for each adaptation-rule will be specified by the developer at a later stage when all adaptation-rules are finalized.

Step 4. Configure the User Preferences Model through the UPI

As analytically described in §6.4, the weight-based methodology provides a configurable User Preferences Model on crosscutting concerns, that is initially configured by the global policy developer at the two following levels:

a) Define Alternative User Preference Modes

At policy development time, the global policy developer is expected to define a qualitative “*generic adaptation behavior description*” for each user preference mode and to design the Performance adaptation-strategy (Sperf) with adaptation-rules specifying a different adaptation behavior under each user preference mode. Each of the user preference modes is chosen to simulate a real-life situational approach that a typical user may encounter. For example in the generic Volare scenario: the default **Normal**, the **LowCost, HighQuality** and **SaveBattery** mode, each is indicative of the corresponding adaptation behavior.

b) Configuring the User Choices Profile (UCP)

The UPI application provides some default context-variables, basically numeric ones, communicating the default or user-customized values to the middleware. Through the UPI the developer configures the User Choices Profile (UCP) by assigning to each UPI sensor an indicative short title for identification, a default value and optionally for numeric variables upper and lower customization limits for user fine-tuning (for instance: 0.80 to 1.25 of the default value).

These user-customizable UPI context-variables, are initially determined by the global policy developer at UPI configuration with default values, and may be:

a. The Strategy Weight Coefficients that modify the relative importance of the rules of every adaptation-strategy under each user preference mode.
b. The goal/constraint parameter value(s) for each LTPG of the global policy.
c. Parameter values influencing adaptation, either as comparison threshold values in Criteria conditions or as action threshold values in adaptation-statements.

The default UCP may later be customized by the user, even for each specific application, and is automatically stored. Once an active application launches SR on the cloud, the middleware will load the associated UCP and extract and introduce in the policy the customized values through the UPI context-variables.

Step 5. Define Long Term Performance Goals

A main feature of the weight-based methodology is the introduction and use in the policy of Long Term Performance Goals over finite horizons that supersede the duration of a CSD session. The LTPG design model actions, as it is analytically described in §6.5, at this stage consist only in the following procedural actions:

a) Define & Assign the LTPG to an Adaptation-Strategy

Define and assign the LTPG in the most related adaptation-concern & strategy, so that all its adaptation-rules will have the associated strategy-weight-coefficient.

b) Define the LTPG Monitoring & Control Strategy

Define how LTPG performance within the horizon duration is to be monitored, mutually exclusive resource availability levels, control-variables and the adaptation actions required at each resource availability level.

Step 6. Declare the Policy Intermediate Variables

Declaring the intermediate variables is an iterative task of the Policy Design Stage for the policy developer, cyclically around Steps 1 to 6, as the policy design proceeds. The need for these intermediate variables is scenario-specific and it is the natural result of the analysis and design work previously described in Steps 2 to 5:

a) Declare the non-LTPG-related Parameters

The Volare APSL – as described in chapter 5 – in addition to the context- and adaptation-variables that are pre-determined by the middleware and the SR, supports the declaration and use of intermediate variables, namely (i) calculation-variables (reprenting constants, metrics and statistical inference parameters through statistic-calculation-variables on the usage data), as well as the policy-driven auxiliary-variables facilitating decision-making at policy development.

b) Declare the LTPG-related Parameters

Especially for each LTPG, it will be required to declare statistic-calculation-variables to establish the metrics required for monitoring performance, decision-making and controlling adaptation for achieving the LTPGs within their temporal horizons. As in §6.5.3, three collections of variables need to be declared:

a. Declare the LTPG-related parameters that characterize it, as calculation-variables: *resourceAllowance, resourceVLowLevel*. *horizonDuration* is indirectly defined by selecting the corresponding "period" for the statistic-calculation-variables, like: "Monthly", "Daily", etc.

b. Declare the LTPG-related monitoring metrics for LTPG performance assessment and decision-making metrics concerning goal achievement within the horizon duration, used to guide the LTPG-related adaptation. The typical five monitoring metrics for an LTPG with a single quantitative constraint are, as analytically described in §6.5.3: *durationTillNow, ltpgDurationRatio, resourceUsed, resourceUseratio, resourceRefRate*.

c. Declare the LTPG-related auxiliary-variables, like the LTPG resource availability levels, or other LTPG-related auxiliary-variables representing alternative algorithms or metrics.

6.7.2 Stage 2 – Policy Rules Development Stage

Stage 2 refers to the adaptation-rules development stage, after requirements are identified, goals are set and variables are declared in Stage 1. It concerns the development of adaptation-rules through the Steps 7, 8, 9 & 10 described below.

Step 7. Define Consecutive Execution Cycle (CEC) Policy File Segments

According to the Policy Authoring Methodology of the weight-based technique, the adaptation-rules section of a policy file is typically distinguished in segments, each containing the adaptation-rules to be evaluated for matching and execution at a specific consecutive execution cycle (CEC). It is possible, typically for application policy files, to have only one CEC segment (i.e. all adaptation-rules are to be evaluated for matching and selection at one CEC).

Recommended Structure of a policy file Adaptation-Rules Section

The PAM recommended structure of the adaptation-rules section of a policy file, suggests three CEC segments.

a) 1st CEC Segment of the Policy File

The 1st CEC segment of the global policy file includes Criteria-Subpolicies composed of:

a. Adaptation-Rules on Policy Engine Configuration-Variables

The adaptation-rules that specify values concerning the Policy Engine configuration-variables, which are resolved at the default 1st CEC and are equired for further policy execution, like: *cyclesMax* (denoting the number of CECs to be executed) and *defaultBindingmargin* (denoting the default binding margin for every QoS-variable without a specifiedbinfing margin value).

b. Adaptation-Rules on Structural or Algorithmic Variant Selection

The adaptation-rules that specify variant selection on structural (or functional) VPs (for instance specifying the On or Off state of active-on-demand sensors or functionalities – like Wi-Fi On or GPS On), or on algorithmic VPs on mutually exclusive algorithm selection or algorithm modification (through alternative metrics and threshold parameters values to be used by the algorithms at policy execution).

b) 2nd CEC Segment of the Policy File

The 2nd CEC segment of the policy file typically includes Criteria-Subpolicy pairs composed of the adaptation-rules that assess LTPGs performance (usually as “High” or “Low” or “VLow” LTPG resource availability level) through auxiliary (policy-driven) variables. In this way, the assessment for each LTPG may be used at the adaptation-rules of the next CEC as predicate for both the global and application policy adaptation-rules, specifying the required adaptation-actions at each availability level. In fact, the 2nd CEC policy file segment includes the adaptation-rules of the LTPG resource availability level assessment algorithm for each LTPG of the global or the application policy.

c) 3rd CEC Segment of the Policy File

The 3rd CEC policy file segment includes the Criteria and corresponding Subpolicies containing the bulk of adaptation-rules specifying values to the remaining middleware configuration variables as well as the QoS-variables representing parameter adaptation for the “control-layer” of the adaptation logic. It concerns adaptation-rules for both “long” and “short-term” adaptation goals.

Remark: *This structure of the adaptation-rules section of a policy file in typically 3 CEC segments, recommended especially for the global policy, assists the application policy developers to more easily understand the global policy structure and easily identify the LTPGs performance assessment levels and use them directly in their adaptation-rules. Usually an application policy only needs to have adaptation-rules specified for the 3rd CEC, except if there are dependencies – in which case more CECs may be required or if there are application-specific structural or algorithmic variants for the 1st CEC or application-specific LTPGs with resource availability assessment at the 2nd CEC.*

Step 8. Specify 1st CEC Rules on Structural/Algorithmic Adaptation

The policy developer needs to identify eventual Structural or Algorithmic Variation Points (SAVPs), whose valid combinations of variants constitutes alternative Mutually Exclusive Configuration Variants (MESCs). Since the middleware manages the policy-based adaptation of the middleware components and of the active SR on the cloud, alternative structural or algorithmic or configuration variants specified in the global policy may concern only the following issues:

a. Adaptation-rules concerning configuration-variables that configure the middleware Policy Engine parameters required for policy execution, like: *cyclesMax* or *defaultBindingmargin.*

b. Adaptation-rules specifying selection of eventual alternative structural variants – predetermined at middleware design time – determining the state for active-on-demand sensors or functionalities or algorithms. Additionaly, Adaptation-rules specifying selection of eventual alternative algorithmic variants, through alternative metrics concerning conditions or actions influencing the adaptation.

All adaptation-rules associated with the above objectives, as discussed in Step 7, need to be included at the 1st CEC to ensure consistency according to the recommended directives in §6.1.4 (or at the first CECs if there are dependencies among structural or algorithmic variants to be considered). In this way, the resulting values of the associated configuration- or QoS- or auxiliary-variables may be used as predicates in the rules of subsequent CECs.

The successive procedural actions to be followed are:

a) Develop the Adaptation-Rules on Policy Engine Configuration

Develop the adaptation-rules concerning configuration of the Policy Engine middleware configuration-variables required for policy execution, like: *cyclesMax* and *defaultBindingmargin*.

b) Develop the Adaptation-Rules on Structural or Algorithmic VPs

a. Identify the Structural or Algorithmic Variation Points (SAVPs)

The policy developer needs to identify and analyze the structural or algorithmic Variation Points (SAVPs) concerning the issue a. or b. above. Every SAVP is attached to the most relevant adaptation-concern, so that adaptation-rules that specify adaptation on it will belong to the respective adaptation-strategy (and will have the associated SWCs).

b. Identify for each SAVP the structural or algorithmic Variants (SAVs)

The developer identifies for each SAVP the mutually exclusive structural or algorithmic variants (SAVs) and defines through predicates their context sub-domains. Such SAVs may be active-on-demand sensors or functionalities, like: Wi-Fi On vs. Wi-Fi Off, GPS On vs. GPS Off, etc. They may be alternative algorithms or metrics, like: usage-based vs. prefixed metrics, each with alternative variants of algorithmic character. The valid combinations in the examined scenario of these mutually exclusive SAVs compose the alternative MESCs and influence the required adaptation.

c. Develop the Adaptation-Rules on Structural or Algorithmic VPs

For every SAVP within each adaptation-strategy, develop the adaptation-rules specifying each variant and assign them to the first CEC, while excluding eventual non-valid combinations of SAVs through the adaptation-rules.

Step 9. Develop Adaptation-Rules on Behavioral VPs

At policy execution, once the structural or algorithmic variant (SAV) for each active SAVP has been selected at the first CEC, their parameter settings under the current context have to be defined. This is implemented in the weight-based technique with adaptation-rules assigned at subsequent consecutive execution cycles (CECs).

The adaptation-rules on behavioral adaptation (parameter settings) should be distinguished in typically three categories: (i) SAV-exclusive adaptation-rules, (ii) adaptation-rules common only in several SAVs, and (iii) adaptation-rules common to all SAVs.

Applying the consistency directives discussed at §6.1.4, two precautions are taken by the policy developer to prevent matching, selection and execution of behavioral adaptation rules belonging to non-selected SAVs:

a. The behavioral (parameter) adaptation rules should be specified with a cycle declaration subsequent to the CEC at which the adaptation-rules specifying SAVs selection are assigned. In this way at policy execution, the already chosen at the first CEC SAVs are known.

b. SAV-specific adaptation-rules on parameter adaptation should have as predicate in their Criteria conditions a reference to the supported SAV(s), in order to ensure only compatible adaptation-rules matching at policy execution.

The following successive actions are followed:

a) Identify the Behavioral VPs and their BVs

Identify the behavioral VPs on LTPGs or on short-goals and their mutually exclusive behavioral variants (BVs) along with their context sub-domains. Assign every behavioral VP to an adaptation-strategy.

b) Specify the Parameter Adaptation Rules for each BV

Within every adaptation-strategy and for each SAVP (or MESC), specify the behavioral adaptation rules for each BV, through which the parameter settings adaptation of the associated configuration- and/or QoS- and/or eventual auxiliary-variables will be specified at each context instance.

c) Specify the Parameter Adaptation Rules for LTPGs

Specifically for parameter adaptation on LTPGs – which are considered as behavioural VPs – the following successive actions are taken:

a. Define LTPG Resource Availability Levels

Define different resource availability levels – based on comparing monitoring metrics against threshold parameter values – with each level requiring different adaptation actions corresponding to a different LTPG BVP behavioural variant.

b. Identify the Control Variables

Identify one or more “control” adaptation-variables, whose adaptation at each CSD, may direct the LTPG towards accomplishing the goal.

c. Develop the Adaptation-Rules at each Resource Availability Level

Develop adaptation-rules to suitably adapt the values of the control variables at each resource availability level.

Step 10. Review the Policy

a) Review the policy until all goals and requirements are satisfied

Review the policy statements to make sure that all used variables are correctly declared, requirements and goals are expressed by adaptation-rules and all rules have a weight function value corresponding to the related adaptation-strategy.

b) Restructure the adaptation-rules in Criteria – Subpolicy pairs

Group together the rules with common cycle-declaration (CEC) and common Criteria, restructuring them in Criteria-Subpolicies pairs.

6.7.3 Stage 3 – Policy Testing & Verification Stage

Stage 3 – Test & Verify and Validate the Policy

This is the final policy authoring stage that is also considered interactively with the previous stages. It includes Steps 11 & 12.

Step 11. Test & Verify the Policy

a) Test the Policy

Note: *According to the IEEE Glossary of Software Engineering Terms 610.2-1990, Testing is the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.*

A simulated policy execution & verification application (PEVApp) is provided by the WBART methodology for offline automated test suite generation and simulated policy execution for testing Volare-compatible policies. A testing procedure is applied for any new policy for syntax correctness, failures and rules fault detection, as described in §6.6.

b) Verify the Policy

Note: *According to the IEEE Glossary of Software Engineering Terms 610.2-1990, Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.*

The above referenced simulated policy execution application (PEVapp) allows evaluation of real or test suite-derived simulated context and adaptation data through automated charts and calculation tables.

This offline analysis tool also allows simulated evaluation of the effectiveness of a Composite Policy to achieve the LTPGs within their horizons over the examined real or simulated usage models.

It may also be used to demonstrate offline the expected adaptation behavior to the policy developer, through automated charts on simulated execution results over test suites, thus assisting in appropriate policy evaluation and necessary corrections.

Step 12. Validate the Policy

After being tested and verified, a policy is validated by comparison of real or simulated to expected results.

Note: *According to the IEEE Glossary of Software Engineering Terms 610.2-1990, Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.*

6.8 Authoring an Application Policy

For a mobile application launching service discovery for a cloud service, in order to be subscribed for SR DCAA by the Volare middleware, it is needed to download and store in the Volare policy files directory of the device an application policy for the adaptation of the service request QoS-variables, and eventually for influencing adaptation on the middleware operation configuration-variables.

For the application policy design and implementation, reference is made again to the 3 Policy Development Stages and the 12 Policy Authoring Procedure Steps described in §6.7.

The application policy is expected to conform to the following generic issues as established by the global policy:

a. Adopt and use the Context & Adaptation Profile declared by the global policy and add only the application-specific QoS or the required intermediate variables.

b. Conform to the User Preferences Model, as it is configured in the global policy for each user preference mode and specify the adaptation-rules expressing the indicated adaptation behaviors. Note that the application policy developer has no access to the configuration of the UPI and the default values of the User Choices Profile.

c. Use the same Weight Assigning Strategy and the adaptation-concerns & adaptation-strategies set at the global policy and the UPI SWCs.

d. Conform to the quantitative Long Term Performance Goals set by the global policy for all applications launching cloud service discovery, and specify the application-specific adaptation-rules in relevance to LTPGs, with the same behavior in mind. It is only required to use in the application policy the same performance assessments derived by the global policy adaptation-rules concerning every LTPG.

e. Develop the application policy considering that it is expected to operate within the framework of the two-level Composite Policy. Assign to every rule: (i) priority level for intra- and inter-policy conflict resolution, (ii) a weight function value demonstrating its relative importance at policy execution, according to the WAS format. Rule priority levels assigned through keywords ensure the integrity of critical rules from inter- or intra-policy conflict and overriding.

6.9 Related Work

6.9.1 Adaptation Reasoning Techniques

Unlike all listed in §2.4 policy-based approaches [34][18][30], only the Volare APSL conflict resolution directives have the characteristics of allowing eventually multiple adaptation-rules to be selected and executed, establishing PWCP for the calculation of the adaptation results through the execution- and weight-values of the selected adaptation-rules. This characteristic elicits a different, more expressive adaptation reasoning approach based on the situation – action paradigm.

RAINBOW [59] on adaptation reasoning extends architecture-based adaptation by adopting *architectural styles* to tailor the adaptation infrastructure by encoding the developer's system-specific knowledge, identifying adaptation strategies and system concerns. Rainbow makes no attempt to tackle mobile environment requirements. Volare is using the two terms: *adaptation-strategies* and *adaptation-concerns,* although in a different meaning and within a different conceptual environment. RAINBOW bases the adaptation strategies on situation-action rules, which specify exactly what to do in certain situations. So does Volare albeit from multiple complementary adaptation-concerns, through the multiple selectable adaptation-rules feature and the WBART methodology.

MIMOSA [36] and its precursor CARE [69] implement Conflict Resolution on adaptation policy adaptation-rules originating from different entities (User, Service Provider) on the same head predicates, setting a resolution mechanism through adaptation-rule priority assigning leading to a single adaptation-rule selectable for each head predicate. Volare makes use of conflict resolution directives and participative weighted contribution common-sense rules, allowing eventually multiple adaptation-rules to be selected and contribute to the adaptation results according to a relative importance (weight-value) procedure.

The use of weights is extensive in the literature, not only in the mobile policy-based DCAA area, for calculating contributions to a parameter by multiple dimensions. However to our best knowledge, it is the first time a weight values based approach is used allowing multiple adaptation-rules to be selected and contribute to the adaptation results.

6.9.2 The User Preferences Declaration Mechanism

As attested by numerous research works [41][42][43][82][83] and surveys [13][61][103], satisfactory adaptation of a DCAA middleware to context change requires incorporation of user preferences – even at runtime. The user has his/her opinion of what high quality is and this opinion may change at runtime with the user's mood, the situation, or current interests [41]. Research papers on policy-based CAA mobile systems have approached the user preferences viewpoint from many different angles.

The first approach consists in equipping the application with an appropriate user preference declaration tool. HERA [31] uses adaptable user preferences set at application load time, as well as the dynamic user browsing history. In project [28] on e-commerce, the user sets dynamically preferences on the desired content with a fully developed User Interface. CARISMA [6] makes use of application profiles originating by applications, dynamically including in them user preferences, resource requirements and appropriate policy and similarly does Q-CAD [15].

The second approach for incorporating user preferences in the adaptation consists in doing so at the middleware level. CHISEL [7] dynamically updates policy adaptation-rules, including changing user preferences on policy at runtime. In MIMOSA [36] on the contrary, use is made of a full user preference declaration process and User Interface at each execution of an application, fine-tuning every service request.

Volare's middleware User Preferences Interface (UPI) – in contrast to the other works referenced – provides the UPI in addition to each application eventual UI, and elicits explicit alternative user preference declaration and dynamic fine-tuning of the adaptation behavior on policy-based generic quantitative cross-cutting concerns, common for all or most applications. Such generic concerns may be: optimal performance (highest bitrate), or low battery power use or low binding cost use, etc. It refers by necessity to generic concerns since it should cover different applications of unanticipated purpose and functionality.

6.9.3 Long Term Performance Goals in the Policy

Time-related tasks are not new in programming. In fact many above referenced research projects deal with device energy management as a time-related task with a goal. However, this is typically handled as a custom-solution in each project.

Research surveys on CAA make extended and in depth references to adaptation reasoning paradigms and viewpoints, but the issue of systematic support of policy-based LTPGs with finite horizons is not directly referenced [30], [33], [54], [56], [61], [74], [80], [81], [82] – with the exception of Kakousis et al [34] and [98].

Inspiration in this field was given by articles like [10] and [79] on self-adaptive software, and [53][55][75][77] and [99-100] on goal-oriented requirements engineering, that provided the impetus to examine common long term goals on repeated activities over finite horizons on mobile DCAA scenarios. Such common cases may involve management strategies over: monthly cost allowance on the cloud, monthly data volume allowance, battery power management and minimum level preservation over every discharging period, or quality metric goals over a time horizon, goals that are conventionally managed by the user, but could well be managed by the middleware adaptation logic

This Thesis considers LTPGs as time-related tasks over a sequence of similar stochastic events and actions, seeking to achieve a quantitative goal. Goal achieving over a sequence of stochastic context changes and relevant actions requires adaptation reasoning capability enriched with appropriate history information retrieval and statistical analysis over the usage data [99][100]. Although higher complexity specification languages like KaOS [39][100] do support goal-based management, in this work the simple Volare declarative DSL is enriched with notation for statistic-calculation-variables for declaring metrics over usage history parameters and the middleware is enabled for data storage and maintenance and simple statistical analysis functionalities.

In this work each quantitative LTPG is abstracted as a virtual consumable resource, so that a common formulation can be designed for easy and unanticipated introduction in an adaptation policy without any middleware code change.

7 Case Study

In the previous chapters 4, 5 and 6 of this Thesis, the three main constituents of the Volare Approach (the mobile middleware, the APSL and the weight-based adaptation reasoning echnique and methodology) have been presented.

In this chapter, the case study describes the detailed development of the global and the application policy for a mobile application launching cloud service discovery (CSD). The case study adaptation logic implementation is presented as realized by two of the motivating example stakeholders referenced in §3.1.1, namely: Jacob – the global policy developer, and Ronald – the developer of the mobile application launching cloud service discovery.

Since the Volare middleware supports dynamic context-aware adaptation for the optimization of the SR on cloud services by mobile applications, it is especially appropriate for “long-binding applications” like: audio- or video-streaming, navigation or video-calling – demanding adaptation even at runtime – in comparison to “one-shot” applications like web browsing. Consequently, the case study refers to a video-streaming application for service discovery and binding to services on the cloud.

7.1 Prototype Development & Operation

7.1.1 The Infrastructure designed

This chapter describes the elements of the infrastructure built in order to investigate the feasibility of the Volare approach by real or simulated data of the device performance on monitored parameters at alternative monitoring scenarios, and evaluate the contributions of the project.

1. The Prototype Middleware Modules

The Volare middleware prototype – whose conceptual model is described in chapter 4 – main modules: Context Monitoring M., Adaptation M. with a Policy Engine, Service Request M., Binding M., Context & Adaptation History M. (CAHiM) were designed and installed on a HTC Hero smartphone.

A generic UPI application was also designed and installed for selecting & changing user preference.

The **Volare Policy Files** directory for the global and the application policies was created at the manual middleware installation on the mobile device storage disc.

2. The Generic Global Policy File

A global policy has been developed and stored in the reserved Volare Policy Files Directory. It contained the Declarations part on all non-statistic-calculation-variables, and the Criteria and Subpolicies part without any LTPG related adaptation-rules. It was a simpler policy than the case study full policy described in §7.4. It was tested and verified.

The User Preferences Model was configured by setting the values of declared user-customizable parameters (User Choices Profile according to §4.2 and §6.4) without requiring the use of the UPI application. The user-customizable variables were declared in the policy not as context- but as calculation-variables and appropriate values were directly assigned – for instance: ***float CalcVar priceMax = 0.0024;*** **–** setting the maximum price per MB downloadable from a cloud service to 0.0024 GBPs/MB). The disadvantage vs. UPI configuration is that these parameters keep the set values constant until policy update, while through the conceptual UPI they can be modified directly by the user, even at runtime.

3. Video-streaming Mobile Application

A generic mobile video-streaming application ***VSTREAM*** (of no adaptation capability) has been developed as described in §7.2 and installed on the mobile.

4. Application Policy File

A simple application policy file was designed and installed, tested and verified, including only the generic part without the LTPGs of the full policy file described in §7.5 and listed in Appendix and B. No statistic-calculation-variables or LTPG associated variables were declared and any LTPG related adaptation-rules were omitted at this stage.

5. Test-bed with four cloud services designed in Amazon S3

A test-bed with four cloud video provisioning services on the same content but with different QoS levels has been designed on Amazon S3 cloud provider, with characteristics described in chapter §7.2.3.

6. A Virtual Cloud Service Broker/Provider for Service Selection

A test CSB/CSP has been designed and installed on the mobile device. Its task consists in receiving the adapted SR of the video-streaming application and discover from the available four cloud services of the test-bed the cloud service with QoS levels most fitting to the SR. Then, it dispatches a Service Offer and if the middleware Binding Module accepts the Service Offer, service binding is instantiated to the selected service, as described in §7.2.4.

7.1.2 Middleware Expected Operation

At a SR on the cloud launched by a mobile application having an application policy stored in the Volare Policy Files Directory, as described in Ch. 4, the Volare middleware transparently to the application intercepts the launched SR, adapts it to the current context (including the current user preference mode) and policy requirements and goals and forwards the adapted SR to the CSP for appropriate service discovery.

It receives a Service Offer by the CSP and evaluates it on the basis of the SR QoS values and the binding margins established at policy execution (i.e. the allowed deviations). If the Service Offer is within acceptable margins, then it is accepted and service consumption begins.

If at runtime of service binding the context changes and policy execution derives new SR QoS terms values that are assessed on policy-based parameters as being in significant deviation to the current in force QoS provisioning values, then service rediscovery will take place at runtime under new QoS values for the SR.

7.1.3 Prototype Operation

The above prototype executables (middleware, video-streaming application, virtual CSP) were installed on a smartphone Samsung HTC Hero with O/S Android 2.1. The device has operated for a period of more than a month with CSD sessions taking place at various times every day at the developer's instigation, while the device was also used at other times for non-cloud conventional use (phone calls, SMS, MMS, surfing the web, audio, gaming, etc.).

The following parameters were recorded automatically by the CAHiM database during each operation step at each repeated **recheck cycle** (of policy-based duration, typically: 30 or 20 or 10 s) at different successive **operation steps** during every CSD session.

1st Step: Context monitoring – The middleware database recorded, within every CSD session. the monitored context data values at each monitoring "recheck" period:

- the *dataRowNo,* the CSD *sessionNo* and the *date* and rowStartTime and rowEndTime of each recheck cycle
- the *battery* level value at the beginning and end of each recheck cycle (the last one being also the value at the beginning of next recheck cycle within CSD session)

- the *bandwidth* value, as described below, as well as the *connectionType* (network) indication (E or 3G or H)
- the *userPref* current value from the UPI
- at the first recheck cycle of each CSD session, the *webMBs,* web MBs downloaded through the MNSP at the ended non-cloud session.

2nd Step: Policy execution (calculation) – Similarly at the same data row, the CAHiD database recorded the new policy execution results on the above data, namely the QoS-variables (global and application-specific) and the middleware configuration-variables values.

3rd Step: QoS evaluation decision-making on rediscovery – At runtime policy execution only, the decision by the QoS evaluation mechanism of the Binding M., on rediscovery or no rediscovery (continue as before without adaptation if policy-based thresholds on the difference between the new and the in force QoS values is not exceeded).

4th Step: Adapted SR dispatch to the CSP – In case the decision in the previous step is for discovery or rediscovery, the SR QoS terms are adapted by the Binding M. on the new policy execution results and the SR is dispatched to the virtual CSP.

5th Step: Service discovery or rediscovery by the CSP – In case of service (re-) discovery, the virtual CSP receives the adapted SR and executes service discovery from the test-bed of four cloud services and (since all are available) selection of the fittest to the SR QoS terms. The corresponding Service Offer is dispatched to the BindingM.

6th Step: Service Offer evaluation – The BindingM intercepts the Service Offer by the CSP and evaluates it on policy-based parameters on whether the service offered satisfies the adapted QoS terms and policy derived preferences through the binding margins. In the middleware prototype the BindingM only accepts the Service Offer and binds to the service, and only in the case of runtime rediscovery it may reject the SO (continuing as before).

7th Step: Service Binding or Binding Continuation for the recheck time period – The middleware specifies by policy execution the duration of each recheck cycle – except if there is manual termination of the CSD session by the application. The video-streaming application at initial discovery or rediscovery binds to the discovered service or it continues operating in the already bound service. The Context MonitoringM monitors and the CAHiM records the values for the service binding, at the recheck cycle namely:

- the corresponding bound *serviceID* and the *QoS* terms values
- the value of *cloudMBs* (MBs downloaded during at the recheck period on the current CSD session) by the end of the recheck cycle
- the value of *cloudCost* due to the *cloudMBs* downloaded by the end of the recheck cycle.

These repeated seven operation steps at each recheck cycle at a CSD session are terminated at the end of each CSD session and the middleware remains idle, however measuring eventual *webMBs* through the MNSP during the non-cloud activity, till next CSD session.

7.1.4 Brief Description of the Video-Streaming Application

Video-Streaming Activity

Video-streaming enables users to start video playback while the content is being downloaded. Users can watch video on their mobiles, either through a web browser or through a mobile application. Depending on the CSP, a different application may be required for different providers or video containers [128][110].

The CSP (or CSB) is provisioning authorized users through their MNSP contract, with videos of the requested content at different qualities and different encoding bitrates depending on the SR QoS terms, at a price per MB downloaded depending on market factors (demand, traffic load, QoS terms, etc.).

At a typical streaming session, the video content is transferred in two stages: the initial *buffering stage* followed by the *steady state stage* [128]. At the *buffering stage* the data transfer is limited to the end-to-end available bandwidth. The video begins playback when a sufficient amount of data is available in the device buffer and does not wait for the buffering stage to end. In the steady state stage, the average download rate should be slightly larger than video encoding bit rate. The buffering stage ensures that the device has a sufficient amount of data to compensate for variations in the end-to-end available bandwidth during video playback.

The Video-streaming application VSTREAM

A shallow adaptation mobile video-streaming application "*VSTREAM"* has been developed for streaming videos by binding to a video-provisioning cloud service through a cellular Mobile Network Service Provider (MNSP).

The application is installed on Evelyn's smartphone along with the Volare DCAA middleware and is subscribed to it, by having downloaded and stored an application policy file in the Policy Files Directory of the mobile device.

The mobile device user, Evelyn, at different times in the day through the VSTREAM application requests video-streaming through application- or user-selected Cloud Service Provider(s) (CSPs), on world or local or entertainment or scientific or cultural news, financial information on the stock exchange, etc. Evelyn has a Mobile Network Service Provider (MNSP) contract authorizing a monthly allowance of 4000 MBs downloadable through the cellular network and 5 pounds/month allowance for binding to the referenced cloud services.

7.1.5 A Virtual CSP/CSB on video-streaming cloud services

A virtual CSB/CSP has been designed and installed on the mobile device, as part of the infrastructure designed for the evaluation of the Volare approach described in §8.1. The virtual CSP receives the adapted SR of the video-streaming application and makes use of a hard-coded service selection algorithm to identify the most fitting discovered and available service from alternative video provisioning cloud services on the same content but different QoS terms that have been designed as a test-bed on Amazon S3.

The service selection algorithm used by the virtual CSP

The CSP discovers cloud services on the same content and ranks on how well they fit the Service Request QoS terms. Suppose that the received adapted SR has n QoS terms and for the QoS term i, the SR and respective Service Offer values are respectively denoted as: *QoSReqi* and *QoSOffi*, 1 <= i <= n.

It is required for each QoS term i value percentage difference not to exceed the binding margin *bMi* plus *rediscQoSThreshold*:

If bMi >= 0 then:

$$0 \le 100 \times (QoSOffi - QoSReqi) / QoSReqi \le bMi + rediscQoSThreshold \quad (1)$$

If bMi < 0 Then

$$100 \times (QoSReqi - QoSOffi) / QoSReqi \ge bMi - rediscQoSThreshold \quad (2)$$

In order for an offered service to be accepted, the above constraints (1) to (2) should be satisfied for every QoS term value of the Service Offer. The same mechanism is applied when at runtime, the new SR QoS values are evaluated (as Service Offer QoS values) against the last adaptation in force QoS values.

Then, it dispatches a Service Offer and if the middleware Binding Module accepts the Service Offer, service binding is instantiated to the selected service (see §7.1.5), otherwise a new SR on modified QoS terms is launched.

7.1.6 Test-Bed for selection of Cloud Services

A test-bed has been designed at Amazon’s Simple Storage Service (S3), simulating a virtual CSP with the capability to provide four different streaming cloud services, on the same video content in four different video qualities and encoding bitrates, at a cost per MB downloaded.

It is assumed in the case study that the cost per MB is not the same for all service qualities, due to marketing reasons.

Table 7-1 presents the basic characteristics of the four cloud services described, while several other QoS terms values are supposed to depend on the current demand and are specified at the Service Offer in response to a SR by the VSTREAM application.

Table 7-1 – The Case Study alternative Services on the Cloud

SERVICE ID	*s1*	*s2*	*s3*	*s4*	*Units*	Max Ref. Value	*Units*
Cost per MB costQoSProv	60	70	80	100	*% Max Reference Value*	0.0024	*pounds/ MB*
Encoding bitrate bitrateQoSProv	28	56	128	256	*Kbps*		*KB/s*
frames per second fpsQoSProv	67	73	80	100	*% Max Reference Value*	30	*frames per second*
Reliability QoSProv	91	92	93	94	*%*		

7.2 The Monthly Duration Usage Data

During the monthly operation of the middleware, the usage data were recorded and from time to time transferred at device idle time, since no aggregation capability was yet provided to the database module.

The two important goals for the derived usage data consist in:

- evaluating adaptation behaviour with the initial Composite Policy and identifying challenges,
- serving to provide a context usage model that may be employed for simulation of the adaptation behaviour in more complicated policies with LTPGs alternative monitoring scenarios.

Some further details are provided below, since these usage data will provide the basis for evaluation purposes and drawing of conclusions. The following issues were to be resolved to obtain the usage data:

- monitoring issues on context parameters
- data aggregation to more compact but meaningful usage data.

7.2.1 Monitoring Issues on battery, bandwidth, recharge time

Battery – The *battery* level indications at each recheck period were monitored and recorded in integer percentage units (this is the precision given by the system), thus at small recheck cycle time intervals (of 10 to 30 s) there is no evident battery drop. This caused the need for usage data aggregation, in order to provide data on longer time intervals.

Bandwidth – The *bandwidth* indications are not provided by the O/S or a device application, only the connection type on the mobile network: *E* for *EDGE, 3G* for *UMTS* and *H* for *HSDPA* or *HSPA* is polled and recorded. As a result, the following procedure was put in effect during the one month operation of the infrastructure. A freeware speed test application has been installed and was activated by the developer/user just before the beginning of a CSD session and the derived value (average of three times) was inserted manually as *bandwidth* value, just before the CSD session. At each recheck time during runtime of a CSD session, the middleware monitors the connection type at the O/S and if it changed from the previous recheck period at a steadily kept value, indicating bandwidth change, then a predetermined typical bandwidth middle-range value was adopted by the middleware, as in Table 8-1 below. As a result, if at the recheck time, a steady change from the initial connection type was monitored, for instance from **3G** to **E** or **H**, then the bandwidth value recorded was the prefixed middle-range value in 8-1. Note than the eventual bandwidth error is of minimal importance in service selection, since the video encoding bitrates for service1 and service2 (28 and 56 KB/s in corresponding data rates) correspond respectively to E or 3G, and only services 3 & 4 (128 and 256 KB/s respectively in equivalent data rates) correspond to H. For instance, having a bandwidth corresponding to 80 KB/s instead of 125 KB/s could not lead to selection of a different service than service2 (on bandwidth reasons), since service 3 (128 KB/s) would require 128/0.8 = 160 KB/s.

Figure 7-1 – Prefixed Bandwidth Values per Connection Type

Connection type	**Typical avg value at middle-range of connection type**
EDGE	320 KB/s corresponding to a data rate of 40 KB/s
3G (UMTS)	1000 KB/s corresponding to a data rate of 125 KB/s
H (HSDPA)	3000 KB/s corresponding to a data rate of 375 KB/s

Bandwidth values, for chart visibility reasons, will be referenced in this work not as bit rates (kbps) but at the equivalent data rate units (KB/s).

Non-cloud Session battery drop – The battery drop from the end of the last CSD session to the next (i.e. the battery drop of a non-cloud activity session), is the difference of the battery levels monitored respectively at the end of the previous and the beginning of the new CSD session, recorded in the database. It should be noted that as the battery level accuracy provided by the device is in integer % values, often the battery drop for short time intervals is a zero value.

Battery discharge cycles during the monthly period – The developer/user established battery recharge of the mobile device during the considered monthly by the end of every 3rd day. Every new battery discharge cycle was identified and recorded by the middleware by the notable increase in battery level at the beginning of the first CSD session of the next day. The beginning battery level in a new battery discharge cycle is the new value first monitored after recharge.

7.2.2 Deriving the Context Usage Model – Assumptions

Consider a battery discharge cycle period of three days. At the each recheck period of (30, 20 or 10 s) within each CSD session, the middleware monitors and records the context-variables values, including battery level. The mobile device activity can be distinguished in succession of one ***cloud activity (CSD) session*** followed by one ***non-cloud activity session*** (with phone calls, SMS, MMS, stand-by duration, gaming, audio, etc.) and so on.

If the mobile device usage data is kept **intact** on non-cloud activity sessions and also the *start* and *end times* (and consequently durations) of cloud activity (CSD) sessions, then modified usage models can be established by modifying only within the CSD sessions, context parameters like: *bandwidth* or *user preference mode or a user choice* value at the *User Choices Profile*. In this case by modifying a context parameter value within a CSD session, the associated context- and adaptation-variables may change value too, for instance: ***battery* drop** or ***cloudMBs*** or ***cloudCost*** values within every recheck period (i.e. MBs downloaded and the associated cost at recheck cycle at CSD session), if a different service is discovered at bound with different QoS terms.

a) Assumption for battery drop within a non-cloud activity session

The assumption is made that, if the mobile device usage context data is kept unchanged on all non-cloud activity sessions (with assumed unchanged activity, like: phone calls etc.), then *battery drop* within each non-cloud activity session (from the end of the previous CSD session till the beginning of the next CSD session) is expected to remain the same, even though context parameters at the previous CSD session (like: bandwidth, battery level or user preference/user choices) may change. The underlying theoretic basis is that battery drop of a mobile device (and in in general of energy drop) in a specified period depends only on the activities during this period and their duration and is independent of the initial (starting) level of battery power.

Once the battery levels at beginning and end of each CSD session are recorded, then the battery drop for each incumbent non-cloud session are directly deducted and may be used on modified CSD sessions, assuming non-cloud activity sessions intact.

Usage data aggregation – Since the prototype generic CAHiM did not have aggregation capability, aggregation has been implemented on the usage data offline on each adaptation occurrence within CSD sessions (each with at least one adaptation – the initial adaptation). Within a CSD session there may be tens of recheck cycles and corresponding data rows, but the service bound to may be the same (no adaptation). Occasionally, adaptation occurs and binding to a new service due to significant context change.

All data rows within an adaptation cycle may be aggregated, retaining the bandwidth at which adaptation occurred, while the time duration and the values of increasing/decreasing parameters (battery drop, cloudMBs, webMBs, cloudCost, etc.) are all updated at the total adaptation cycle duration. This lead the usage data from an initial database with about 1800 data rows (one at every recheck cycle) to 217 adaptation cycle aggregated data rows.

Aggregation to adaptation cycles increases the time interval of the context monitored and reduces context data uncertainties, especially for battery drop, since in this way battery drop now is calculated in minutes instead of 10, 20 or 30 seconds.

b) Assumption on Systematic Monitoring Errors

Noise is introduced in the simulation study data by the device monitored context parameters values due to the fact that the device indications on physical parameters on battery are approximate values to the nearest integer, and for bandwidth values at runtime to even larger scale intervals.

Note though that at each comparison of outcomes, all other parameters (including noise from approximate context data) are the same at all considered monitoring scenarios, consequently this error is practically the same (or with small difference) in all outcomes.

The device usage context data recorded on a monthly period is listed in Appendix G for documentation and reproducibility purposes and will be used for evaluation purposes in chapter 8. The chart in Fig. 7-1 refers to the usage data.

7.2.3 The Context Usage Model

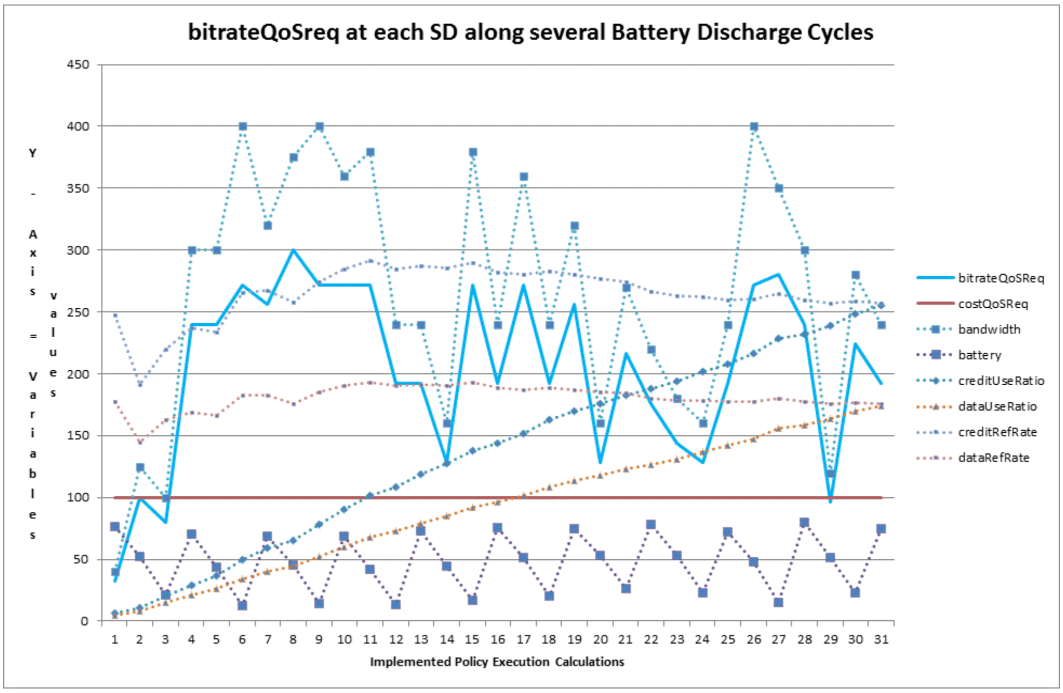
The usage data refer to 217 adaptations that have taken place within a sequence of 154 CSD sessions over a monthly period, of total duration of CSD sessions on the cloud: 42.999 s, aggregated out of almost 1800 recheck cycle data rows.

From the aggregated usage data, the context data corresponding to the sequence of adaptations within the recorded CSD sessions were isolated in a two-dimensional context data matrix, as they present the basis for simulation studies, called in this work the context usage model.

The context usage model data, also used in the regression analysis above, are detailed in Appendix G for documentation and reproducibility purposes and will be used for simulations in chapter 8.

The following chart on usage data parameters is demonstrating policy execution results on device adaptation behaviour QoS parameters: *bitrateQoSReq* & *costQoSReq* over the monthly usage data period. In dotted lines context parameters like: battery level, bandwidth, creditUseratio, dataUseratio, creditRefRateand dataRefRate, as derived by the infrastructure with adaptation logic consisting of the generic policy files described in §7.1.

Figure 7-2 – QoS Variables over the Monthly Usage Data



7.3 Outline of the Adaptation Logic

As described in chapter 4, at an active SR on the cloud by a mobile application, the adaptation logic is constituted of a Composite Policy automatically formed by the middleware by merging the global policy adapting the middleware operation and serving generic goals and the application policy for the application-specific adaptation.

Each Volare compatible policy file (global- or application-level) includes a Variables Declarations part, and a Policy Rules part further distinguished in a Criteria part with the groups of predicates of the adaptation-rules with common predicates and CEC and a Subpolicies part with the groups of the corresponding adaptation-statements.

1. Policy Variables Declarations Part

A policy file includes an extensive Variables Declarations part for the context-, configuration- and QoS-variables, as well as for the intermediate variables (declared as calculation, statistic-calculation or auxiliary-variables) since it ensures the declaration and monitoring of all the variables involved.

To facilitate the task of the application policy developer, all variables declared in the global policy may directly be used (as named) in the predicates or adaptation-statements of the application policy without re-declaring them.

2. Policy Rules Part

In a Volare policy file, when naming the Criteria-Subpolicy pairs of the policy file, the following indicative application developer-friendly format is suggested:

Xn_OBJECTIVE_CHARACTERISTIC, where:

X = G for global policy and A for application policy,

n stands for the corresponding cycle declaration (CEC),

OBJECTIVE stands for the objective served by this Criteria-Subpolicy name,

CHARACTERISTIC is optional and stands for eventual characteristic of this Criteria-Subpolicy pair within this objective. For instance: “G1_CREDITMETRICS_ USGBASED” or “G1_DATAMETRICS_PREFIXED”, or “G2_DATALTPG_HIGH” or “G2_BATTERYLTPG_LOW”.

7.3.1 Global Policy Objectives & Structure

The global policy aims at managing the middleware DCAA operation activities:

a. context monitoring on device resources, user preferences on the UPI and external and computing environment parameters (like bandwidth, GPS, etc,),
b. adaptation reasoning through the Policy Engine,
c. initial service request (SR) interception and policy-based SR adaptation, as well as adaptation of the middleware components,
d. functionality for offered service evaluation and either binding, or service request adjustment and rediscovery.

Additionally, the global policy integrates the User Preferences Model by configuring the UPI for user preference mode selection and customization of the adaptation logic on predetermined threshold parameters. Finally, it introduces quantitative LTPGs on cross-cutting concerns, common for all applications.

It should be noted, that the global policy is designed without anticipating the applications that may subscribe to and use the middleware. Additionally, it is understood that it may be modified concerning several features for each type of mobile device, like additional sensors and capabilities.

The global policy file includes an extensive Variables Declarations part for the supported context- (including the user preferences context-variables), configuration- and global QoS-variables, as well as for the intermediate variables (declared as calculation-, statistic-calculation- or auxiliary-variables) since it ensures the declaration and monitoring of all the variables involved, with the exception of the application-specific ones.

Policy Rules Structure in CEC Segments

According to the Policy Authoring Methodology of the weight-based adaptation reasoning technique, the adaptation-rules part of a policy file is typically designed in three consecutive execution cycles (CECs). Jacob adopts the above generic model for the global policy, since it there are concerns for all three CECs, as follows:

1st CEC – At the 1st CEC part of the global policy file, adaptation-rules Criteria-Subpolicies are included on:

The adaptation-rules concerning configuration-variables that configure the Adaptation Module Policy Engine parameters required for further policy execution, like: *cyclesMax* or *defaultBindingmargin*.

The adaptation-rules concerning configuration-variables that specify variant selection on structural (or functional) VPs (for instance specifying the (non-default) activate state of active-on-demand sensors or functionalities – like Wi-Fi On or GPS On), as well as on algorithmic variant selection or algorithm modification (through alternative metrics and threshold parameters values to be selected and used at policy execution).

The 1st CEC segment includes 4 Criteria-Subpolicy pairs, one common for all three LTPGs – concerning the use of usage-based metrics, and one LTPG-specific for every one of the three quantitative LTPGs of the global policy on: battery power, credit and data volume, specifying threshold parameters for the resource availability level assessment algorithm and the adaptation actions algorithm for each LTPG. Additionally, the “Default” Criteria-Subpolicy includes the adaptation-rules specifying the default “prefixed” metrics and default threshold parameter values for the LTPG algorithms.

2nd CEC – The 2nd CEC segment of the global policy is designed by Jacob to include the adaptation-rules concerning the LTPG resource availability level assessment algorithm of reach LTPG, thus serving as a reference for the common LTPGs that concern both the global and every application policy.

It includes 6 LTPG-specific Criteria-Subpolicy pairs, two for every one of the three quantitative LTPGs of the global policy on: battery power, credit and data volume, specifying the “High” or “VLow” resource availability level for each LTPG, and 1 “Default” Criteria-Subpolicy specifying the default “Low” resource availability level for each LTPG (with lower weight values).

3rd CEC – The 3rd CEC segment of the policy includes the Criteria and corresponding Subpolicies containing the adaptation-rules specifying values to the remaining middleware configuration-variables, as well as the global QoS-variables representing parameter adaptation for the “control-layer” of the adaptation logic. It concerns adaptation-rules for both “long” and “short-term” adaptation goals. Namely: 4 Criteria-Subpolicies on the user preferences BVP (one for each user preference), 3 Criteria-Subpolicies for the Monthly Credit LTPG BVP – corresponding to the “High”, “Low” and “VLow” availability levels, and 1 “Default” Subpolicy including the adaptation-rules on default values or invariants.

7.3.2 The Application Policy Objectives and Structure

The case study mobile application concerns video-streaming, an especially demanding task for mobile systems both on the computational load and on the cellular data transfer activity of the mobile device.

The application policy, based on the current context as monitored by the middleware through the global policy context-variables and the Composite Policy requirements and goals, aims at specifying the application-specific QoS-variables for discovery of the most appropriate service. A fundamental requirement is to ensure at all times a **video encoding bitrate** value >= 80% of the available bandwidth, so that the play-back quality is not perturbed. If however the context changes at runtime, typically due to bandwidth variation, then adaptation is required and rediscovery of another cloud service provisioning the same content at a different more appropriate quality, as it is calculated by the policy execution.

The application policy developed by Ronald specifies the parameter adaptation for the SR QoS-variables, as well as eventual middleware configuration-variables of interest to the application. Although most of the required variables are declared by the global policy and the LTPGs availability levels are also specified in the global policy 2nd CEC, Ronald has to specify the adaptation actions for the battery and data volume LTPGs that may be managed by the bitrate application-specific QoS-variable as control variable.

An additional, application-specific short-term goal is designed by Ronald, concerning runtime disruption minimization at video-streaming time at a CSD session if rediscoveries per 5min exceed a threshold value, by only accepting rediscovery at runtime if it is technically unavoidable (like bandwidth drop). For instance, rediscovery due to bandwidth increase or due to gradual battery drop or due to high credit use at runtime within such a CSD session is to be rejected.

All adaptation-rules of the application policy concern the 3rd CEC of the Composite Policy and are included in 14 Criteria-Subpolicies.

7.4 PAM – Authoring the Global Policy

Authoring a global policy file in the weight-based technique requires implementing the procedural 12 *Steps* described in §6.7 and takes place at the aforementioned three successive *Policy Development Stages*, that are presented below in sub-subsections 7.4.1 to 7.4.3, and in the sequence indicated.

Each of the three *Stages* is autonomous to a certain degree and typically iteration is required over the Steps mentioned, for corrections or enrichment at policy design and implementation. Finalization of each Stage is required for advancing to the next one, although iterative review of Steps of previous Stages may be necessary.

Enumeration of the procedural *Stages* and *Steps* identical to §6.7 will be kept, to facilitate the developer or reader to correspond the issues in the Policy Authoring Methodology in §6.7 to its application on the case study in this chapter.

This procedure is most important for the global policy efficient authoring as it requires all the steps that are described below, in comparison to application policy authoring where many steps are unnecessary or of marginal burden since they are already implemented at the global policy development.

7.4.1 Stage 1 – Policy Design Stage

Step 1. Define the Context and Adaptation Variables

The context-variables (like: battery, CPU, bandwidth etc.) are predefined at middleware design time, while the user preferences context-variables are configured at the User Preferences Interface in Step 5 below.

The adaptation-variables include: (i) the middleware configuration-variables which are already defined at middleware design time, described in Table 4.3, and (ii) the global QoS-variables which are common for all applications.

The Table 7-2 below presents the case study global policy context- and adaptation-variables.

Table 7-2 – Case Study Context & Adaptation Profile

Context-variable ID	Description	Datatype/ Units	Example
Device & Computing Environment			
battery	Remaining Battery level	percentage/%	80
Network			
commChannel	Active Communication Channel	string / -	"GSM" or "Wi-Fi"
bandwidth	The current max data rate	integer / KB/s	384
User Preference Parameters provided through the UPI			
userPref	The User Preference Mode	string / -	"Normal", "LowCost", "SaveBattery", "HighQuality"
allowOptimization	"Allow usage-based Optimization?"	string / -	"Y", "N"
priceMax	Maximum price of the cloud service resources per MB downloaded.	float / pounds/MB	0.003 pounds per month
Strategy-weight-coefficients (SWCs)	One SWC for each adaptation-strategy & user preference.	float / -	>= 0 <= 1
cloudCostMonthlyAllowance	Monthly cost allowance on cloud services	float / pounds/month	5 pounds per month
overallMBsMonthlyAllowance	Monthly data volume allowance by the MNSP	float / MBs	4000 MBs
minBatteryLevel	Minimum battery level reserved for emergencies at the discharge cycle	percentage / -	20 %
serviceID	The serviceID that the device is currently bound to	string / -	service3
costQoSProv	Provisioning cost (%) of the current Recheckcycle	percentage /-	80

Adaptation-variables	Description	Datatype / Units	Example
Middleware configuration-variables			
See §4.3	Predetermined middleware components configuration variables		
Global QoS-variables			
costQoSReq	The SR max requested price per MB rate	percentage / % of unadapted value	80
responseTimeQoSreq	The SR max req. response time	float / s	<= 0.7 s
availabilityQoSReq	The SR min req. availability	float / -	>= 0.96

Step 2. Define the Adaptation-Concerns & Adaptation-Strategies

Following the weigh-based technique and the provided generic models, Jacob finds satisfactory for the scenario and decides to adopt the generic Adaptation-concerns & Adaptation Strategies Model on the five major adaptation-concerns that need to be considered at policy design, as described in §6.1.2 and outlined synoptically in Table 7-3 below.

The policy will be built around these five targeted adaptation viewpoints through adaptation-rules belonging in the corresponding five adaptation-strategies.

Table 7-3 – The Volare Five Generic Adaptation-Concerns/Strategies

1 Adaptation-concerns/Strategies
1.1 Sgen – Adaptation-strategyategy: Generic Operational Constraints
1.2 Sperf – Adaptation-strategyategy: Performance optimization
1.3 Sres – Adaptation-strategyategy: Optimization of resource (battery) use
1.4 Scost – Adaptation-strategyategy: Optimization of cost of binding
1.5 Sdisr – Adaptation-strategyategy: Minimization of disruption at runtime

Step 3. Configure the User Preferences Model through the UPI

As indicated in the methodology in §6.4, the global policy developer configures the Volare User Preferences Model through the User Preferences Interface (UPI) application, assigning names and values to each sensor representing a user preference element. The configuration operates at two levels, as described in the following paragraphs.

a) Configuring the alternative User Preference Modes at the UPI

Jacob decides that the default four user preference modes: “HighQuality”, “Normal”, “LowCost” and “SaveBattery”, as specified by the generic User Preferences Model of the Volare methodology (see Table 7-5), are fully satisfactory for the considered scenario.

The User Preferences Model allows the user to select, even at runtime, for the context element **userPref** representing the user preference mode, one of four available user preference modes:

- At *userPref* = *“HighQuality*” to bind at the service with the highest attainable QoS terms values, without restriction on device resources
- At *userPref* = *“Normal”* to bind at a service with high QoS values taking into consideration the resource levels and policy constraints
- At *userPref mode = “LowCost” or “SaveBattery”* to bind at a service with reduced QoS values, in view of expected (not necessarily monitored) low availability in battery power or credit for binding to cloud services.

Table 7-4 – User Preference Mode specific Adaptation-Rules

User Preference	Relevant Adaptation-Rules
Normal	Set SR the generic QoS terms at high values and costQoSReq at 100% of the unadapted value.
HighQuality	Set all SR QoS terms at maximum attainable values (<= unadapted values). No restrictions even at decreasing resources (battery, credit, data volume, etc.). Set recheckRate at 10 secs for frequent monitoring.
LowCost	Set the binding price global QoS term costQoSReq at a reduced binding price.
SaveBattery	Set battery power related SR QoS terms values at 50% of the Normal value to reduce power consumption. Set recheckRate to 30 secs to reduce power consumption.

Jacob also decides that the scenario does require a 2nd layer of user preference options, but not like the generic: “business” vs. “personal”, or “data roaming On” vs. “data roaming Off”, but instead for the new parameter: “Allow Usage-based Optimization” with options “Y” or “N”. Consequently, he configures the first string type UPI sensor as “user preference mode” and the second string type UPI sensor with the above parameter on the two selectable alternative options and inactivates the remaining available string type sensors of the UPI.

b) Specifying the default Strategy Weight Coefficients

Jacob decides to increase the default number of adaptation-concerns and strategies to 6, creating a new one on the Data Volume LTPG (Sdat), which is not the same with the Cost adaptation-strategy.

Jacob specifies the default User Choices Profile by first configuring the 20 numeric strategy-weight-coefficient (SWC) sensors (4 user preference modes x 5 adaptation-strategies), assigning a name, a default value and an upper and lower percentage margin to each one of them, already presented in Table 7-4.

By default the remaining wSgen (i.e. the single SWC of the Generic Adaptation-concerns adaptation-strategy) is set to 1 (the maximum weight coefficient value) at any user preference mode, since this adaptation-strategy represents adaptation-rules independent of the current user preference and of the highest importance, and is specified by a declaration statement as constant calculation-variable in the policy.

Table 7-5 – Case Study Default Strategy Weight Coefficient (SWC) Values

SWC / userPref	Normal	SaveBattery	LowCost	HighQuality	Upper Bound	Lower Bound
Sgen	1.00	1.00	1.00	1.00		
Sperf	1.00	0.50	0.50	0.80	1.00	0.00
Scost	1.00	0.50	0.80	0.20	1.00	0.00
Sres	1.00	0.80	0.50	0.20	1.00	0.00
Sdat	1.00	0.80	0.50	0.20	1.00	0.00
Sdisr	1.00	0.40	0.40	0.10	1.00	0.00

The above 24 SWCs (four last rows) that depend on the UPI are declared as context-variables (i.e. in the *ContextVar* Repository). By default the maximum value of a SWC is 1 and the minimum value is 0. The declaration statements for the SWCs in the global policy may be:

```
float CalcVar wSgen == 1.00;

float ContextVar wSperf == 1.00 * UPI.wSperf;

float ContextVar wScost == 0.80 * UPI.wScost;

float ContextVar wSres == 0.50 * UPI.wSres;

float ContextVar wSdisr == 0.20 * UPI.wSdisr;
```

float ContextVar wSdat == 0.80 * UPI.wSdat;

c) Configuring other UPI-related context-variables

The UPI allows sensors configuration (assigning a name, a default value and eventually a customization margin for the user) for other global policy parameters that may or should be customized by the user.

Such parameters values (constants) may be goal values or related threshold values on Long Term Performance Goals, which will be set at the definition of every LTPG. For example, although the metrics for LTPGs are specified in the policy as calculation-variables, default values for some parameters or goals, may alternatively be declared as context-variables, so that they may be user-customizable through the UPI:

float ContextVar monthlyCreditAllowance == 5.0*UPI.creditAllowance;

specifying monthly credit allowance for binding to cloud services to 5 £/month,

float ContextVar monthlyDataAllowance == 4000*UPI.dataAllowance;

specifying monthly data allowance through the MNSP to 4000 MBs/month,

float ContextVar minBatteryLevel == 20*UPI.batteryVLowLevel;

specifying the minimum battery level that the battery should preserve for eventual emergency needs by the end of the discharge cycle.

Similarly, several threshold parameters in the Criteria conditions or the adaptation-statements may be declared as context-variables, user-customizable through the User Choices Profile, declared in the generic format of the APSL:

[float | integer | percentage] ContextVar parameterID == numeric-value*UPI.parameterID;.

Step 4. Specify the Weight Assigning Strategy

Jacob adopts the generic Weight Assigning Strategy, recommended by the weight-based technique and the weight function format in §6.3, through the two explicit coefficients (strategy and rule weight coefficients). The SWCs monitored at the UPI, take into consideration the selected current user preference mode, as well as the strategy-weight-coefficient-modifiers that may be set by the user at the UCP customization (default value is 1.0) :

rule weight function = wSstrat * wcRule

Thus, at the and of the RHS of each adaptation-statement, Jacob will assign the rule weight function in the format: "**(wSstrat * wcRule);**".

The following PAM-determined actions are implemented in Step 4:

a) Setting the strategy-weight-coefficients SWCs

The strategy-weight-coefficients (SWCs) – one for every adaptation-strategy under each user preference mode – are specified when configuring the UPI as described in *Step 5* below.

b) Setting the rule-weight-coefficients wcRule

Jacob will determine the rule-weight-coefficient *wcRule* when each rule is specified, in relation to its role and relative importance, after consideration of the other adaptation-rules on the adaptation-variable that may be selectable at policy execution through overlapping Criteria conditions. For instance, one such case may refer to specifying mutually exclusive variants, as described in §6.2.3.

Step 5. Define Long Term Performance Goals

Jacob, as discussed previously, decides to introduce in the global policy three quantitative LTPGs on crosscutting concerns, common for all applications launching CSD:

- The Credit LTPG with monthly horizon, aiming at not exceeding the monthly cost allowance for binding to cloud services: 5 GBP/month in the case study).
- The Data Volume LTPG with monthly horizon, ensuring that the monthly data volume allowance for MBs downloaded for web and cloud services through the MNSP is not exceeded (4000 MBs/month in the case study).
- The Battery Use LTPG on battery power preservation with horizon the periodic battery discharge cycle, aiming to retain by the end of each cycle a user-customizable minimum battery power level ***minBatteryLevel*** for eventual emergency needs (20% at the case study).

Following the LTPG design model of the weight-based technique presented in §6.5, Jacob considers each LTPG as a real or virtual consumable resource over its finite horizon, with one one or more constraints on a monotously increasing or decreasing metric.

The Monthly Credit LTPG is briefly analysed below as an example, while the two other LTPGs are analysed in Appendix F.

At the Policy Design Stage, Step 5 consists in defining each LTPG, the parameters and metrics required for monitoring and decision-making, and its control strategy through adaptation of control variable(s).

1. Monthly Credit Allowance LTPG

Successive Procedural Actions

a) Define the LTPG & Assign it to an Adaptation-Strategy

Goal: *The policy-based Monthly Credit LTPG aims at not exceeding the monthly cost allowance while binding to services on the cloud.* This adaptation logic goal is of primary interest to Evelyn – and any mobile device user with relatively intensive use of cloud service discovery – otherwise she would need to enter frequently her account to check the currently remaining credit value, use the “LowCost” user preference mode to decrease the binding cost, or reduce binding to cloud services to avoid exceeding the credit allowance.

Jacob’s ambition is, through the global policy, to provide the adaptation logic framework for satisfying this LTPG transparently to each application and the user.

This LTPG is evidently assigned to the Cost Optimization adaptation-strategy (Scost), and is considered as a behavioral VP it concerns parameter adaptation of the SR QoS-variable *costQoSReq* (which is also a global QoS-variable).

Cost of Cloud Service – Pay as you Go. The cost of binding to a cloud service is proportional to the cloud resources used (in the case study: the MBs of the video downloaded), but with price per MB possibly different for each different service provisioned at different QoS terms.

b) Define the LTPG Monitoring & Control Strategy

Jacob selects as monitoring and control strategy for the LTPG, the LTPG design model in §6.5:

- Define the required LTPG-related parameters and monitoring metrics
- Define the credit availability “High”, “Low” and “VLow” levels, through threshold parameter values on the monitoring metrics
- Define as control variable the global QoS-variable *costQoSReq*, denoting the maximum price per MB downloaded
- At each of the three credit availability level, define the appropriate actions concerning the *costQoSReq* QoS-variable.

Step 6. Declare the Policy Intermediate Variables

The Volare APSL – as described in chapter 5 – in addition to the context- and adaptation-variables, supports the declaration and use of intermediate variables, constants, metrics, statistical inference parameters (calculation-variables and the subcategory of statistic-calculation-variables on the usage data), as well as the policy-driven auxiliary-variables facilitating decision-making at policy development.

Non-LTPG-related Intermediate Variables

The need for these intermediate variables is policy -specific and it is the natural result of the analysis and design work previously described in Steps 2 to 5. Declaring the intermediate variables is an iterative task of the Policy Design Stage for the policy developer, cyclically around Steps 1 to 6, as the policy design proceeds. See excerpts from the global policy declarations:

Table 7-6 – Global Policy Intermediate Variables Declaration

```
 // Calculation-variables Declarations
// float value E [0, 1], indicating the Sgen SWC
        float CalcVar wSgen == 1.00;
// integer value indicating the total number of days from the beginning of the data history
        integer CalcVar daysNo == StatisticalAnalysisM.daysMonthly.Overallhistory.Count;
// No of months from beginning of history storage, indicating the No of Monthly Periods
        integer CalcVar monthsNo == StatisticalAnalysisM.monthsNo;
// integer value indicating the rediscoveries occurred during current session
        integer CalcVar rediscoveries == StatisticalAnalysisM.rediscoveries;
//  Name of the current day
        string CalcVar dayName == StatisticalAnalysisM.dayName;
// integer value indicating the rediscoveries occurred during current session
        integer CalcVar sessionAdaptations == rediscoveries + 1;
// integer value indicating the current Adaptation No
        integer CalcVar adaptationNo ==
  StatisticalAnalysisM.sessionAdaptations.OverallHistory.Count;
// integer value indicating the current inSessionRecordNo
        integer CalcVar inSessionRecordNo ==
  StatisticalAnalysisM.cloudDurationRecheckcycle.Session.Count;
// integer value indicating the current dischargecycleNo
       integer CalcVar dischargecycleNo ==
  StatisticalAnalysisM.dischargecycle.overallHistory.Count;
// The duration (in seconds) on Cloud Service Discovery (CSD) during the current Session
        float CalcVar cloudDurationSession ==
  StatisticalAnalysisM.cloudDurationRecheckcycle.Session.Sum;
// The Max duration value (in seconds) on CSD in current Dischargecycle period
        float CalcVar cloudDurationDischargecycleMax ==
  StatisticalAnalysisM.cloudDurationDischargecycle.Dischargecycle.Max;
```

```
// The Max time interval duration (in seconds) in every Dischargecycle period
        float CalcVar overallDurationDischargecycleMax ==
  StatisticalAnalysisM.overallDurationDischargecycle.Max;
```

LTPG-related Intermediate Variables

Especially for LTPGs, it will be required to declare parameters and statistic-calculation-variables in order to establish the metrics required for monitoring performance, decision-making and adaptation for achieving the LTPG within horizon duration.

a) Define & Declare the LTPG-related Parameters

The main LTPG-related parameters for each single quantitative goal are:

- **resourceAllowance** denoting the LTPG Max resource value. In this case the term **creditAllowance** is declared as a user-customizable UPI context-variable initially equal to 5 pounds/month:

```
// float, indicating the LTPG value for Monthly cost allowance for cloud services
float ContextVar creditAllowance == UPI.creditAllowance;
```

- **resourceVLowLevel** denoting the LTPG resource very low availability level value. In this case the term **creditVLowLevel** is declared as an implied percentage auxiliary-variable initially equal to **creditInitialVLowLevel,** a user-customizable UPI context-variable with default value 10% of the *creditAllowance*:

```
// float, user-customizable context-variable, 10 % of the creditAllowance
float ContextVar creditInitialVLowLevel == 10 * UPI.creditInitialVLowLevel;
```

b) Declare the Monitoring Metrics

The typical monitoring metrics for each single quantitative goal are: **durationTillNow** and **ltpgDurationRatio,** which are the same for both LTPGs with "Monthly" period, and **resourceUsed, resourceUseratio** and **resourceRefRate,** as defined in §6.5.3:

- *ltpgDurationRatio = 100 x durationTillNow / horizonDuration;*
- *resourceUsed = resource used till now (in resource measurement units);*
- *resourceUseratio = 100 x resourceUsed / resourceAllowance;*
- *resourceRefRate = 100 x resourceUseratio / ltpgDurationRatio.*

For each monthly LTPG (credit & data volume), the metrics: *durationTillNow* and *ltpgDurationRatio* are the same and consequently are declared once only.

The middleware at every active SR on the cloud, starts monitoring context at a *recheckRate* (initially every 30 seconds, except when at user preference mode =

“HighQuality” every 10 seconds) and records current context-variables values including the serviceID bound to and the related QoS provisioning terms (*bitrateQoSProv, costQoSProv and priceMax*) as described in §7.1, and calculates the resulting values at every consecutive Recheckcycle, as follows:

- *cloudDurationRecheckcycle* is the duration in seconds of each Recheckcycle during a CSD session ;
- *cloudMBsRecheckcycle* represents the MBs downloaded during each CSD Recheckcycle time interval;
- *webMBsRecheckcycle* represents the MBs that have been downloaded through the mobile network before the current CSD session – for instance for web services – which also count for the Monthly Data Allowance in addition to the *cloudMBsRecheckcycle*;
- *overallMBsRecheckcycle* represents the total cloud & web MBs downloaded during each CSD Recheckcycle time interval;
- *cloudCostRecheckcycle* represents the binding cost during each CSD Recheckcycle time interval, corresponding typically to the product of the unit price of the bound service per MB downloaded times the MBs: *cloudCostRecheckcycle = cloudMBsRecheckcycle * costQoSProv * priceMax*.

Below is a global policy excerpt with the declaration of the Credit LTPG monitoring metrics.

Table 7-7 – Global Policy Credit LTPG Metrics

```
// The Monthly Credit LTPG Parameters & Monitoring Metrics
// float value, indicating the LTPG value for initial credit LTPG VLowLevel ( = 10%)
percentage CalcVar creditInitialVLowlevel == 10;
//Duration (in seconds) on CSD during the current monitoring Recheckcycle
integer CalcVar cloudDurationRecheckcycle == 86400 * (rowEndTime - rowStartTime);
//MBs of data downloaded on CSD during the current monitoring Recheckcycle
integer CalcVar cloudMBsRecheckcycle == StatisticalAnalysisM.cloudMBsRecheckcycle ;
//The cost of binding on a cloud service during the current monitoring Recheckcycle
float CalcVar cloudCostRecheckcycle == StatisticalAnalysisM.cloudCostRecheckcycle;
//The cost incurred on CSD  in the current Monthly period till now
float CalcVar cloudCostMonthly ==
 StatisticalAnalysisM.cloudCostRecheckcycle.Monthly.Sum;
// integer value indicating the current day number  in the current month
integer CalcVar daysMonthly == StatisticalAnalysisM.daysNo.Monthly.Count;
// integer value indicating the total number of days in the current month
integer CalcVar totalDaysOfMonth == StatisticalAnalysisM.totalDaysOfMonth;
```

```
 // The monthly Duration Ratio, common for both LTPGs
percentage CalcVar monthlyDurationRatio == 100 * ((daysMonthly - 1 + rowStartTime) /
  totalDaysOfMonth);
 // The usage-based Credit Use Ratio over the current Monthly period
percentage CalcVar creditUseratio == 100 * (cloudCostMonthly / creditAllowance);
 // The decision-making float type auxiliary variable, modifying Credit VLowLevel
float AuxiliaryVar creditVLowLevel;
 // It represents the selected date-based or usage-based based cost use rate metric
float AuxiliaryVar creditRefRate;
 // The decision-making float type auxiliary variable: actionCoeff, initially equal to 1.0
float AuxiliaryVar creditActionCoeff;
 // The decision-making string type auxiliary variable for the Credit LTPG Variants
string AuxiliaryVar creditLTPGLevel;
```

7.4.2 Stage 2 – Policy Rules Development Stage

Stage 2 refers to the adaptation-rules development stage, after requirements are identified, goals are set and variables are declared in Stage 1. It concerns the development of adaptation-rules through the Steps 7, 8, 9 & 10 described below.

At this *Stage 2* of adaptation-rules development, Jacob focuses on identifying and specifying Variation Points first of structural or algorithmic character and then of behavioral character, with mutually exclusive alternative variants whose combination defines the valid alternative MESCs, in the successive Steps described below.

Step 7. Policy Structure in Consecutive Execution Cycle Segments

According to the Policy Authoring Methodology of the weight-based technique described in §6.7, Jacob will design the adaptation-rules section of the policy file in the recommended 3 CEC segments policy structure:

1st CEC: The adaptation-rules specifying: (i) Policy Engine middleware configuration-variables, (ii) structural variants and/or algorithm selection, and (iii) “supervisory-layer” algorithmic variant selection.

2nd CEC: The adaptation-rules specifying resource availability level assessment for each of the LTPGs.

3rd CEC: The bulk of the “control-layer” adaptation-rules specifying behavioral (parameter settings) adaptation of the configuration-variables and global QoS variables, under each selectable structural and algorithmic variant of the 1st CEC.

Every CEC segment of a policy file, global or application-level, may include up to one “default” Subpolicy (with Criteria void of predicates, with only the “default” keyword) – thus matched at every context instance, and as many as required Criteria-Subpolicy pairs.

Step 8. Specify Adaptation-Rules on Structural or Algorithmic Adaptation

The 1st CEC segment of the global policy file includes Criteria-Subpolicies composed of:

a. the adaptation-rules specifying Policy Engine parameters required for policy execution, like: *cyclesMax* or *defaultBindingmargin;*

b. the adaptation-rules that specify variant selection on structural (or functional) or algorithmic VPs (for instance specifying the (On or Off) state of active-on-demand sensors or functionalities – like Wi-Fi On or GPS On) or algorithm selection or modification at policy execution.

The following successive actions are implemented, in order to specify the associated adaptation-rules:

a) Develop the Adaptation-Rules on Policy Engine Configuration

Several middleware configuration-variables are also determining the configuration and operation of the Policy Engine for interpreting and executing the composite adaptation logic. Such configuration-variables need to be adapted at the first CEC, so that their adapted values may be known and used at the next CECs by the Policy Engine are the variables: *cyclesMax* and *defaultBindingMargin*. In this way Jacob allows the application policy developer(s) to optionally contribute to Policy Engine configuration.

For instance although the global policy may set a value of 3 for *cyclesMax*, denoting that the Policy Engine will operate policy execution in three CECs, the application policy may introduce an adaptation-rule with higher priority setting *cyclesMax* = 4, if there are dependencies to be considered for the adaptation of application-specific QoS variables at the 4th CEC. Thus, the new resolved-value of *cyclesMax* = 4 may come up from the adaptation.

Excerpt of the global policy

```
Subpolicy G1_DEFAULT{
    defaultBindingMargin = 20 (wSgen * 0.20);
    overridesAsUpperLimit defaultBindingMargin = 50 (wSgen * 1.00);
    yields cyclesMax = 3 (wSgen * 0.20);
 };
```

b) Identify the Structural or Algorithmic Variation Points (SAVPs)

In this case study, there are not alternative structural configurations concerning alternative structural componnents, like GPS On or Off, or alternative communication channels like Wi-Fi On or Off, although they could be introduced.

Instead Jacob, the global policy developer, decides to give emphasis in the use of algorithmic VPs that aim to optionally optimize the adaptation logic at the beginning of each new LTPG horizon episode, by evaluating the results after the end of each LTPG horizon (potentially in relation also to recorded previous LTPG performance results).

Each algorithmic VP consists of two mutually exclusive variants on the selection of alternative metrics or parameter values that modify the LTPG resource availability level assessment algorithm for each LTPG or the adaptation actions corresponding to each availability level. This algorithm modification takes place if the context & adaptation data history is considered adequate for statistic inference (through hypothesis testing predicates) and if the last episode usage data justify the use of it for optimizing the policy.

In the case study, Jacob introduces four VPs of algorithmic character – each with two variants – concerning the battery power, the monthly credit and the data volume LTPGs:

a. The first algorithmic VP concerns the choice of “prefixed” vs. “usage-based” metrics (statistically-inferred through the recorded usage data and representing better the usage model) for each of the three LTPG resource availability assessment algorithms.

b. Every one of the other three algorithmic VPs, corresponding to each of the three LTPGs, specifies alternative threshold parameters values that modify two algorithms by adapting threshold parameters values. If at the last episode (LTPG horizon) the LTPG allowance value has been exceeded (goal failure), then by increasing the **VLowLevel** parameter value, the LTPG resource availability level assessment algorithm is modified with domain of the “High” availability level being reduced – since it is defined as: 100 – **VlowLevel**, and the domains for “Low” or “VLow” levels being enlarged, thus switching adaptation to more restrictive actions at the new LTPG horizon. Secondly, specifying the increase of the **actionCoeff** parameter from its default value, the LTPG-related adaptation algorithm is modified leading towards more intense adaptation actions.

Thus at the first CEC of the global policy, Jacob defines four algorithmic VPs, each with 2 mutually exclusive variants, raising the number of alternative variants to: 2 x 2 x 2 x 2 = 16 algorithmic variants.

c) Develop Adaptation-Rules on each Structural or Algorithmic VP

As described in Such mutually exclusive structural or algorithmic variants that influence the required adaptation are in the case study the three VPs for usage-based vs. prefixed metrics and threshold parameters: (i) for the battery power LTPG within the resource optimization adaptation-strategy, (ii) for the monthly credit allowance LTPG within the cost optimization adaptation-strategy, (iii) for the data volume allowance LTPG again within the cost optimization adaptation-strategy.

Note that all rules on the same adaptation-variable should be assigned the same cycle declaration.

Table 7-8 - Case study algorithmic VPs at the 1st CEC

```
// 1st CEC: Setting default values to configuration-variables

   [1] Criteria G1_DEFAULT{
           default;
   };
   // 1st CEC:  "PREFIXED or UG-BASED CRITERIA FOR METRICS"
   [1] Criteria G1_BATTERY_USGBASED{
           daysNo > 30;
           monthsNo > 1;
           minBatteryLevel > batteryDischargecycleMin;
           allowOptimization = "Y";
   };
   // 1st CEC:  "PREFIXED or USG-BASED CRITERIA FOR CREDIT LTPG MONITORING"
   [1] Criteria G1_CREDIT_USGBASED{
           daysNo > 30;
           monthsNo > 1;
           creditUseratioMax > 100;
           allowOptimization = "Y";
    };
   // 1st CEC:  "PREFIXED or USG-BASED CRITERIA FOR DATA VOLUME LTPG MONITORING"
   [1] Criteria G1_DATA_USGBASED{
           daysNo > 30;
           monthsNo > 1;
           dataUseratioMax > 100;
           allowOptimization = "Y";
   };

  Subpolicy G1_DEFAULT{
       batteryVLowLevel = minBatteryLevel (wSperf * 0.20);
```

```
        creditVLowLevel = creditDataInitialVLowlevel (wScost * 0.20);
        dataVLowLevel = creditDataInitialVLowlevel (wScost * 0.20);
        creditActionCoeff = 1 (wScost * 0.20);
        dataActionCoeff = 1 (wScost * 0.20);
    };
    Subpolicy G1_BATTERY_USGBASED{
        batteryVLowLevel = minBatteryLevel + 10 (wSres * 1.00);
    };
    Subpolicy G1_CREDIT_USGBASED{
        creditVLowLevel = creditDataInitialVLowlevel + 10 (wScost* 1.00);
        creditActionCoeff = 110/100 (wScost * 0.20);
    };
    Subpolicy G1_DATA_USGBASED{
        dataVLowLevel = creditDataInitialVLowlevel + 10 (wScost * 1.00);
        creditActionCoeff = 110/100 (wScost * 0.20);
    };
```

Table 7-9 – Algorithmic SAVPs in the global policy

ALGORITHMIC VPs	CEC	Subpolicies Concerned	Algorithmic Variants
Prefixed vs. Usage-based LTPG Metrics	1st	G1_DEFAULT G1_METRICS_USGBASED	2
Battery LTPG Parameters Adaptation	1st	G1_DEFAULT G1_BATTERY_USGBASED	2
Credit LTPG Parameters Adaptation	1st	G1_DEFAULT G1_CREDIT_USGBASED	2
Data LTPG Parameters Adaptation	1st	G1_DEFAULT G1_DATA_USGBASED	2

Step 9. Specify Adaptation-Rules on Behavioral VPs

At policy execution, once the SAV for each active SAVP has been selected at the first CEC, their parameter settings under the current context have to be defined. The adaptation-rules on behavioral adaptation (parameter settings) should be grouped in typically three sets: (i) SAV-exclusive adaptation-rules, (ii) adaptation-rules common only in several SAVs, and (iii) adaptation-rules common to all SAVs.

Consistency Directives at Specification of Behavioral Adaptation

Applying the consistency directives discussed at §6.1.4, two precautions are taken by the policy developer to prevent matching, selection and execution of behavioral adaptation rules belonging to non-selected SAVs:

a. The behavioral (parameter) adaptation rules should be specified with a cycle declaration subsequent to the CEC at which the adaptation-rules specifying SAVs selection are assigned. In this way at policy execution, the already chosen at the first CEC SAVs are known.

b. SAV-specific adaptation-rules on parameter adaptation should have as predicate in their Criteria conditions a reference to the supported SAV(s), in order to ensure only compatible rules selection.

The following successive actions are implemented according to PAM:

a) Identify the Behavioral VPs and their BVs

Identify the behavioral VPs on LTPGs or on short-goals and their mutually exclusive behavioral variants (BVs) along with their context sib-domains.

Assign every behavioral VP to an adaptation-strategy.

Table 7-10 – Behavioral VPs in the global policy

BEHAVIORAL VPs	CEC	Subpolicies Concerned	Behavioral Variants
Battery LTPG Level Assessment BVP	2nd	G2_BATTERYLTPG_HIGH G2_BATTERYLTPG_LOW G2_BATTERYLTPG_VLOW	3
Monthly Credit LTPG Level Assessment BVP	2nd	G2_CREDITLTPG_HIGH G2_CREDITLTPG_LOW G2_CREDITLTPG_VLOW	3
Monthly Data LTPG Level Assessment BVP	2nd	G2_DATALTPG_HIGH G2_DATALTPG_LOW G2_DATALTPG_VLOW	3
User Preference Modes BVP through the User Preferences Model	3rd	G3_NORMAL G3_HIGHQUALITY G3_LOWCOST G3_SAVEBATTERY	4
Monthly Credit LTPG Adaptation Actions BVP	3rd	G3_CREDITLTPG_HIGH G3_CREDITLTPG_LOW G3_CR4EDITLTPG_VLOW	3

b) Specify the Parameter Adaptation Rules for each behavioral VP

Within every adaptation-strategy and for each SAVP, specify over the whole context sub-domain of each SAV, the behavioral adaptation rules for parameter settings adaptation of the associated configuration- and/or QoS- and/or eventual auxiliary-variables.

Adaptation-rules should have as predicate in their Criteria conditions a reference to the SAV(s) they are compatible (or inversely excluding incompatible SAV(s)) with, in order to avoid incompatible rules selection. In the case study for each variant-specific adaptation-rules, it is sufficient to define in their predicates the relevant SAV.

Within every adaptation-strategy, under eavh SAV and for each behavioral VP, specify over the whole context sub-domain of each BV, the behavioral adaptation rules that express adaptation on the associated configuration- and/or QoS- and/or eventual auxiliary-variables.

Rules on parameter adaptation settings with common Criteria in an adaptation-strategy are grouped together as Criteria-Subpolicy pairs and constitute the alternative BVs of each BVP of the adaptation-strategy. Typically, some behavioral Criteria-Subpolicy pairs may be valid for all SAVs, other Subpolicies are SAV-specific and some are valid for only several SAVs.

c) Specify the Parameter Adaptation Rules for each LTPG

Specifically for LTPGs – which are considered as behavioral VPs – the following steps are applied, which are described here for the Monthly Credit LTPG:

a. Set Credit LTPG Performance Assessment Levels

Jacob implements the weight-based methodology LTPG control strategy, consisting in classifying credit LTPG availability at any context instance as “High” or “Low” or “VLow”, so that appropriate actions can later be specified.

As described in §6.5.3, Jacob declares a threshold parameter that will be pivotal for LTPG management: **creditVLowLevel** = remaining percentage of the resource monthly allowance that switches to emergency adaptation (initial value = 10%). Through it, Jacob will later specify the rules that assess LTPG resource availability as “High”, “Low” or “VLow”. Jacob establishes three credit availability levels: *High*, *Low* and *VLow*, by declaring in the policy the required auxiliary-variable *creditLTPGVariant*:

```
string AuxiliaryVar creditLTPGVariant;
```

taking values: “HighCredit” or “LowCredit” or “VLowCredit”.

The credit availability levels “High”, “Low” or “VLow” are specified by Jacob through adaptation-rules in the 2nd CEC of the policy, as indicated in the mutually

};

exclusive Criteria-Subpolicies below, where the “default” Criteria-Subpolicy represents the default value.

Table 7-11 – Credit LTPG Resource Availability Level Assessment

```
 // 2nd CEC: Setting the default value to the creditLTPGVariant
 [2] Criteria G2_DEFAULT{
           default;
 };
 Subpolicy G2_DEFAULT{
      creditLTPGVariant = "LowCredit" (wScost*0.10);
 };
 // 2nd CEC: Define the “HighCredit” availability level
// Allow up to 20% violation on creditRefRate at up to 50% of total allowance
// or creditRefRate <= 100 And creditUseratio >= creditLTPGVLowLevel
 [2] Criteria G2_CREDITLTPG_HIGH{
          creditUseratio <= 100 – 50
          creditRefRate <= 120;
          Or
          creditUseratio <= 100 - creditEmergencyLevel;
          creditRefRate <= 100;
          Or
          userPref = "HighQuality";
 };
 // 2nd CEC: At High Credit level, costQoSReq = 100
 Subpolicy G2_CREDITLTPG_HIGH{
      creditLTPGVariant = "HighCredit" (wScost*0.80);
 };
 // 2nd CEC: At Very Low Credit level, reduced values are set for costQoSReq
// Set “VLowCredit” when: creditUseratio > 100 And creditRefRate > 100
 [2] Criteria G2_CREDITLTPG_VLOW{
          creditRefRate > 100;
          creditUseratio > 100 – creditLTPGVLowLevel;
          userPref <> "HighQuality";
 };
 Subpolicy G2_CREDITLTPG_VLOW{
      creditLTPGVariant = "VLowCredit" (wScost*0.80);
 };
```

As it is evident, if the above two Criteria: G2_CREDITLTPG_HIGH or G2_CREDITLTPG_VLOW – with conditions defined on the monitoring metrics – are not satisfied by the current context, then the default value “LowCredit” will be selected in the G2_DEFAULT Subpolicy.

b. **Select the control variable(s)**

Control variable for the Credit LTPG in the global policy is the global QoS-variable *costQoSReq* representing a tentative maximum service request value for price per MB from the sought cloud service. At the same time it implements a second more important task: as costQoSReq is reduced, the services offered for the same video content are typically of lower quality and consequently of reduced encoding bitrate. For instance instead of an “optimal” quality at video encoding bitrate = 256 KB/s, the service corresponding at video encoding bitrate = 128 KB/s will have half the MBs, while at 56 KB/s less than a quarter of the initial video file size in MBs, consequently reducing several times the implied cost of service consumption (binding cost).

The cost of binding at any CSD session depends on the MBs downloaded x price per MB for the provisioned service during the duration of each adaptation. Note that a CSD session may include more than one successive binding to different services of the same content at different QoS levels, because of dynamic context-aware adaptation due to significant context change and service rediscovery.

c. Develop the Adaptation-Rules at each Resource Availability Level

Develop adaptation-rules to suitably adapt the values of the control variables at each resource availability level.

For managing this LTPG Variation Point, Jacob defines at the 3rd CEC three Criteria-Subpolicy pairs for each of the three availability levels: G3_CREDITLTPG_HIGH when *creditLTPGVariant = “HighCredit”,*

G3_CREDITLTPG_LOW when *creditLTPGVariant = “LowCredit”,*

G3_CREDITLTPG_ VLOW, when *creditLTPGVariant = “VLowCredit”*.

Switches from “High” to “Low” or “VLow” credit level situation or the inverse are possible within each monthly period, if the metrics values improve or deteriorate depending on the extent of use of cloud services, restricting or allowing respectively higher QoS terms at CSD. At “HighCredit” instances maximum *costQoSReq* attainable values (100% of the maximum threshold *priceMax)* are allowed. At “LowCredit” instances, reduced *costQoSReq* values are imposed, higher as the deviation of the *creditRefRate* metric value from the “correct refernce rate” (100). At “VLowCredit” minimal costQoSReq values are specified.

Since the control variable *costQoSReq* specifying a maximum value for the cloud service per MB is a global QoS-variable, the whole LTPG management is implemented by the global policy specifying the *costQoSReq* value, as designed. However, the application policy may also have adaptation-rules on *costQoSReq* as well. Below is a policy segment with the global policy adaptation action rules, concerning the Credit LTPG Adaptation Actions algorithm:

Table 7-12 – Credit LTPG Adaptation Actions Algorithm

```
// 3rd CEC: At HighCredit, allow max costQoSReq
[3] Criteria G3_CREDITLTPG_HIGH{
        creditLTPGLevel = "HighCredit";
};
// 3rd CEC: At LowCredit, reduce costQoSReq
[3] Criteria G3_CREDITLTPG_LOW{
        creditLTPGLevel = "LowCredit";
};
// 3rd CEC: At VLowCredit, reduce costQoSReq further
[3] Criteria G3_CREDITLTPG_VLOW{
        creditLTPGLevel = "VLowCredit";
};
 Subpolicy G3_CREDITLTPG_HIGH{
    costQoSReq = costQoSReq [-20] (wScost*1.00);
};
Subpolicy G3_CREDITLTPG_LOW{
    overridesAsUpperLimit costQoSReq = 0.90 * costQoSReq (wScost * 0.50);
    overridesAsLowerLimit costQoSReq = 0.70 * costQoSReq (wScost * 0.50);
    costQoSReq = costQoSReq - costQoSReq * creditActionCoeff * (1 + creditRefRate/100) *
(1 + creditUseratio/100) * (1 + monthlyDurationRatio/100) / 6 [-12] (wScost*1.00);
};
Subpolicy G3_CREDITLTPG_VLOW{
    overridesAsUpperLimit costQoSReq = 0.70 * costQoSReq (wScost * 0.50);
    costQoSReq = costQoSReq - costQoSReq * creditActionCoeff * (1 + creditRefRate/100) *
(1 + creditUseratio/100) * (1 + monthlyDurationRatio/100) / 6 [-12] (wScost*1.00);
};
```

Step 10. Review the Policy

The following successive actions at policy development for reviewing policy development are implemented at Step 10:

a) Review the policy until all goals and requirements are satisfied

Jacob reviews the policy statements to make sure that all defined and used variables are correctly declared, that all requirements and goals and all SAVPs and BVPs are expressed by adaptation-rules, and every rule has a weight function with SWC corresponding to the related adaptation-strategy.

b) Restructure the adaptation-rules in Criteria – Subpolicy pairs

Jacob groups together the adaptation-rules with common cycle-declaration (CEC) and common Criteria, restructuring them in Criteria-Subpolicies pairs.

7.4.3 Stage 3 – Policy Testing & Verification Stage

Stage 3: Test & Verify and Validate the Policy

This is the final stage of the Policy Authoring Methodology, that is also considered interactively with the previous stages. It concerns Step 11: Test & Verify the Policy, and Step 12: Validate the Policy.

Step 11. Test & Verify the Policy

As described in §6.6, the following two successive actions are implemented.

a) Test the Policy

Jacob runs the global policy file on the simulated policy execution & verification application (PEVApp) for Volare-compatible policy files that is provided by the weight-based methodology, first for identifying eventual syntax errors. Once these errors are corrected, the offline automated test suite generation is activated and the simulated policy execution on the test suite(s) generated. Failures are identified and corrected.

b) Verify the Policy

The simulated policy execution application (PEVapp) allows evaluation of real or test suite-derived simulated context and adaptation data through automated charts and calculation tables. For testing the global policy a very simple application policy may be used, since the global policy may be self-supported for initial testing.

Jacob verifies offline the expected adaptation behavior, through automated charts on simulated execution results over generated test suites, for policy evaluation and necessary corrections.

On the derived simulated results, the PEVApp rules fault detection algorithms may be applied for identifying eventual dead rules, dead predicates etc. A visual tool may also assist for comparative “Metamorphic Relations” evaluation, i.e.

relations that the adaptation results should comply to, on given context sets relations, For instance the adaptation results can be verified through automated charts for each certain adaptation-variable results under different user preference mode, expecting decreasing values, moving from “HighQuality” to “Normal” and to “LowCost” or “SaveBattery” mode under the same other context data.

Step 12. Validate the Policy

After being tested and verified, Jacob validates the policy file by comparison of real or simulated results to expected results.

7.5 PAM – Authoring an Application Policy

For a mobile application launching service discovery for a cloud service, in order to be subscribed for SR DCAA by the Volare middleware, it is required to download and store in the Volare Policy Files directory of the device an application policy for the adaptation of the service request QoS-variables, and eventually for influencing adaptation on the middleware operation configuration-variables.

For the application policy design and implementation, reference is made again to the three Policy Development Stages and the 12 Steps of the Policy Authoring Methodology described in §6.7.

Ronald, the application policy developer, should be aware of the Volare weight-based methodology, as well as of the general orientation of the middleware global policy or the family of global policies on different products, the parameters used and supported and the goals set, since it concerns the same application for several types or families of mobile devices on the same platform.

Remark: *The global policy developer has the possibility to specify all rules in the highest rule priority level, thus forbidding intervention of the application policy. However, it is suggested that this may take place for only the most critical operations, and allow the application policy developer access to the adaptation of the middleware configuration-variables.*

The application policy is expected to conform to the following generic issues as established by the global policy:

a) Conform to the User Preferences Model, as it is configured in the global policy for each user preference mode and specify the adaptation-rules expressing each user preference-indicated adaptation behavior. Note that the application policy developer has no access to the configuration of the UPI and the default values of the User Choices Profile.

b) Use the Weight Assigning Strategy and the adaptation-concerns & adaptation-strategies defined at the global policy.

c) Conform to the quantitative Long Term Performance Goals set by the global policy for all applications launching cloud service discovery, and specify the application-specific adaptation-rules in relevance to LTPGs, with the same behavior in mind. It is only required to use in the application policy the same performance assessments derived by the global policy adaptation-rules concerning every LTPG.

d) Develop the application policy considering that it is expected to operate within the framework of the two-level Composite Policy. Assign to every adaptation-rule: (i) priority level for intra- and inter-policy conflict resolution, (ii) a weight function value demonstrating its relative importance at policy execution, according to the WAS format, (iii) a cycle declaration (CEC) corresponding to the adaptation-variable concerned.

The whole application policy is attached with brief comments in Appendix B of the Thesis.

7.5.1 Stage 1 – Analyze System - Identify Requirements & Goals

This is the design stage of the application policy and the developer has to define the requirements and short term goals, eventual new LTPGs, and how to cope with established User Preferences Model and generic LTPGs. Steps 2 to 4 are integrated in the global policy.

Step 1. Define the Context & Adaptation Profile

All context- or configuration- or global QoS-variables are predetermined at the global policy file, as well as most of the calculation- and auxiliary-variables.

Ronald only needs to declare the SR application-specific QoS-variables *bitrateQoSReq* (video encoding bitrate) and *fpsQoSReq* (frames per second), in addition to the global ones (*costQoSReq, availability, reliability and throughput*), already declared in the global policy.

Additionally on the intermediate variables, Ronald, having analyzed the requirements and goals of the application policy, needs only to declare any application-specific intermediate variables, namely any metrics that are required for the application policy authoring.

Table 7-13 – Application-specific SR QoS-variables

QoS-Variable ID	Represented Quantity	TypeID / Units	Typical Value

bitrateQoSReq	The SR max video encoding bitrate requested	Integer / in KB/s	256 KB/s
fpsQoSReq	The SR min requested frames per second	percentage / % of Max unadapted value	Unadapted Max value = 30 fps

The developer needs only declare the application-specific service request QoS-variables and any application-specific calculation- or auxiliary-variables or service request related context-variable, that may be required for the adaptation-rules. The developer may directly reference in the adaptation-rules all declared variables in the global policy without re-declaring them.

If a service request QoS-variable is also a global QoS-variable under a different name, then a declaration-statement of equivalence is required, assigning the application variable to the respective global variable.

Table 7-14 - Application Policy Declarations

```
// App Policy VSTREAM - context-variables Declarations
 //  The corresponding to the bound to serviceID bitrateQoS on current CSD
 integer ContextVar bitrateQoSProv == ContextMonitoringM.bitrateQoSProv;
// End of context-variables declarations of the Application Policy VSTREAM
 // QoS Variables Declarations
 // QoS Variables that are common with the global QoS of the are not declared again
 // The following QoS Variables are application-specific QoS variables
 // The Service Request bitrate QoS value for the service requested on the Cloud
float QoSVar bitrateQoSReq == ServiceRequest.activeRequest.bitrateQoSReq;
 // The Service Request "frames per second" QoS value for the service requested on the Cloud (typically: 23 to 2
integer QoSVar fpsQoSReq == ServiceRequest.activeRequest.fpsQoSReq;
// End of QoS-variables declarations of the Application Policy VSTREAM
```

Steps 2, 3 & 4

Step 2: Defining the adaptation-concerns and adaptation-strategies, **Step 3**: Defining the Weight Assigning Strategy and setting the SWC values and **Step 4**: Configuring the User Preferences Model, are completely the tasks of the global policy. The application policy developer has only to adopt and conform to the global policy choices in these matters.

Step 5: Introducing LTPGs in the Application Policy

Similarly, for the common LTPGs for all applications, the global policy should have declared all the required monitoring and decision-making metrics and the

LTPG resource availability assessment Criteria-Subpolicies for each context instance at the 2nd CEC part of the policy file. Consequently the task of the application developer is mainly focused, based on the LTPG performance assessments specified in the global policy, on specifying through adaptation-rules the required adaptation actions on the application-specific control QoS-variables at the next CEC part of the application policy – in this case at the 3rd CEC.

Ronald has the task to specify adaptation-rules conforming to the established behavioural VPs concerning the battery power use LTPG and the monthly data volume LTPG, for which the application-specific QoS-variable **bitrateQoSReq** is the most appropriate control variable.

a) Battery LTPG Adaptation Control Strategy

The Resource (battery power) Use LTPG of the global policy concerns battery power preservation with horizon the periodic discharge cycle and target to retain by the end of each cycle a percentage of battery for emergency needs. Already through adaptation-rules of the 2nd CEC part of the global policy, the LTPG availability levels are selected.

Ronald uses as control variable the bitrate of the SR *bitrateQoSReq* for battery power management. Reducing the video encoding bitrate, for instance instead of a nominal video encoding bitrate of 256 KB/s – request a 128 KB/s (or 56 or 28 KB/s) video, will reduce significantly both the device power use from CPU, storage, playback and data reception through the cellular network.

Consequently, at the Low and VLow availability levels, gradual restrictions on the allowed bitrateQoSReq value will be imposed, in increasing relation to values of the LTPG monitoring metric batteryRefRate and the criticality level metrics: batteryUseratio and monthlyDuratioRatio.

b) Data Volume LTPG Adaptation Control Strategy

The Data Volume LTPG with monthly horizon, aims at ensuring that the monthly data volume allowance for MBs downloaded from the Web or the Cloud is not exceeded, which cannot be served by rules in the global policy, as no such global variables are declared. Consequently, this LTPG is defined in the global policy, and the adaptation-rules concerning the availability level are included in the 2nd CEC. Every application policy should design rules to serve this goal on the appropriate QoS-variable.

By reducing at the Low or VLow data volume availability levels the SR bitrate QoS Request value – although leading to a lower quality video – the MBs to be downloaded may be reduced by several times and consequently the data volume may be under control. Ronald selects as more appropriate control variable the *bitrateQoSReq*, imposing at the Low and VLow availability levels gradual restrictions, in increasing relation to values of the LTPG monitoring metric *batteryRefRate* and the criticality level metrics: *batteryUseratio* and *monthlyDuratioRatio*.

Step 5. Declare Intermediate Variables

Ronald declares the following application-specific intermediate-variables that are required for monitoring, like eventually repeated rediscoveries during a CSD session (rediscovPer5min) in order to take measures to reduce it.

```
Declarations{
    // Calculation-variables Declarations
    // The current session rediscoveries expressed per 5 min eq. Session on current CSD
   float CalcVar rediscovPer5min == rediscoveries * 300 / (cloudDurationSession + 1);
```

7.5.2 Stage 2 – Policy Rules Development Stage

Step 7: Defining the CEC segments of the Policy Rules Section

No structural or algorithmic variants need to be defined at the application policy. Consequently, no 1st CEC adaptation-rules are required.

Similarly, no application-specific LTPG is defined, only short-term adaptation goals and the global policy battery and data volume LTPGs are relevant. Consequently no 2nd CEC adaptation-rules need to be specified for the application policy.

Only at the 3rd CEC the adaptation-rules for the adaptation of the QoS-variables will be specified, based on the context-context-variables, metrics and intermediate variables already declared in the global policy.

Step 8. Specify Rules on Structural/Algorithmic Adaptation

No structural or algorithmic or operation mode specifying adaptation-rules are application-specific. Consequently no rules are designed for the 1st CEC part of the policy.

Step 9. Specify Rules on Behavioral Adaptation

At Stage 3 Ronald’s task consists in building the adaptation-rules for the behavioral variants of interest to the application SR, either common behavioral variants established by the global policy behavioral VPs or application-specific ones.

All adaptation-rules specifying parameter settings for the SR QoS-variables or eventually the middleware configuration-variables will be included in the 3rd CEC part of the Policy, along with the similar adaptation-rules of the global policy.

Table 7-15 – Application Policy Parameter Adaptation Excerpt

```
// 3rd CEC: At Low Data Budget, low attainable values are set for QoSvar: bitrateQoSReq
[3] Criteria VSTREAM3_DATALTPG_LOW{
        dataLTPGVariant = "LowData";
};
Subpolicy VSTREAM3_DATALTPG_LOW{
    overridesAsUpperLimit bitrateQoSReq = 0.40 * bitrateQoSReq [-20] (wScost*1.00);
    bitrateQoSReq = 0.20*bitrateQoSReq + 0.20 * bitrateQoSReq * (100 / (dataRefRate + 100))
    [-12] (wScost*1.00);
    fpsQoSReq = 0.85*fpsQoSReq [10] (wScost*1.00);
```

Ronald has to specify adaptation-rules for the application-specific QoS-variables for each behavioral variant (BV) of the following behavioral VPs:

a) The Alternative User Preference Modes BVP

Four mutually exclusive Criteria-Subpolicy pairs are built, specifying settings for QoS-variables related to each BV of the User Preferences Model VP, namely the four user preference modes “HighQuality”, “Normal”, “LowCost” and “SaveBattery” within the Sperf adaptation-strategy.

b) The Battery LTPG BVP over every Discharge Cycle

Three mutually exclusive Criteria-Subpolicy pairs are built, one for each BV of the Battery LTPG VP, namely for High, Low and VLow availability levels, within the Sres adaptation-strategy.

c) The Monthly Data Volume LTPG BVP

Three mutually exclusive Criteria-Subpolicy pairs for each BV of the Data Volume LTPG VP, namely on High, Low and VLow availability level, within the Scost adaptation-strategy, with adaptation-rules on QoS-variables related to the monthly data volume LTPG, namely the *bitrateQoSReq*.

d) The Monthly Credit LTPG BVP

This LTPG is fully covered in the global policy and Ronald does not specify additional adaptation-rules on it.

e) The Runtime Disruption Minimization BVP

Ronald introduces an application-specific VP within the Disruption Minimization adaptation-strategy, aiming at minimizing disruption at runtime by rediscoveries due to context or QoS variation. Although every runtime adaptation aims to best adapt the service discovery & binding to the current context, there is a penalty for the user (delay, change of performance characteristics, etc.) and one of the goals is to avoid rediscoveries that offer less to user satisfaction than the disruption cost that may be incurred, especially for a video-streaming application.

This session-level goal aims at retaining within each session period the metric: ***rediscPer5min*** *below a policy -specified value.*

Ronald uses two options to minimize rediscoveries at runtime, first by increasing the *rediscQoSThreshold* and secondly by assigning the “NoAdapt” option to the middleware configuration-variable *preferredVariant,* thus specifying adaptation rejection at runtime, if no important context change imposes it.

Ronald establishes through appropriate Criteria conditions two disruption levels: “LowDisruption” at which no adaptation action is specified (configuration-variables set at the default values in the default Subpolicy), and “HighDisruption” at which the configuration-variables are modified, through the VSTREAM3_HIGHDISRUPTION and the VSTREAM3_DEFAULT Criteria-Subpolicy pairs.

Table 7-16 – Disruption Minimization BVP

```
    // 3rd CEC At High Disruption condition, max values set for rediscQoSThreshold or
rediscContextThreshold

   [3] Criteria VSTREAM3_HIGHDISRUPTION{

         rediscovPer5min > 1;

         inSessionRecordNo > 1;

         bandwidth > 1.25 * bitrateQoSProvLast;

         userPref <> "HighQuality";

   };

    Subpolicy VSTREAM3_DEFAULT{

         preferredVariant = "Adapt" (wSdisr * 1.00);

    };

   Subpolicy VSTREAM3_HIGHDISRUPTION{

       overridesAll rediscQoSThreshold = 10 (wSdisr*1.00);

       preferredVariant = "NoAdapt" (wSdisr*1.00);

   };
```

Table 7-17 – Behavioral VPs in the Application Policy

BEHAVIORAL VPs	CEC	Subpolicies Concerned	Behavioral Variants
User Preference Modes BVP through the User Preferences Model	3rd	VSTREAM3_NORMAL VSTREAM3_HIGHQUALITY VSTREAM3_LOWCOST VSTREAM3_SAVEBATTERY	4
Battery LTPG Adaptation Actions BVP	3rd	VSTREAM3_BATTERYLTPG_HIGH G3_BATTERYLTPG_LOW VSTREAM3_BATTERYLTPG_VLOW	3
Monthly Data Volume LTPG Adaptation Actions BVP	3rd	VSTREAM3_DATALTPG_HIGH VSTREAM3_DATALTPG_LOW VSTREAM3_DATALTPG_VLOW	3
Runtime Disruption BVP	3rd	VSTREAM3_DEFAULT VSTREAM3_DISRUPTION_HIGH	2

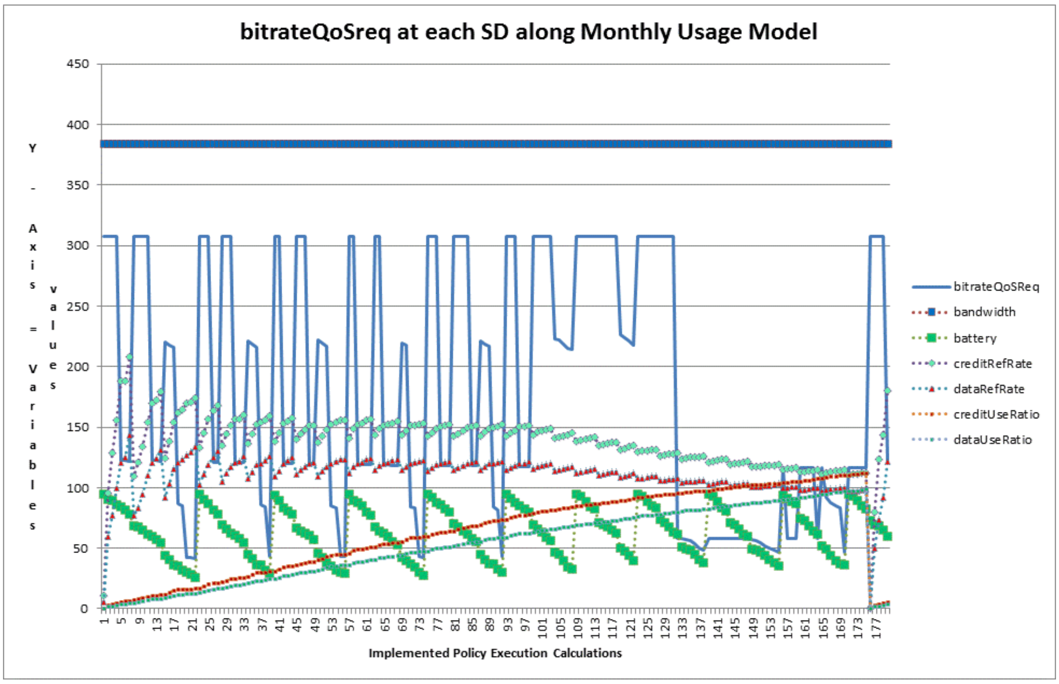
7.5.3 Stage 3 – Policy Testing & Verification Stage

The **Steps 11 & 12** of **Stage 3** of the Policy Authoring Methodology are of importance also at application policy authoring. Since the adaptation logic is composed of the already existing global policy and the application policy, once the application policy file is built and reviewed, the PEVapp offline simulation tools may be used for:

a. Testing the syntax correctness of the policy
b. Creating automatically test suites generated on developer-selected available options on policy predicate coverage and domain testing strategies
c. Activating offline repeated Composite Policy execution on the test suite(s) to test for failures and verification of results
d. Evaluating the simulated dynamic results for detecting rules faults like dead predicates, dead adaptation-statements, etc.
e. Through semi-automated generation of charts on the test suites execution results, drawing conclusions on the effectiveness of the adaptation.

Ronald uses the PEVApp tool on all above.

Figure 7-3 – Policy execution Results across a Monthly Usage Model



7.6 Related Work

The issue of efficient energy management for a mobile device is a most critical one for mobile equipment and especially one of the most constrained types: smartphones. An analytic description of power consumption in a smartphone per activity and component system is given in NICTA [92] and [94]. Different approaches for saving energy in mobiles are been researched and applied.

One trend, with basic example DYNAMO [93], implements a cross-layer active energy saving strategy by modulating CPU voltage and backlight intensity and similarly does AURA [91] with management strategies using MDP techniques. In [97] the User talk time is taken as an explicit variable and with MDP techniques power mgmt is implemented to make the battery last as requested.

Another – mostly MCC-oriented – trend consists in offloading part of computationally intensive tasks on the Web or the Cloud and is extensively researched with works like CloneCloud [27], MAUI [96], CasCap [67] etc, with always the question which application tasks may be offloaded at a positive energy surplus. Video-streaming without transcoding is clearly not a candidate.

A third trend makes use of prefetching-friendly and/or delay-tolerant applications, eventually executing cellular or Wi-Fi communication when the signal is strong and with appropriate packet modulation for energy savings, like: Stratus [29], ENVI [89], eTime [90] etc. Dropbox [53] saves energy of mobile devices by storing and synchronizing mobile data on Amazon S3 storage system.

Volare is tasked with Cloud Service Discovery (CSD) and implements energy Mgmt during CSD through application-level DCAA. It adapts the active application Service Request as well as the middleware configuration variables, not simply according to the current energy level but considering the power level within the current battery discharge cycle and the expected remaining battery use. It evaluates power usage with respect to the estimated remaining time interval till recharging and based on this evaluation - not only at an initial SD but even at runtime - it may impose adaptation to a more appropriate service with lower quality and energy needs.

8 Evaluation of the Project

In the “Introduction” section of this Thesis where the Research Hypothesis is defined, a clear reference is made to the goals of this research project, presenting the Volare approach for assigning through a mobile adaptive middleware policy-based DCAA to the SR of mobile applications launching cloud service discovery (CSD). The scope of the Volare approach includes especially the more demanding “long-lasting” mobile applications on service discovery (like media-streaming, tele-conferencing, navigation applications, etc.) that require dynamic adaptation capability.

The Volare approach is composed of the following three constituents:

- **the mobile adaptive middleware, with a User Preferences Interface application, to be installed on the mobile device**
- **the Volare Adaptation Policy Specification Language (APSL),** and
- **the Weight-based (WB) Adaptation Reasoning Technique & Methodology**

with the two last constituents supporting the development of the adaptation logic.

The Volare approach incorporates several novel characteristics concerning the authoring of external adaptation logic and the middleware-based adaptation, enabled through specific features of the aforementioned three constituents.

Qualitative Evaluation of the Volare Components

Examining the three constituents of the Volare approach, and comparing their characteristics with the corresponding requirements in §3.3, §3.4, §3.5, all three constituents have covered the requirements set.

The middleware only had missing modules – in comparison to the conceptual model speciofied in §3.4 and described in chapter 4, namely the Statistical Analysis Module, which was not required in the policy since no LTPGs were introduced in the policy for the usage data. Similarly the UPI, described in §4.2 and §6.4, was imitated by changes in the policy, at any new session user preference change was required.

As a consequence, the qualitative evaluation has identified that all requirements set were respected at the runs executed.

Contributions

As described in the “Introduction” of this Thesis (chapter 1) and the Project Overview (chapter 3), the Volare approach is characterized by the following main contributions:

1. **Dynamic context-aware middleware-based adaptation at CSD sessions of the SR of unanticipated mobile applications not custom-made to the middleware and transparently to them**
2. **Two-level policy structure of the adaptation logic**
3. **Integration of a configurable dynamic User Preferences Model in the adaptation logic through a User Preferences Interface application (UPI)**
4. **Policy authoring allowing multiple rules, each with a weight value, to be selectable at policy execution on the same adaptation-variable**
5. **Defining major adaptation-concerns and design the adaptation logic in overlapping sub-policies (adaptation-strategies), each serving an adaptation-concern over the valid context domain**
6. **Introduction of quantitative LTPGs in the adaptation logic over finite horizons exceeding many times a CSD session**
7. **Policy rules distributed in multiple consecutive execution cycles (CECs)**
8. **Policy authoring with high behavioral variability, by-passing the combinatorial explosion**

Although some of the above contributions may have been used in some way individually by other research papers for DCAA of mobile applications, Volare claims the unique combination of all these features that are introduced by the APSL and the weight-based adaptation reasoning approach and are supported by the middleware, leading to benefits that will be evaluated in the following subsections.

Evaluation Research Questions

The following Research Questions are raised, to assist in the evaluation of this project, each corresponding to one of the contributions of the Thesis:

RQ1: Can adapting the QoS levels of a service request (SR) launched by a mobile application before service discovery and binding, based on the device current context and policy requirements and goals, lead to discovery of more fitting services and to more efficient use of device resources?

RQ2: Does enabling mobile applications SRs adaptation that takes into account both the device’s capabilities and application’s requirements, by utilizing two-level *policy architecture, provide advantages in policy authoring while reducing policy complexity?*

RQ3: Can we enable the user to influence the QoS levels adaptation of the SRs launched by mobile applications, by either customizing and/or dynamically fine-tuning on-the-fly the adaptation behaviour on cross-cutting concerns?

RQ4: Would adding in the policy quantitative long term goals over finite horizons enable the user to delegate to the adaptation logic long term resource and/or quality management?

RQ5: Would utilizing a policy architecture using multiple simultaneously activated adaptation rules per adaptation-variable allow easy incorporation of multiple adaptation-concerns in the policy?

RQ6: Can we enable policy authoring of adaptation policies with high behavioural variability while reducing the combinatorial explosion by using multiple adaptation rules per adaptation-variable?

RQ7: Can the use of multiple execution cycles in the adaptation policy allow for further reduction in the number of rules necessary to model specify adaptation behaviour?

Evaluation of the Volare approach will proceed through one or more RQs in reference to each contribution of the Thesis. The evaluation on each RQ will be based on two evaluation methods:

- Qualitative issues, such as issues of structure or methodology, will be evaluated by the logical arguments presented.
- Quantitative issues may additionally be assessed by comparing the results attained by the case study derived simulated usage results, through the infrastructure designed concerning selected performance metrics of the mobile device at alternative monitoring scenarios.

Evaluation Strategy

Reasoning – Although usage data are of primordial importance to demonstrate how the real system behaves and to validate a project, it does not always provide sufficiently strong evidence on verifying the influence of each specific parameter to the adaptation behavior – even for relatively simple systems with few parameters. This is due to context parameters stochastic variation (bandwidth, user preferences, battery drop at cloud and non-cloud activity, services availability, etc.), that at the usage data are out of the control of the researcher. On the other side, analytic description of the system behavior is not available.

Consequently, the influence of each specific parameter or function in adaptation behavior may be evaluated more effectively, if it is possible to simulate the system behavior on the same sequence of context data, in alternative runs having the considered parameter at different values along with simulated correlation of associated data, and then compare the alternative simulated operation outcomes.

Example – As an example of this argument, the following two charts represent the policy execution results and context parameters on the same context usage matrix of the case study, but with one modification. In Fig. 8-1 there are the real context usage data over a monthly period, with alternating user preference causing frequent change of adaptation behavior, while Fig. 8-2 depicts the same parameters but at only one user preference (UP = “Normal). It is clear that isolating one discreet values parameter at one value at each run, may prove illuminating on the questions under investigation.

Figure 8-1– Monthly SR QoS parameters with UP changes Dataseries: 3LTPG_3CEC_USGM&4UPs_All24

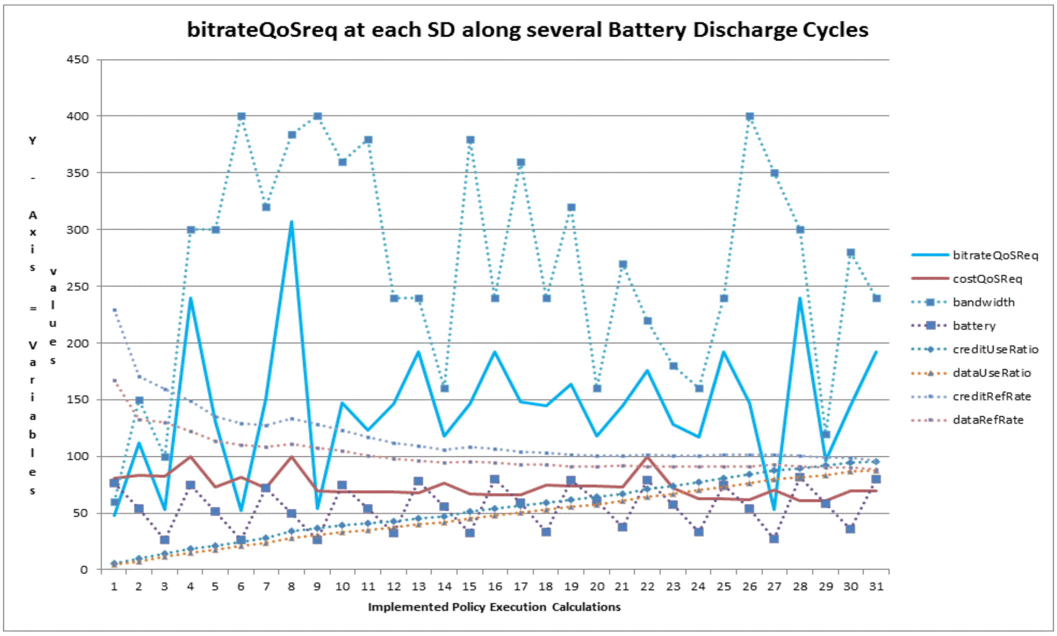
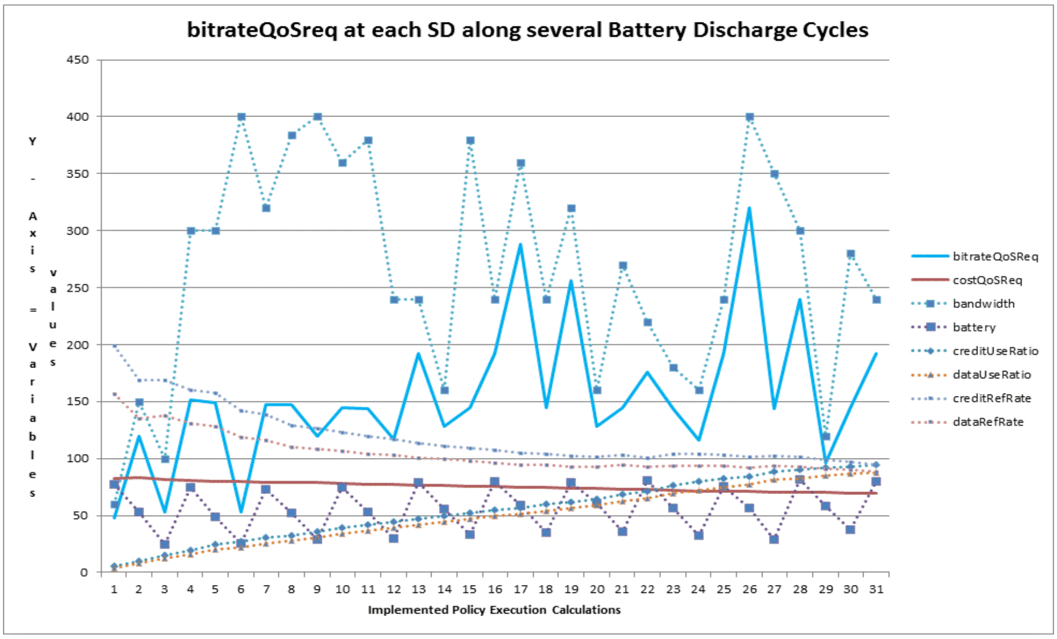


Figure 8-2 – Monthly SR QoS parameters with UP = "Normal" Dataseries: 3LTPG_3CEC_USGM&NO_All24



Simulation Setting – Fig. 8-1 and 8-2 depict the two main policy execution QoS parameters bitrateQoSReq and costQoSReq along with context parameters, at the initial adaptation of the last CD session of each day over the monthly context usage model. Fig. 8-1 represents the initial context usage model with alternating UPs, while in Fig.8-2 the same context usage model is set at user preference UP

=”Normal”. Within the monthly period, 11 battery discharge cycles are fully shown with dropping battery level till recharge at each fourth day.

Result Analysis – In both figures the generic constraint: **bitrateQoSReq <= 0.8*bandwidth (1)** is always applied, demonstrated by how the bitrate graph tends to be parallel to the bandwidth graph. Similarly at the first battery discharge cycles, due to **VLow** battery level, bitrateQoSReq is further reduced at the VLow battery points (battery LTPG rules), at all UPs except “HighQuality”.

Considering the differences in the two charts, in Fig. 8-1 in both QoS parameters graphs the peaks are due to occasional UP = “HighQuality” that imposes the highest attainable values for the QoS parameters without consideration of other constraints. When the UP changes to another one, a drop in the graph values is apparent.

On the contrary Chart 8-2 data with UP = ”Normal”, obviously allows to verify the gradual costQoSReq reduction as the creditUseratio increases. Chart 8-2 is much more helpful in validating and verifying the adaptation behavior, having isolated by simulation specific parameters at fixed values.

8.1 Quantitative Evaluation Methodology

For this purpose, the chosen, results-based, quantitative evaluation strategy for each RQ – where appropriate – consists in deriving simulations of the mobile device performance metrics over a period associated to the RQ in consideration, based on alternative monitoring scenarios for the context parameter of interest, and then comparing the outcomes on performance evaluation metrics.

In this perspective the evaluation study makes use of the recorded by the mobile device context usage model (derived from the usage data) on a CSD sessions sequence over a monthly period, and then runs simulations of the device adaptation behavior on the same CSD sessions monthly sequence, modifying one context parameter value at a time.

In the next paragraphs follows the design of the simulation methodology, concerning the main parts:

- the design of monitoring scenarios (context data matrices), based on the initial context usage model described in §7.2 and Appendix G
- the device adaptation behavior simulation methodology that will produce the simulated artifacts (simulated usage data) for comparison and evaluation.

Two sets of artifacts are required by the simulation case study:

- test suite(s), in this case the monitoring scenarios described in §8.1.3, each with different values on a specific parameter
- the simulated context & adaptation data matrix resulting from the simulation study on the sequence of CSD sessions over each monitoring scenario.

The simulation study will apply alternative monitoring scenarios providing simulated results on the mobile device performance – by simulating adaptation and operation results over the sequence of CSD sessions of the context usage model within the selected temporal horizon. Comparison of the alternative device performance results with each monitoring scenario, will provide the opportunity for drawing and documenting conclusions on each RQ.

Based on the infrastructure designed as outlined in §7.1 and the real monthly usage data and the assumptions presented and the derived context usage model in §7.2, a RQ simulation-based evaluation methodology has been designed on three consecutive stages, applicable for each monitoring-scenario (input context data set), when simulating the middleware operation:

1. **Define the appropriate Monitoring Scenarios**

Define the Monitoring Scenarios (context data matrices), the horizon(s) of interest to the RQ under examination, by modifying the basic context usage model (data matrix) on the selected parameter(s) of interest, setting alternative values.

2. **Implement Simulated Device Operation on each Monitoring Scenario**

Implement repeated simulated device operation imitating the device functions concerning the video-streaming application and deriving the observable metrics values at each successive recheck cycle of the monitoring scenario.

3. **Compare the alternative Simulated Results on the Monitoring Scenarios**

The results of the RQ-related device performance metrics will be compared at the end of the simulated device operation on the selected monitoring scenarios, evaluating the influence of the differentiated parameter concerning the RQ in consideration.

8.1.1 Designing the Monitoring Scenarios

In order to be able to evaluate the simulated device performance, by comparing RQ-related performance parameters influence on adaptation behavior metrics over alternative input (context) data matrices, appropriate monitoring scenarios need to be developed. These monitoring scenarios will cover the full sequence of

CSD sessions as well as of non-cloud activity sessions within the monthly (or in some cases the discharge cycle) horizon, adopting the context usage model of the recorded usage data values (§7.2) – thus also keeping intact the non-cloud activity sessions. Only a specific context parameter will be modified at a time. The context data fields of the data matrix may be distinguished in two categories:

1. **Fixed Context Data**

The fixed (non-modified) context data drawn from the context usage model are:

- *sessionStartTime, sessionEndTime, sessionNo, adaptationNo, dischargecycleNo* of each CSD session within the monthly horizon,
- the *webMBs* downloaded through the MNSP while surfing the web at each non-cloud session (from the *sessionEndTime* of each CSD session till next *sessionStartTime*), that count for the monthly data volume LTPG,
- the recorded *batteryDrop* at each non-cloud activity session of the mobile device (like: phone calls, SMS, MMS, surfing the internet, stand-by power drop, etc.) is denoted as *nonCloudbatteryDrop*$_j$ for the pair j of non-cloud and cloud sessions,
- the initial battery level at each new battery discharge cycle at the first CSD session, denoted as *batteryInitial.*

2. **Modifiable Context Data of the Monitoring Scenarios**

The remaining context data required by the Policy Engine for policy execution at the beginning of each recheck time within CSD sessions are:

- *bandwidth* at each recheck cycle;
- *user preference mode* at each recheck cycle;
- the values of the user-customizable context-variables (like: *priceMax* or the *SWCs, etc.)*.

By modifying a context parameter, like: bandwidth, user preference or a user-customizable context-variable value, alternative monitoring scenarios are defined and used in the following subsections. For instance in Fig. 8-2 vs. Fig. 8-1, only the UP has been modified from the context usage model, and has been set to “Normal”.

8.1.2 Regression Analysis for cloud session battery drop

A challenge for simulating the device adaptation behavior using the aggregated context data model derived above is the lack of a known relation to calculate battery drop within a CSD session. The aim is to calculate the approximate

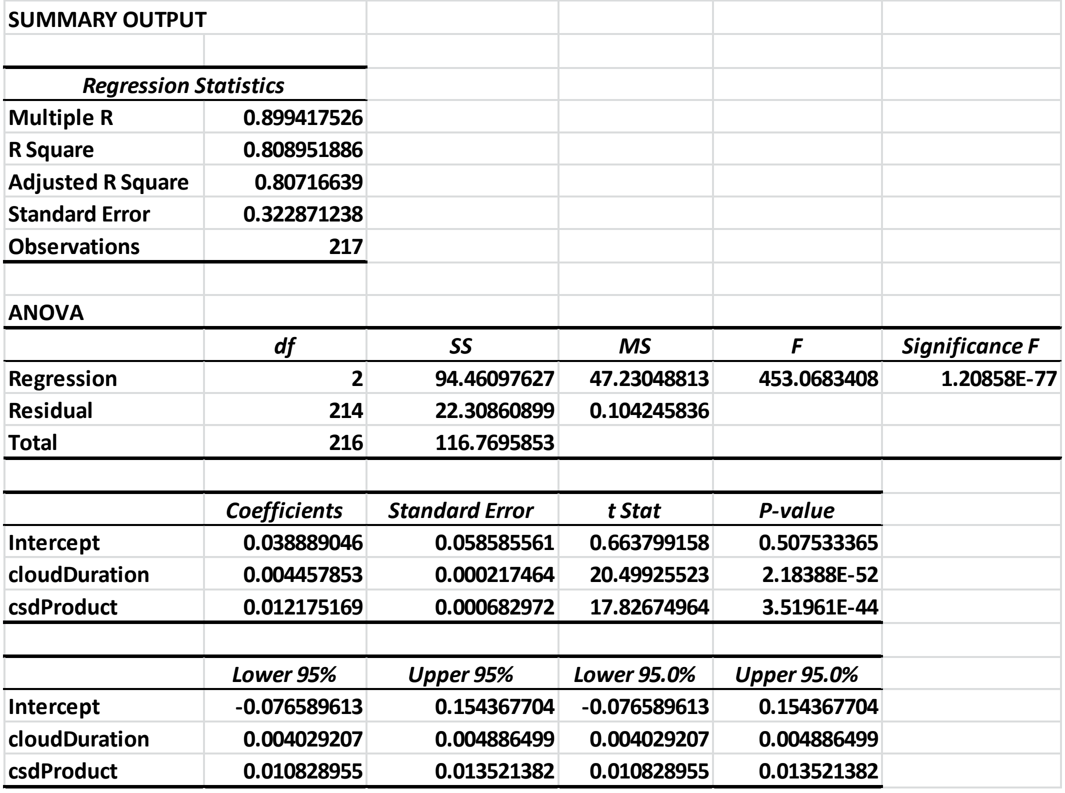
battery drop at every successive monitoring period (called “recheck cycle”) within each simulated CSD video-streaming session, as a function of recorded context parameters associated with energy consumption, based on the device usage data recorded over a monthly period (see §7.2 and Appendix G).

Current research indicates that mobile device energy consumption increases with more intensive calculations burden for CPU and RAM (by higher video-encoding bitrate), as well as with more intensive communication burden (by higher download data rate), especially if bandwidth is not high in relation to the download bitrate (on an analysis in smartphone energy sinks, refer to [92][94]). Consequently, the following energy consumption related context parameters are chosen:

- the *cloudDuration* of the monitoring period within the CSD session
- *bandwidth* monitored at this period within the CSD session
- the video-encoding bitrate *bitrateQoSProv* at which the device was downloading MBs from the bound *serviceID* at the monitoring period.
- the above parameters in the form of the *csdProduct* product:

  csdProduct = cloudDuration x bitrateQoSProv / bandwidth

Table 8-1 – Regression Analysis Statistics on Battery Drop



SUMMARY OUTPUT					
Regression Statistics					
Multiple R	0.899417526				
R Square	0.808951886				
Adjusted R Square	0.80716639				
Standard Error	0.322871238				
Observations	217				
ANOVA					
	df	*SS*	*MS*	*F*	*Significance F*
Regression	2	94.46097627	47.23048813	453.0683408	1.20858E-77
Residual	214	22.30860899	0.104245836		
Total	216	116.7695853			
	Coefficients	*Standard Error*	*t Stat*	*P-value*	
Intercept	0.038889046	0.058585561	0.663799158	0.507533365	
cloudDuration	0.004457853	0.000217464	20.49925523	2.18388E-52	
csdProduct	0.012175169	0.000682972	17.82674964	3.51961E-44	
	Lower 95%	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*	
Intercept	-0.076589613	0.154367704	-0.076589613	0.154367704	
cloudDuration	0.004029207	0.004886499	0.004029207	0.004886499	
csdProduct	0.010828955	0.013521382	0.010828955	0.013521382	

Based on the usage data recorded and through regression analysis, a *battery drop* relation has been found – applying ideas on battery level drop rates discussed in [42] and on smartphone power use in [92] and [94] – of the form:

battery drop = 0.0331 + 0.00445 * cloudDuration + 0.121 * cloudDuration * bitrateQoSProv / bandwidth (1), in % battery drop

with *multiple coefficient of determination* $R^2 = 0.815$ for the multiple regression, which is considered satisfactory for the evaluation study purpose.

The regression analysis statistics are indicated in Table 8-1, based on 216 usage data rows by the aggregated monthly usage data. The aggregation of the initial usage data allowed the reduction of errors in the monitored battery drop values, by integrating over longer time intervals.

Thus, relation (1) with parameters is retained as sufficiently accurate for simulating battery drop at each recheck cycle within a CSD session of the particular video-streaming application VSTREAM on the mobile, while assuming intact all non-cloud sessions (and all other eventual device parallel activities).

As discussed in [92] and [94], different devices and different applications may have different battery drop rates. Consequently this battery drop simulation on CSD sessions serves only as a proof of concept and cannot be generalized to other mobiles or applications. Additionally, some noise is included in the data by the battery level indication in % battery values and in bandwidth, since the monitored battery level values were rounded to the nearest integer with some error. Note however that the noise form not very accurate values is similarly introduced in all monitoring scenarios, thus all compared outcomes include a similar error.

8.1.3 Development of Simulation Calculation Functions

In order to implement the simulation of the device adaptation behavior on the case study video-streaming application, it is required to develop several calculation algorithms in imitation of specific functions of the device software components (middleware, application, CSP).

1. **Battery level calculation at each simulated recheck cycle**

Within a battery discharge cycle, consider recheck cycle **j** of the cloud session **i** (just after the non-cloud session i, since non-cloud and cloud sessions are typically interchanged) and referenced as recheck cycle **i,j**. At the first recheck cycle of the 1st cloud session of a new discharge cycle:

$\mathbf{batteryStartcloudsession_{1,1}}$ = batteryInitial **(1)**

The battery levels at the beginning and end of each recheck cycle are denoted as **$\mathbf{batteryStartcloudsessionCycle_{i,j}}$** and **$\mathbf{batteryEndcloudsessionCycle_{i,j}}$**, and may be calculated as follows, according to the battery drop formula in §8.1.2:

$\mathbf{batteryEndcloudsession_{i,j}}$ = $\mathbf{batteryStartcloudsession_{i,j}}$ – (16/3600)*$\mathbf{cloudDuration_{i,j}}$ + (40/3600) * $\mathbf{cloudDuration_{i,j}}$ * $\mathbf{bitrateQoSProv_{i,j}}$ / $\mathbf{bandwidth_{i,j}}$ **(2)**

if j = 1 (i.e., 1st recheck cycle of the cloud session i), then

$\mathbf{batteryStartcloudsession_{i,1}}$ =
$\mathbf{batteryEndcloudsession_{i-1,j'}}$ – $\mathbf{nonCloudbatteryDropsession_{i-1}}$, **(3)**

where j’ is the last recheck cycle of the previous cloud session i-1.

Thus the battery level calculation algorithm is established through these three formulas and is applied successively at the beginning and then at the end of each recheck cycle of a cloud session.

2. Calculation of context- or history-dependent parameters

The calculation-variables that depend on the context and on the last recheck cycle usage data like: cloudMBs, webMBs, cloudCost, increasing monthly data volume, decreasing allowances, etc., are updated at each recheck cycle to their new values and used as context data at the next recheck cycle. The simulation study calculations need to include the updating of all these non-constant parameters, declared as calculation-variables in the policy, before providing the new values to the Policy Engine Working Memory.

3. Simulating the SR QoS levels evaluation mechanism

After a simulated policy execution, the simulation tool simulates the middleware QoS evaluation algorithm (as described in §4.5.2 and §4.7.1) is imitated and evaluates the SR QoS levels to reach the decision to implement rediscovery or not (i.e. to reject policy execution results and continue as before), employing the associated policy-based threshold parameter values.

4. Simulating the CSP service selection mechanism

At the simulation procedure, the simulation tool simulates the virtual CSP component. Since the CSP service selection algorithm is known as well as the four available cloud services with their QoS levels, a simulation function implements the service selection at each recheck cycle, deriving the offered service QoS levels.

8.1.4 The Simulation Procedure

The simulation procedure implements repeated cycles of simulated service discovery and binding to a cloud service over a sequence of CSD sessions, introducing successively each context data row of the data matrix, deriving the simulated policy execution SR QoS terms values, simulating the SD and binding to the discovered service and starting or continuing consumption of the bound service.

Developer-Guided Simulation Tool

An automated, developer-guided offline tool has been designed, incorporating the Policy Engine, to implement sequential policy execution of the context data rows for each monitoring scenario context data matrix, deriving the new adaptation QoS levels and middleware configuration-variables values based on the current context. The simulation tool needs to simulate also the CSP activity on service discovery and service selection from the test-bed, as well as the activities of the middleware modules, functions described in §8.1.2

The Simulation Procedure Operating Steps

The simulation procedure at each repeated **recheck cycle** within every simulated CSD session, implements successively the following simulated **operation steps:**

1st Step: Get the new context data – At each new simulated recheck cycle, the new data row includes all device and middleware operation parameters declared in the policy as context-variables (like: battery drop, bandwidth, user preference, webMBs, etc.), as well as the updated by the last simulated recheck cycle context or history dependent calculation-variables.

2nd Step: Initiate policy execution (calculation) – Execute simulated policy execution on the current context data row and derive the new adaptation calculation results concerning the QoS-variables values and the middleware configuration values.

3rd Step: QoS evaluation decision-making on rediscovery – If the recheck cycle concerns a CSD session at runtime, the decision-making algorithm of the BindingM on rediscovery, decides on policy-based threshold parameters if adaptation & rediscovery results are to be implemented or not, based on the deviation between current and new QoS terms values. At the initial policy execution in a CSD session however, service discovery is directly implemented without evaluation.

4th Step: SR adaptation and dispatch to the CSP – In case the decision of the previous step is for discovery or rediscovery, the SR QoS terms are adapted by the new policy execution results and the SR is dispatched to the CSP.

5th Step: Service discovery & selection by the CSP – The simulated CSP receiving the SR QoS values executes service discovery from the test-bed of four cloud services and (since all are available) selection of the fittest service and a builds and dispatches a simulated Service Offer.

6th Step: Service Offer evaluation – If the recheck cycle represents the first on in a CSD session (initial service discovery), the simulated Service Offer evaluation mechanism of the (§8.1.2), accepts the Service Offer and binds to the selected service. If the recheck cycle represents a runtime adaptation at a CSD session, the simulated Service Offer evaluation mechanism (§8.1.2), evaluates the SO and either accepts it and binds to the selected service, or it continues as before if the offered service is the same or there is no significant benefit or need for adaptation.

7th Step: Service consumption for the recheck time period – The simulated results specify by policy execution the duration of each recheck cycle – except if there is manual termination of the CSD session by the application. The simulation tool based on the corresponding bound *serviceID* and the *QoS* terms values calculates the values associated calculation-variables of the Composite Policy, that represent the adaptation behaviour of the system.

End of this repeated cycle at the last context data row of the monitoring scenario data matrix.

8.1.5 Common characteristics of the simulation study

Comparisons are supported by developer-guided semi-automated charts of the simulation tool on the selected parameters of interest from the derived simulated usage data.

1. Charts Presentation

Through the charts below, we examine several selected context data matrices over a full monthly cycle or a battery discharge cycle (from full battery to very low battery).

In each chart a different bandwidth variation may be introduced (including fixed bandwidth data series as reference), while battery level is dropping with time.

The battery level graph typically indicates a battery discharge cycle every 3 days and imposes bitrate reduction, mainly when battery <= 30%.

Dotted lines in the Charts depict the basic context-variables and monitoring metrics of the LTPGs, in contrast to the full lines of the depicted results on QoS-variables.

Usually the 6 most common context-variables and LTPG metrics depicted are:

- **battery** level (at % full scale) at the beginning of each recheck cycle (adaptation) depicted, thus presenting each battery discharge cycle by the decreasing battery level
- **bandwith** typically in equivalent data rate units (KB/s)
- **creditUseratio** and **dataUseratio** (% of the respective allowance already spent)
- **creditRefRate** and **dataRefRate,** the decision-making metrics on High or Low or VLow LTPG availability level, with “correct” values <= 100. If values >= 100, then Low or VLow assessment is taken.

The main QoS-variables derived by policy execution, depicted in the charts with bold continuous line, are:

- **bitrateQoSReq**, typically in equivalent data rate units (KB/s)
- **costQoSreq** (at % of the **priceMax** parameter value – for comparison).

Of importance on the simulated usage data depicted in each chart is the user preference selection (one UP or alternating UPs).

2. Bandwidth restriction on the highest attainable bitrateQoSReq values

For client – server network communications reasons, the current bandwidth value imposes the technical constraint that: **bitrateQoSReq <= 0.8*bandwidth** **(1)**

As it is obvious, any *bandwidth* value decrease causes a constraint to the *bitrateQoSReq*, expressed by the HighQuality *bitrateQoSreq* graph, since at UP = “HighQuality” it takes the maximum technically attainable value bitrate respecting constraint (1).

8.2 Dynamic Context-Aware Adaptation of the SR by mobile applications

RQ1: Can adapting the QoS levels of a service request (SR) launched by a mobile application before service discovery and binding, based on the device current context and policy requirements and goals, lead to discovery of more fitting services and to more efficient use of device resources?

The solution chosen by the Volare approach in order to minimize the adverse effects for a mobile application launching CSD due to mobility and subsequent context variation and constrained mobile resources – as discussed extensively in §2.1 – consists in adapting not the active application launching CSD, but instead adapting: (i) the SR QoS terms – even at runtime context change by activating re-discovery, (ii) the mobile middleware assigning the DCAA capability (through adaptation of its configuration variables).

8.2.1 Research Question Issues Investigation

RQ1 raises two issues, distinguished in the following two research questions:

RQ1a: Can adapting the SR QoS levels based on the current context lead to discovery of more fitting services?

RQ1b: Can adapting the SR QoS levels based on the current context lead to more efficient use of device resources?

a) Investigating RQ1a

At a CSD session, given that the mobile device may be at a state of low bandwidth or low resources (like: battery or credit availability) or user preference for constrained requirements, it is evident that the current context and the policy constraints and goals lead to an optimal QoS derived at policy execution, and the most fitting service for discovery and binding to is the one with QoS terms closest to it. If at a CSD session, the SR QoS levels are not specified in relation to the current context and policy requirements and goals but represent prefixed values, then performance problems may appear at various contextual situations.

For instance at the case study video-streaming application, there may be:

Low bandwidth situation – If the video-encoding bitrate provisioned by the CSP is higher than 80% of the current bandwidth, delays are to be expected and application performance degradation due to bandwidth lower than required by the video-streaming process [128][110]. For example, if the video encoding bitrate provisioning value is 256 KB/s and the available bandwidth is 150 KB/s, binding to this service will lead to long delays.

Low battery situation – Similarly, if current battery level is low, discovering and binding to a high QoS levels service with high demand on device resources will further intensively reduce power level, that risks to leave the device without enough power before re-charge time.

User preference situation – As described in chapter 7, Evelyn (the user) may set her preference for *"LowCost"* or *"SaveBattery"* mode, or set the *priceMax* maximum value per MB downloadable from the cloud service bound to. Predetermined and non-adapted SR QoS levels at CSD are incompatible with user preferences – which by default represent compromise between what is desirable and what is attainable under the circumstances.

Consequently, only policy-based context-aware adaptation of the SR may lead to the discovery of the most fitting service.

b) Investigating RQ1b

A mobile device has consumable resources like battery power or a ceiling value at other resources like CPU or RAM, etc. Similarly, the adaptation policy may introduce consumable "virtual resources" like: monthly credit allowance on cloud services, monthly data volume allowance through the mobile network, etc.

By "more efficient use of device resources" at a CSD session in RQ1, it is meant that policy-based context-aware service discovery should ensure that physical or virtual consumable resources are spent with caution. This requirement may be implemented in different policy-based ways, either by a simple policy of the form: "If battery < Low Level A Then …" with adaptation action on specified resource levels, or in a more advanced form as in Volare by introducing in the policy LTPG models with consideration of each resource time horizon, resulting in monitoring resource use metrics and taking corresponding adaptation actions on the SR QoS levels within the LTPG horizon. Consequently, it is argued that only policy-based context-aware adaptation of the SR QoS on mobile applications launching CSD can manage efficiently physical or virtual (policy-based) consumable resources or constrained resources with a ceiling value, as the following simulation artifacts will attest.

Simulation Setting

In order to demonstrate the critical importance of policy-based context-aware adaptation on the efficient use of cloud services and of the device physical or virtual (policy-based) resources, the simulation study examines two adaptation cases employing the same monitoring scenario, namely the context usage model that includes a sequence of 154 CSD sessions over monthly duration. The only difference between the two cases, as explained below, is that at the first adaptation case the SWCs values of all adaptation-strategies in the "Composite Policy" (except the default Sgen) are set to 0, while at the second adaptation case all SWCs have their default values.

1. Very Shallow Adaptation Case 1

At this case, very shallow SR adaptation is implemented and only on the video-encoding bitrate so that:

bitrateQoSReq = 0.8*bandwidth **(1)**

to ensure that the video-streaming will continue without delays.

No constraints are imposed for other resources (like: battery level, monthly credit or data volume allowance or battery level) while the context usage model dispays an average battery discharge time of 3 days.

The case study global and application policy files for the video-streaming application are used, but the SWC coefficients are set to zero:

wSpref = wScost = wSres = wSdat = 0 **(2)**

As described in chapters 4 and 5, at policy execution all adaptation-rules with SWC =0 are overridden (a testability feature of the Volare APSL and the Policy Engine) for overriding easily all rules of one (or more than one) adaptation-strategy to examine the adaptation behavior with or without it.

In this case of SWCs = 0, at policy execution only the adaptation-rules of the Sgen adaptation-strategy are evaluated by the Policy Engine, specifying only default values and generic shallow adaptation of the bitrate in relation to the bandwidth (formula (1) above).

All constraints existing in the adaptation-rules of the other adaptation-strategies on battery, data volume, credit LTPGs or user preferences are overridden, allowing maximum attainable value to *bitrateQoSReq* (as long as it is <= 0.8*bandwidth) and maximum values to costQoSReq, two SR QoS-variables that mainly determine the selected *serviceID* (on cost and bandwidth issues).

2. Full Adaptation Case 2

In this case, the same monthly context usage model is used and the same policy files, but with the SWCs at their default values as indicated in Table 7-6 of the case study. Depending on the context, the selected rules at policy execution on the consumable resource LTPGs may specify QoS level values reduced in comparison to the maximum attainable values, to help achieve the LTPGs.

Result Analysis

The selected parameters from the sequential policy execution results on the two adaptation cases are depicted in the two charts 8-1 & 8-2. In order to improve visibility of results, only the last daily CSD session every day in the month is depicted in the charts, with the evolution of selected metrics.

Figure 8-3 – Adaptation Case1: Very Shallow Adaptation Dataseries: 3LTPG_3CEC_USGM&4UPs_SIMPLE

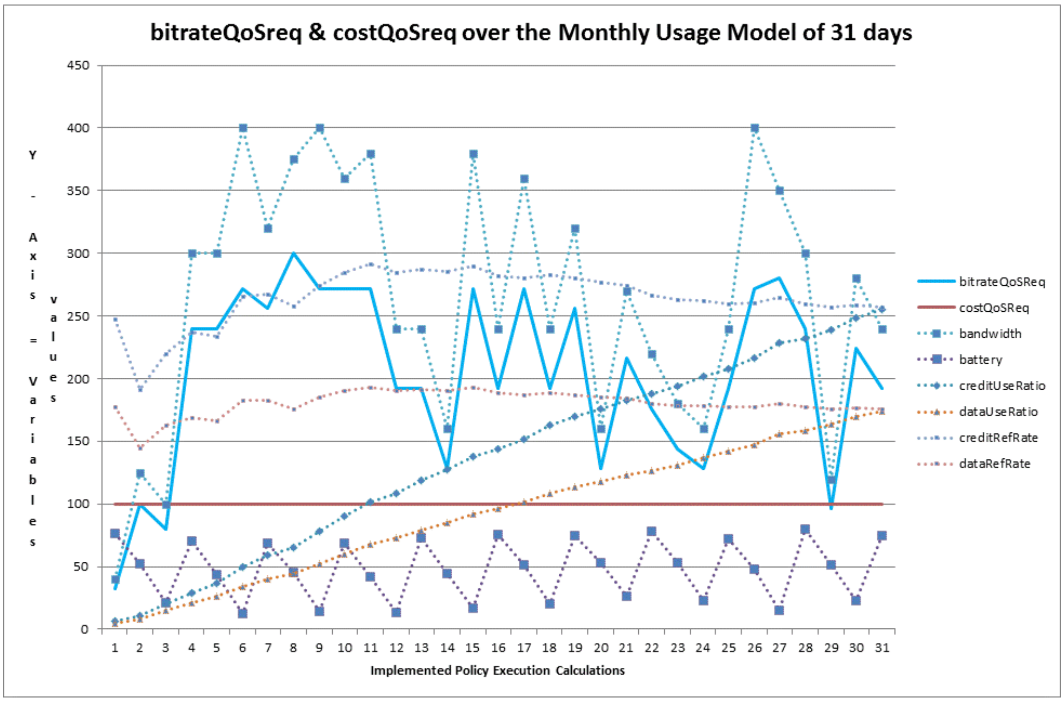
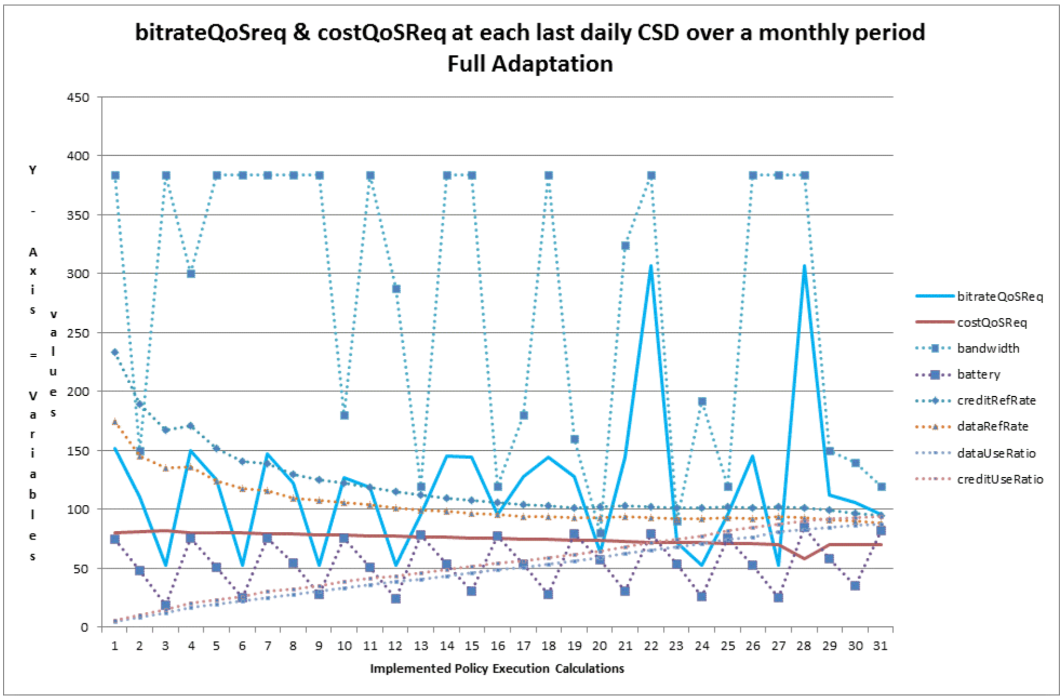


Figure 8-4 - Adaptation Case 2 – Full Adaptation with LTPGs Dataseries: 3LTPG_3CEC_USGM&4UPs_All24



Each of these charts demonstrates selected context & adaptation parameters values over the full monthly period of the usage data sequence of CSD sessions. The full lines in blue and red represent the video-encoding bitrate *bitrateQoSReq* QoS request in KB/s and the *costQoSReq* of the service requested in percentage of the policy *priceMax* value. The dotted lines represent the context parameters: *bandwidth* in KB/s (upper chart part), *battery* level % (lower chart part), *creditUseratio* and *dataUseratio* in % of the respective credit or data allowance. The battery level (% full scale) dotted line at the lower chart part, makes distinct the 11 successive decreasing line segments of battery discharge cycles.

Comparison of Figs. 8-3 & 8-4 – Adaptation Case 1 vs. Adaptation Case 2

Case 1 -- Since the SR is adapted at nominal QoS levels, high QoS bitrate and nominal cost QoS levels service is discovered and bound to at each adaptation, thus leading to high resources use. **Case 2** – On the contrary in case 2, the SR is adapted to in a context-aware manner, thus leading to discovery of more services with more fitting QoS, economizing on device resources. Specifically:

***bitrateQoSReq* – Case 1 –** It is clear from the *bandwidth* & *bitrateQoSReq* lines that bitrate follows the formula bitrateQoSreq = 0.8*bandwidth (1), without any other restriction from battery level drop, or credit or data volume increase, at the very shallow Adaptation Case 1. **Case 2 –** It is additionally clear from the *bandwidth* & *bitrateQoSReq* line graphs that in this case the bitrate QoS request values are reduced when *battery* is low and when *dataRefRate* exceeds 100.

***costQoSReq* – Case 1 –** This QoS-variable is specified at 100% the maximum attainable value. **Case 2 –** In this case *costQoSReq* value is reduced when *creditRefRate* exceeds 100%, since credit is being spent at a faster than expected rate.

***creditUseratio* – Case 1 –** The resulting parameters credit use ratio already at the 11th day of the month has covered the credit allowance and by the end of the month escalated to 254.9%. **Case 2 –** The resulting credit use ratio in this case does not exceed 100% by the end of month, thus ensuring that the credit goal is achieved.

***dataUseratio* – Case 1 –** The data volume use ratio exceeds the monthly allowance by the 16nth day and escalates to 174.5% by the end of month. **Case 2 –** The data volume use ratio does not exceed the monthly allowance by the end of month and escalates only to 87%.

***Min battery level* – Case 1 –** The average value of the minimum battery level at the end of each discharge cycle is 18.4% instead of the minimum 20% postulated. Out of the 10 full discharge cycles in the month there, have been 6 LTPG failures with battery level min < 20% to shallow adaptation logic. **Case 2 –** The average value of the minimum battery level at the end of each discharge cycle is 27%, satisfying the LTPG of minimum 20% battery level.

Table 8-2 above recapitulates key values of the parameters considered in the above two adaptation cases, to focus numerically on the different outcomes.

Table 8-2 – Characteristics of Monthly Usage Results in Charts 8-3 & 8-4

Parameter Considered – Units	**Shallow Adaptation Adaptation Case 1**	**Full Adaptation Adapt. Case 2**
CSD Sessions	153	153
Data rows	1843	1604 mon. cycles
No of Adaptations	217	208 adaptations
Max price per MB downloaded	0.0024 GBP/MB	0.0024 GBP/MB
Total MBs downloaded	**6977 MBs**	**3513 MBs**
Data Volume Allowance MBs	4000 MBs	4000 MBs
dataUseratio %	**174.5%**	**87.8%**
Cloud Cost Monthly GBP	**12.75 GBPs**	**4.71 GPBs**
Monthly Credit Allowance GBP	5 GBPs	5 GBPs
creditUseratio %	**254.9%**	**94%**
No of battery discharge cycles	10	10
Average Min battery level %	18.4%	27.16%
Failures in Min Bat. Level 20%	6 failures in 10	1 failure in 10

Table 8-3 – Distribution of services bound during the month in Fig. 8-3

serviceID	**service1**	**service2**	**service3**	**service4**	**Total**
Duration	1730	9283	20310	11676	42999
%	4.0%	21.6%	47.2%	27.2%	100%

Table 8-4 – Distribution of services bound during the month in Fig. 8-4

serviceID	**service1**	**service2**	**service3**	**service4**	**Total**
Duration	13604	21924	6494	977	42999
%	31.6%	51.0%	15.1%	2.3%	100%

Conclusions -- The conclusions drawn on the RQ1a & b are the following:

- The policy-based context-aware adaptation of the SR QoS levels is absolutely required for discovery of the most fitting service
- Policy-based context-aware adaptation of the SR QoS levels allows the adaptation logic to ensure that the device physical or virtual (policy-defined) resources can be managed efficiently as in the adaptation case 2 where all LTPGs are achieved, while in the shallow adaptation case 1 all goals failed.

8.2.2 Implications

In both the above examples, generic adaptation on the bitrateQoSreq (the video-encoding bitrate) was established by the adaptation logic even in case 1. If this generic adaptation did not exist, and the SR had only a nominal unadapted value, than in all cases where bandwidth were smaller to the bitrateQoSReq, long delays would be expected and unsatisfactory operation for video-streaming.

In the two above examples on the same context usage model, the middleware-based Volare DCAA approach enables improved performance for mobile applications with no inherent adaptation capability, by enabling capability for service discovery with QoS levels corresponding to the current needs of the mobile user, with benefits like:

a) Satisfactory performance quality and minimization of delays due to fitting video encoding bitrate in relation to the current bandwidth
b) Optimized use of policy-based physical (battery) or virtual resources like cost of binding on the Cloud, by adapting the service request QoS levels to the current context (including user preference requirements), and thus discovering a fitting service that allows reasonable use of the resources, and relieving the user from the burden of monitoring them and acting accordingly.
c) Dynamic context-aware adaptation at runtime, at significant discrepancy between current and new QoS levels due to context change, is also required for long service bindings, since context variation at runtime may make the service bound to not fitting to the new current context.

8.3 Two-level Policy Structure of the Adaptation Logic

RQ2: Does enabling mobile applications SRs adaptation that takes into account both the device's capabilities and application's requirements, by utilizing two-level policy architecture, provide advantages in policy authoring while reducing policy complexity?

Typically for policy-based adaptation logic that takes into account both the device's capabilities and the application's requirements, the firmware (middleware) developer and each application developer should cooperate to develop the specific adaptation logic for each application that answers to the joint requirements. However, as the purpose of the Volare approach is to enable for support by the middleware of unanticipated mobile applications launching cloud service discovery (CSD) in an easy way for the application developer, without requiring a joint cooperated effort of different developers, two-level policy architecture has been adopted.

The Volare Composite Policy Procedure -- At an active SR by a mobile application with a stored application policy, the adaptation logic – called in this work “Composite Policy” – is automatically created by the middleware by parsing and merging two policy files: the firmware “global” policy file and the active “application” policy file. Then, at each policy execution the Policy Engine runs the “Composite Policy”, the middleware updates the context-variables and the statistic-calculation-variables (Working Memory), and new values for the QoS-variables and the middleware configuration-variables are calculated.

This solution leads for each active application to a “Composite” adaptation logic that also covers the active application requirements, but with the firmware part prefixed. All the middleware-supported context-variables are predetermined in the global policy as well as the global QoS variables and any intermediate variables for the global policy constraints and goals, while the application declares the application-specific QoS and eventually required intermediate variables.

Benefits in using the “Composite Policy” -- The Volare two-level policy architecture, by having all middleware-supported context-variables and the global QoS-variables as well as required intermediate variables (calculation- or auxiliary-variables) declared at the global policy, facilitates the task of the application policy developers that only need to declare application-specific QoS-variables.

Similarly, it is the global policy developer’s task to configure the User Preferences Model and to define the User Choices Profile and the SWCs, thus totally relieving the application developer of this burden, while all these declared parameters may be directly used in the application adaptation-rules.

On adaptation-variables common across both policies (global QoS-variables and configuration-variables) the application policy developer may introduce in the application policy file the required rules, thanks to the Volare approach characteristics of: (i) allowing multiple rules in the “Composite Policy” (i.e. across both policies) that may be selectable at policy execution and (ii) inter-policy conflict resolution capability through priority assigning by keywords, (iii) the weight expression for each rule. These three features allow the application policy developer to author rules on the common for both policies adaptation-variables, although such rules may exist in the global policy – selectable under the same context instance, and:

- either to set a higher priority to a rule on a common adaptation-variable, thus overriding global policy selectable rules on this variable;
- or to specify the weight expression so that all selectable rules across both policies may contribute to the adaptation result.

Through these characteristics, the complexity of application policy authoring is reduced to the minimum necessary, while allowing the policy developers for unanticipated applications to easily provide the application policy, having read the global policy and conforming to its constraints and goals, but without the obligation to cooperate with the global developer in order to build a joint policy.

8.4 Integrating a configurable User Preferences Model

RQ3: Can we enable the user to influence the QoS levels adaptation of the SRs launched by mobile applications, by either customizing and/or dynamically fine-tuning on-the-fly the adaptation behaviour on cross-cutting concerns?

Mobile applications may have a User Interface for setting preferences, but this is an application-specific feature. The middleware assigning DCAA capability is expected to support different mobile applications launching CSD and its adaptation logic may have goals and requirements on quantitative or qualitative characteristics on adaptation behaviour common for all applications, such as managing resources cost, battery, data volume, rediscoveries at runtime, etc. These goals or requirements should be user-customizable, without the need to update the policy.

The solution the Volare approach introduces to satisfy this need for a customization tool on common issues of the middleware adaptation logic for every mobile application assigned SR DCAA capability, consists in installing a configurable by the developer User Preferences Interface (UPI) application, that operates as an external multi-sensor dispatching to the middleware the current user preferences (see also related subsections §6.4 and §4.2).

1. Global Policy Parameters Customization

The global policy developer configures through the UPI the User Preferences Model, specifying the customizable parameters, their default values and upper/lower bounds (for numeric parameters) in the User Choices Profile (UCP).

Then customization of the global policy on the customizable parameters by the user is easy through the UPI, by modifying the default values at the User Choices Profile. Such parameters configured through the UPI and the global policy to be user-customizable are: the LTPG goals, threshold parameters, maximum allowed price per MB downloaded, whether adaptation at runtime should be restrained on LTPG issues or not to prevent frequent rediscoveries, etc. Such parameters may be either quantitative or decision-making parameters (string or Boolean type).

Since the global policy is common for every mobile application installed on the device that is subjected to SR DCAA through stored application policy, this UPI feature allows uniform adaptation behaviour customization on all application sessions. Consequently it facilitates the customization by the user, without policy update, of operation goals that concern all applications, such as respecting LTPGs on credit or data volume on a monthly cycle or battery drop within a battery discharge cycle, or avoiding non-necessary rediscoveries at runtime, etc.

2. User Preference Mode Selection

Another need detected when running the prototype, is the capability for the user to impose dynamically different adaptation behaviour on common characteristics for all applications, in response to a change of user intentions or real-life needs.

As described in §6.4, the UPI application when configured by the global policy developer allows user preference mode selection – by selecting on-the-fly one of several policy-based options, like: "HighQuality", "Normal", "SaveBattery", "LowCost" – with each option specifying alternative adaptation behaviour.

Demonstration of effectiveness of the configurable User Preferences Model – In the following examples is demonstrated the effectiveness of the Volare configurable User Preferences Model through the UPI for customization or fine-tuning of the adaptation behaviour by the user on cross-cutting concerns on the case study video-streaming application, through simulations run on monitoring scenarios based on the context usage model.

8.4.1 Simulation Setting

The 4 User Preference (UP) Graphs – The parameter graph at each User Preference shows the differentiated adaptation behavior as derived by policy execution for each UP, under all other context conditions the same.

Battery Discharge Cycle – The *battery* level in the chart within the Battery Discharge Cycle (time indications omitted for simplicity) is dropping from a maximum (full charging) to a minimum value. In the monthly model, there are 11 discharge cycles. The *batteryUseratio*, depicted by a dotted line, is steadily increasing and is one of the monitoring metrics for specifying adaptation for the Battery LTPG.

The bandwidth restriction on the highest attainable bitrateQoSReq values – For client-server network communications reasons, the current bandwidth value imposes the technical constraint that: ***bitrateQoSReq* <= 0.8****bandwidth*** **(1).** As it is obvious, any *bandwidth* value decrease causes a constraint to the *bitrateQoSReq*, expressed exactly by *bitrateQoSReq* graph at UP = “HighQuality”, where it takes the maximum technically attainable value respecting constraint (1) without any other constraint (except at VLow battery).

Remaining Context or Calculation Input Parameters – In addition to the basic context-variables *bandwidth* and *battery*, the LTPG monitoring metrics “creditUseratio” & “creditRefRate”, and “dataUseratio” & “dataRefRate”, are all depicted with dotted lines in contrast to the full lines of the QoS-variable(s). In order to better differentiate on the influence of each context-variable on the results, we construct and group together a family of Charts for each parameter within the same Battery Discharge Cycle. In these successive charts, we examine the change of one context variable at a time, along with the Battery Discharge Cycle at each of the 4 Ups, producing the following charts:

a) A chart depecting only the battery variation (and the dependent batteryUseratio) and bandwidth at affixed high value, as reference Chart, demonstrating the influence of the Battery Mgmt LTPG.

b) A chart with bandwidth dropping in 3 steps along the Discharge Cycle, to demonstrate parameter changes with bandwidth drop.

c) A chart with regularly repeated bandwidth variation, to demonstrate parameter changes with bandwidth drop.

2. The most used Context Data Series used for the Charts

DataSeries_1 – Fixed bandwidth along a Battery Discharge Cycle

DataSeries_1 have a fixed high bandwidth value along the whole Battery Discharge Cycle. It serves as reference for the other context data series and their respective charts that have varying input parameters in addition to battery and batteryUseratio.

DataSeries_2 – 3 Step fixed bandwidth along a Battery Discharge Cycle

DataSeries_2 have a decreasing bandwidth value scaled in 3 steps along the Battery Discharge Cycle. It serves to demonstrate the decreasing bandwidth constraining influence on *bitrateQoSReq* due to constraint (1).

DataSeries_3 – 5 Step variation bandwidth along a Battery Discharge Cycle

DataSeries_3 have a regularly repeated bandwidth values profile in 5 intervals across the Battery Discharge Cycle. It serves to demonstrate the adaptation behavior in varying bandwidth conditions at different batteryUseratio values, demonstrating the Battery Mgmt LTPG influence on the parameter under consideration.

Dataseries_4 - Monthly Usage Model Data at each of the 4 UPs

The Monthly Usage Model data comparison purposes have been simulated with each of the four User Preference Modes. It serves to demonstrate the adaptation behavior across the monthly horizon, with increasing consumption of resources.

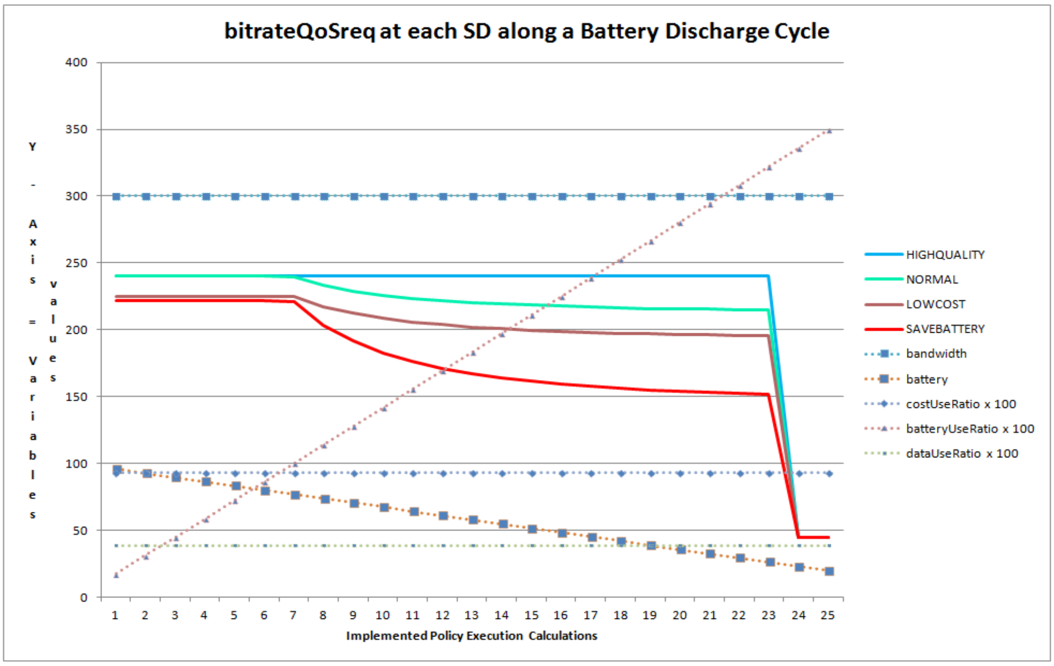
However, case-specific context models may be used also, with simulation-based usage data.

8.4.2 Results Analysis

1. Charts with adaptations within a full Battery Discharge Cycle (3 days)

1st Parameter: bitrateQoSReq -- In Figure 8-1 above the adapted bitrate QoS Request (bitrateQoSReq) parameter graphs are depicted, with results provided under each of the 4 User Preference Modes. The X-axis depicts policy executions at initial Cloud Service Discovery sessions within a full Battery Discharge Cycle data series, with bandwidth and all other context parameters at the same values for all UPs.

Figure 8-5 – SR bitrateQoSReq Graphs/UP without bandwidth variation Dataseries_1: Context data within a full Battery Discharge Cycle



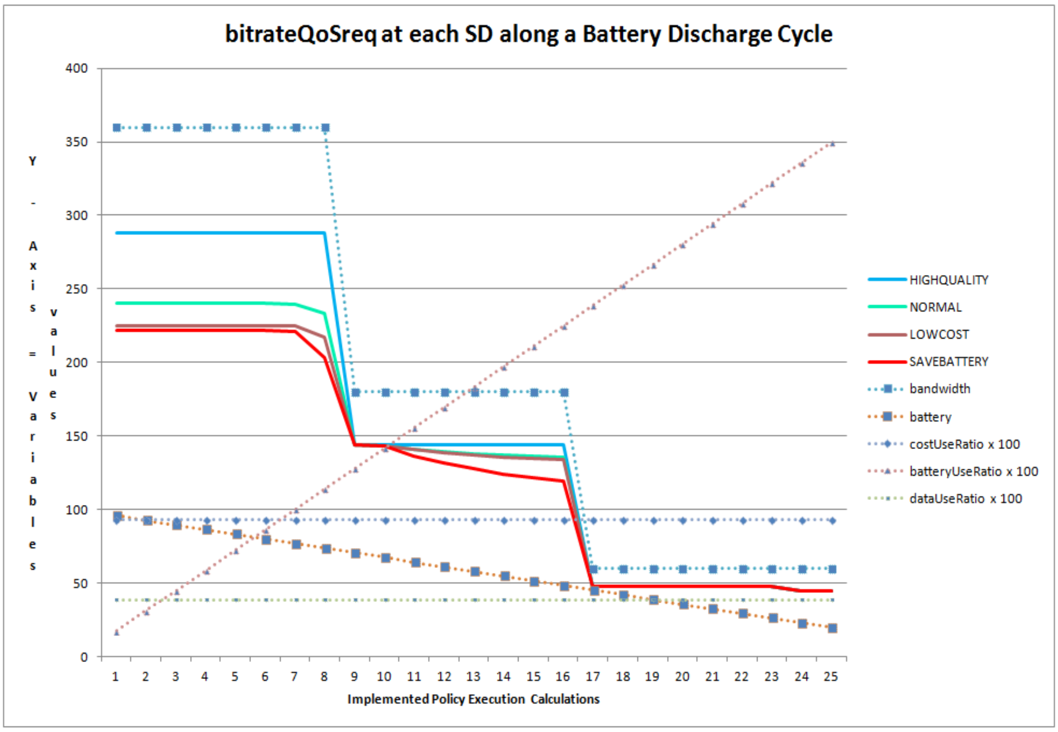
Main Remarks

a) Influence of the Battery Mgmt LTPG with the batteryUseratio

It is noted – according to the Adaptation Policy – that as the batteryUseratio value is increasing above the 100% gridline, it causes a gradual drop in bitrateQoSReq, depending on the current UP (except at “HighQuality” where the ratio is ignored). Similarly, at VLowBattery (< 25%), the VLowBat adaptation-rules of the application policy cause a steep bitrateQoSReq value drop. The difference of bitrateQoSReq graph (and consequently of the SR) at each alternative UP, under the same other context data, demonstrates the alternative adaptation behavior that the user may impose dynamically through the UPI.

In Figure 8-2 below, the adapted bitrate QoS Request (bitrateQoSReq) graphs are depicted with results provided under each of the 4 User Preference Modes by policy execution at initial Cloud Service Discovery sessions. Here we are using decreasing bandwidth values scaled in 3 steps within the Battery Discharge Cycle and all other context parameters at fixed values.

Figure 8-6 – SR bitrateQoSReq Graphs/UP with 3 Step bandwidth variation Dataseries_2: Context data within a full Battery Discharge Cycle



Main Remarks

In addition to the equally valid previous remarks (a) and (b), it can be noted:

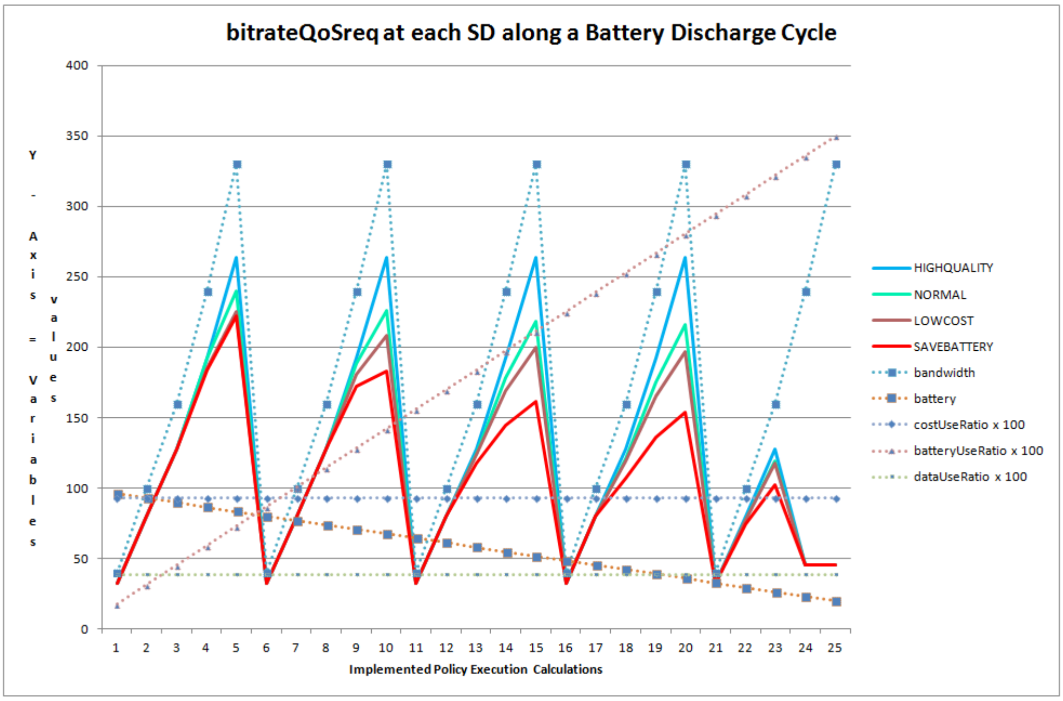
c) Constraining Influence of decreasing bandwidth on bitrateQoSReq

At each bandwidth value step in the chart, the bitrateQoSReq is abruptly reduced as imposed by constraint (1) in (a) above, to allowable levels.

d) Constraining Influence of the increasing metric batteryUseratio > 100

The same explanation on the "irregular" delay that bitrateQoSReq values expected at bandwidth value step 2, as the decreasing graphs due to the policy-based gradual adaptation should have started earlier – as soon as the batteryUseratio exceeded 100 (the 100 gridline in the chart). And in fact the adaptation-rules provided values were decreasing, but the decrease has been abruptly pruned, since the max values should have been <= 0.8*bandwidth = 0.8*180 = 144. This policy-based value pruning by constraint (1) is the reason why the 4 graphs show this "peculiar" delay in conforming to the Battery Mgmt LTPG when the batteryUseratio becomes > 100.

Figure 8-7 – SR bitrateQoSReq Graphs/UP with bandwidth variation Dataseries_3: Context data within a full Battery Discharge Cycle



In addition to the previous remarks (a), (b), (c) and (d) above, it can be noted:

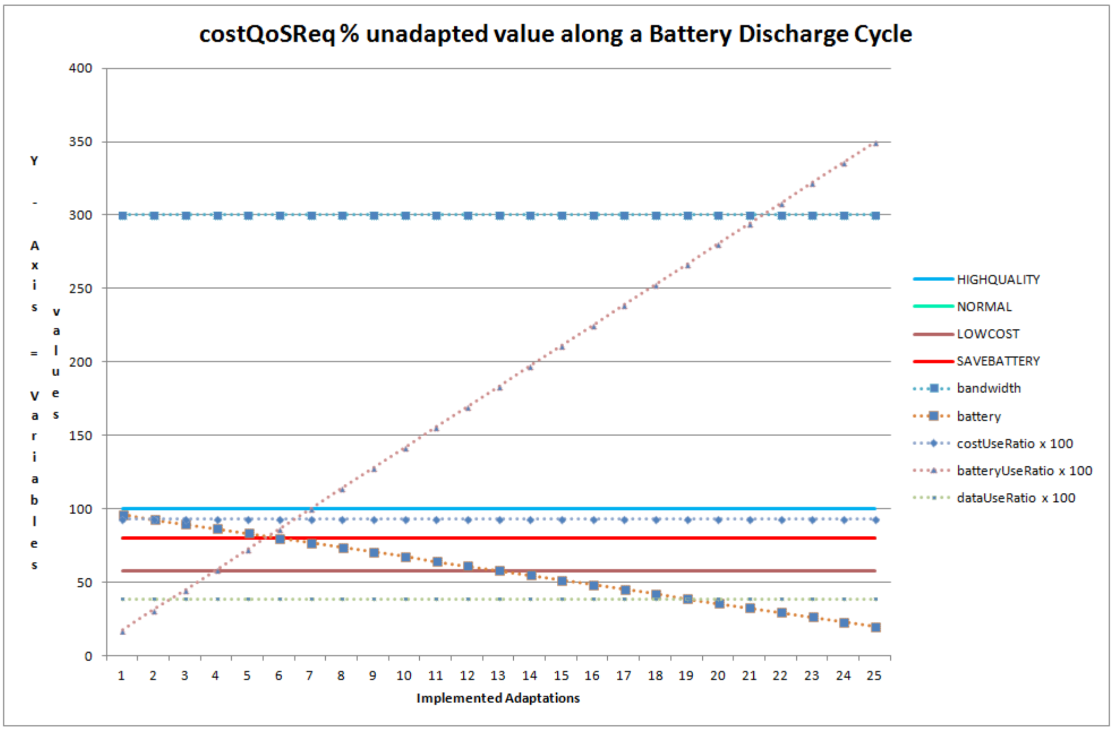
e) Constraining Influence of the increasing metric batteryUseratio > 100

In the figure 8-7 above, the adapted bitrate QoS Request (*bitrateQoSReq*) graphs under the 4 user preference modes is depicted, as calculated by policy execution at initial CSD along a Battery Discharge Cycle, with repeated regular bandwidth variation, but at decreasing battery level and increasing battery use metric of the Battery Mgmt LTPG. These metrics impose gradual reduction on the bitrateQoSReq values, under each user preference mode while all other context parameters are at fixed values.

2nd Parameter Chart: costQoSReq

In the following charts, the QoS-variable *costQoSReq* is depicted along a full Battery Discharge Cycle, as derived by policy execution under the Dataseries_1 context model, with fixed bandwidth and all other context parameters the same, under each of the 4 user preference modes (UPs).

Figure 8-8 – SR costQoSReq Graphs/UP without bandwidth variation Dataseries_1: Context data series within a full Battery Discharge Cycle



Remarks It is evident that every UP, even under all other context parameters the same, specify different adaptation behaviour to the *costQoSReq*. Only three graphs are shown, since for *costQoSReq* UP = "Normal" or "SaveBattery" gives the same results. It is noted though the clear difference in successively decreasing levels for costQoSReq from the "HighQuality" UP (blue line) to the "Normal" or "SaveBattery" UPs (red line coverung the green line) and to the bown line corresponding to UP = "LowCost".

Adaptation influence of User Preference Mode selection -- Modifying the SR QoS-variables by selecting a different user preference mode under the same other context conditions, typically leads to discovery of a different service that modifies application performance quality, MBs downloaded and cost. In the following charts in Figs. 8-9. 8-10 and 8-11 with CSD sessions and adaptations along a battery discharge cycle, the difference in battery drop, in rate of MBs downloaded, or in the cost of binding on cloud services, is clearly demonstrated for alternative simulated runs on the same other context data.

Figure 8-9 – Battery Level Graphs per User Preference Usage Model USGM1 within a full Battery Discharge Cycle

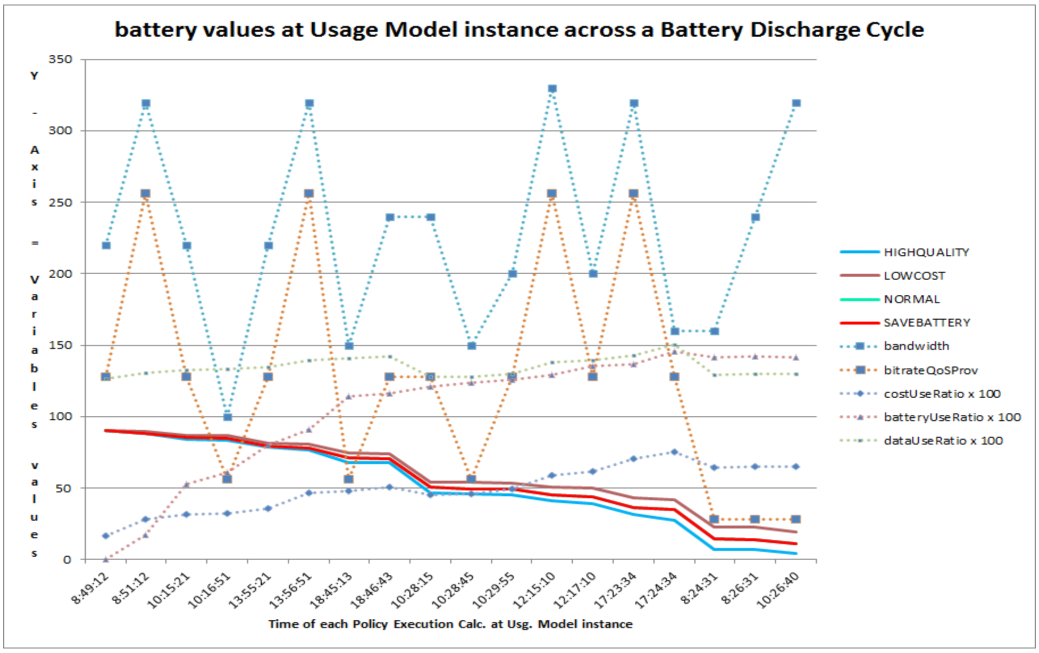


Figure 8-10 – creditUseRatio over Monthly Context Model – 4UPs Dataseries_6: 3LTPG_3CEC_USG1M_All24_4UP

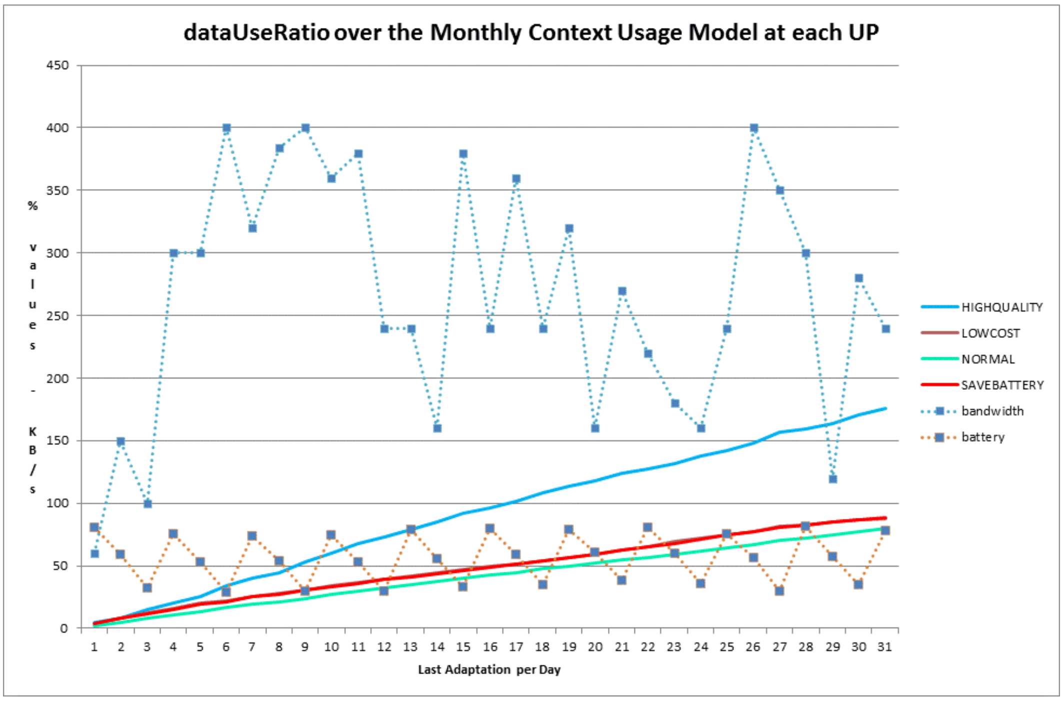
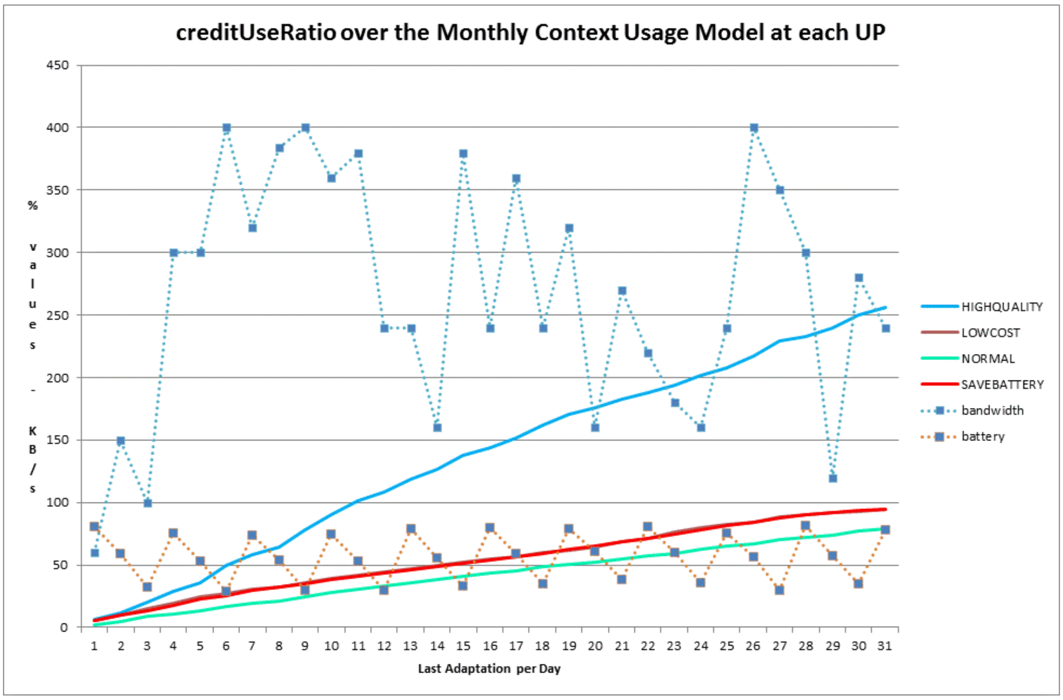


Figure 8-11 – creditUseRatio over the Monthly Context Usage Model – 4UPs Dataseries_6: 3LTPG_3CEC_USG1M_All24_4UP



2. Charts with adaptations along the full Monthly Context Usage Model

Results along the Monthly Usage Model on dataUseRatio & creditUseRatio

The following two charts: 8-10 and 8-11 depict the cumulative variables:

- *dataUseRatio* in % of the data allowance (which multiplied by 4000 MBs yields the downloaded MBs evolution)
- creditUseRatio in % of the credit allowance (which multiplied by 5 GBPs yields the cost evolution within the month in GPBs/month),

but with values derived over a full run of the monthly context usage model for each of the 4 UPs. These charts serve for demonstrating quantitatively the difference in adaptation behavior at different user preference modes on cumulative parameters over a monthly period. The percent metrics dataUseRatio and creditUseRatio have been used instead of the monthly parameters for MBs and cost, for chart scale reasons. Note the max values at end of month per UP.

User Pref	**HighQuality**	**Normal**	**LowCost**	**SaveBattery**
dataUseRatio%	174%	87%	88%	79%
MBs	6196	2698	2708	2360
creditUseRatio%	256%	94%	74%	94%
Cost GPBs	12.8	4.7	4.0	4.7

Results along the Monthly Usage Model on bitrateQoSReq & costQoSreq

The following two charts: 8-12 and 8-13, also depict the basic QoS variables *bitrateQoSReq* and *costQoSReq* respectively, but with values derived over a full run of the monthly context usage model for each of the 4 UPs. The bandwidth variation depicted is the one of the monthly context usage model.

In the two charts 8-12 and 8-13 below, it should be noted that *battery* level and the monitoring metrics depicted: *creditUseratio, creditRefRate, dataUseratio* and *dataRefRate* correspond only to the “Normal” UP. It is evident that their evolution within the monthly usage model would be different under each UP, but as the visibility on the chart would be impaired if more metrics were depicted, only the ones under UP = “Normal” are demonstrated.

For *bitrateQoSReq* in Chart 8-12 – It is clear from the chart that all previous remarks (a) to (d) are valid also on the monthly usage model. The blue line corresponding at UP = HighQuality also indicates the application of constraint of bandwidth on the *bitrateQoSReq* maximum values. Additionally, the influence of the Battery LTPG that imposes bitrate drop at the very low points in Very Low (< 30%) battery level (defined by the orange dotted line) at the first 3 and at the 9th battery discharge cycle is clear.

Figure 8-12 – bitrateQoSReq along the Monthly Usage Model at each UP Dataseries_4: 3LTPG_3CEC_USGM_All24_4UP GRAPHS

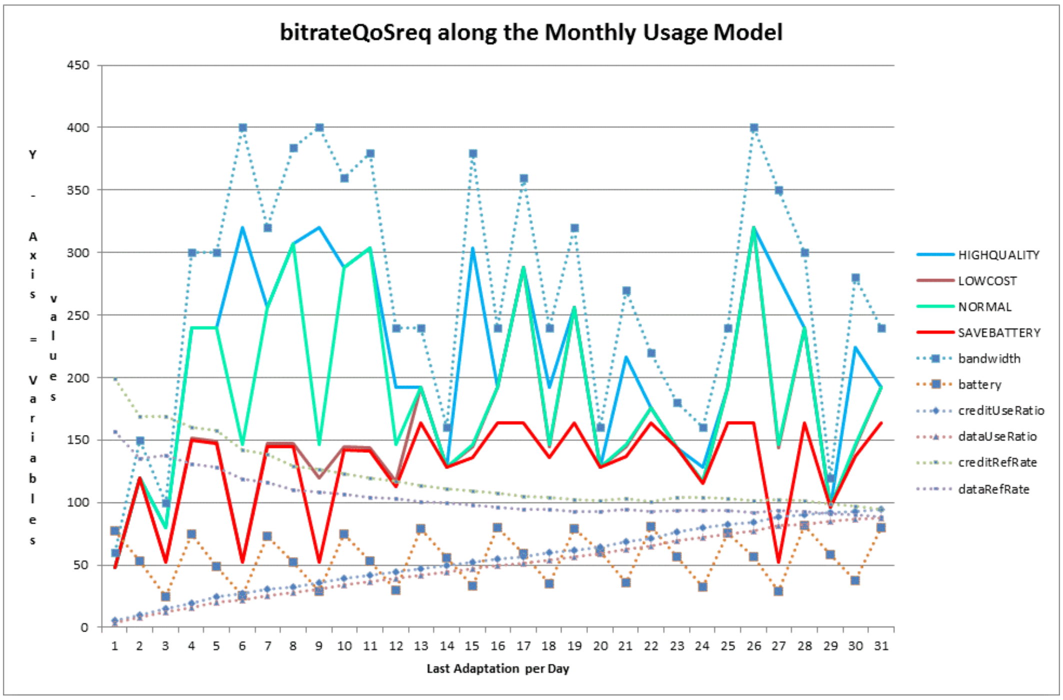
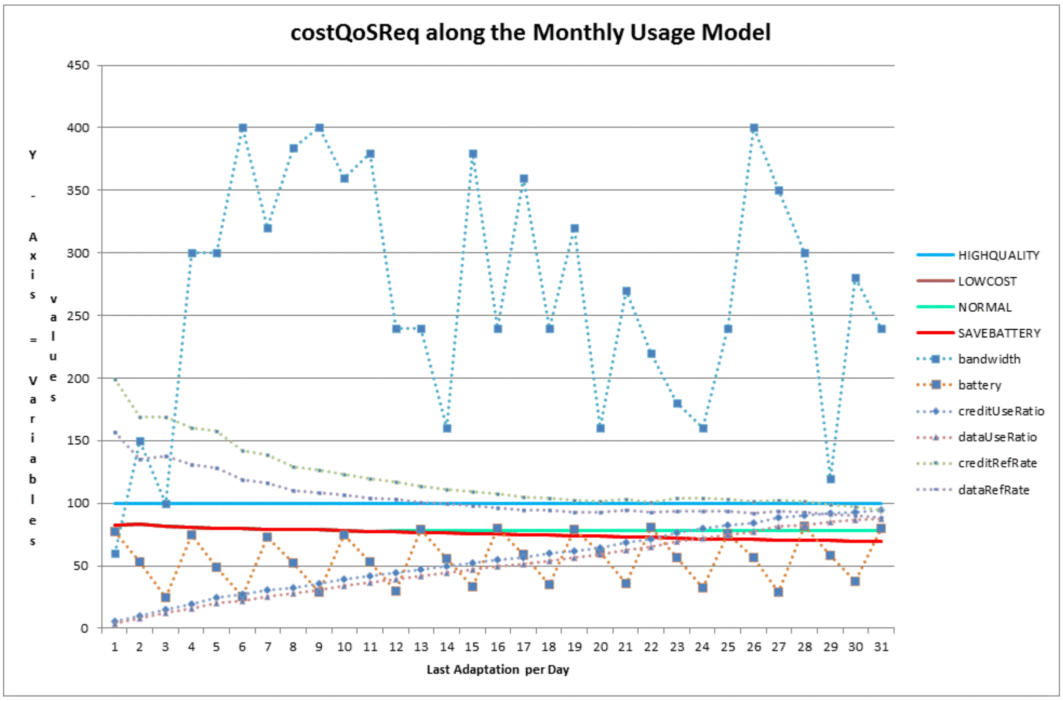


Figure 8-13 – costQoSReq along the Monthly Usage Model at each UP Dataseries_4: 3LTPG_3CEC_USGM_All24_4UP GRAPHS



At the same time, the difference in the adaptation behavior by each UP is fundamental on *bitrateQoSReq* and in consequence to the service with fitting QoS to be selected at SD for consumption.

For *costQoSReq* in Chart 8-13 – Since *costQoSReq* is not influenced in the policy by the external context (bandwidth or battery level), the only parameters that influence it are the current UP on the one side and on the other side the Credit LTPG metrics. As the *creditUseratio* increases (dotted blue line) or the *creditRefRate* (dotted green line) is above 100, there is also gradual reduction at the QoS parameter values.

8.4.3 Implications

We can make the following remarks on the charts in the previous figures, from fig. 8-3 to fig. 8-13:

1. Graphs conformity to the User Preference Metamorphic Relation

It is noted that the Metamorphic Relation referenced before is clear. Under same all other context conditions, the *bitrateQoSReq* and *costQoSReq* values conform to the expected inequality:

HighQuality value >= Normal value >= (LowCost value or SaveBattery value) (2)

Similar conclusions are drawn from Figs. 8-9 to 8-11, on adaptation dependent parameters, like cloudMBs, cloudCost, or battery drop within a battery discharge cycle. This is a testability feature of the User Preferences Model that allows qualitative verification of the adaptation behavior on test suites or through charts, on whether relation (2) is respected.

2. User Preference-dependent distinct alternative Adaptation Behavior.

In all charts, a clear differentiation of behavior is evident. It is a great advantage to enable the user, easily and on-the-fly, to select a UP and have the appropriate adaptation behavior in case of constrained resources through “SaveBattery” or “LowCost”, full technically attainable performance at other times through “HighQuality”, or the standard high performance through the “Normal” UP.

In all charts of this subsection with overlapping graphs of adaptation-related parameters, either QoS-variables (like costQoSReq or bitrateQoSReq) or adaptation-dependent (like MBs, cloudCost, battery drop), it is clear that the change of user preference mode does affect the adaptation behavior of the system.

3. Enabling the User to Fine-tune the Adaptation Behavior

Finally, although not directly referenced in the charts considered, through the UPI, the user is empowered without the need for policy updates, to customize the adaptation logic on the LTPG goal values, setting for instance the value for battery Very Low Level, or credit allowance, or data volume allowance, etc. This is a useful feature of the Volare approach and is valid for all mobile applications assigned SR DCAA at CSD sessions.

8.5 Introducing LTPGs

RQ4: Would adding in the policy quantitative long term goals over finite horizons enable the user to delegate to the adaptation logic long term resource and/or quality management?

A Long Term Performance Goal (LTPG) introduced by this work, constitutes a particular type of adaptation-rules cluster in the adaptation logic, because it reasons not only on the current context and the current application session but on a stochastic sequence of many recurrent sessions within a finite temporal horizon, by the end of which a quantitative goal should have been achieved, and then a new LTPG episode starts (see §6.5 for a more detailed description).

For enabling the policy developer to introduce LTPGs in the adaptation logic in a systematic and not in a custom-made manner, requires:

- supporting functionality by the middleware for storing relevant information on the past sessions within the LTPG horizon, i.e. a data base,

- supporting functionality by the middleware for a minimum of statistic functions that allow information retrieval from the database and statistical processing and monitoring of periods over which the statistic functions are applied (from the most simple of calculating the new sum of a parameter at each policy execution to more wide statistic functions),
- the APSL capability to allow the developer to declare and use in the policy of metrics derived by specified statistic functions on the database data over s specified horizon,
- a design template on adaptation reasoning for building and introducing the LTPG cluster of adaptation-rules in the policy.

The LTPG concept and design model, to the best of our knowledge not encountered – at least in a non-custom but systematic manner -- in the relevant literature on mobile middleware, is one of the fundamental contributions of this Thesis.

It entails a development cost for endowing the middleware and the APSL with the capabilities referenced and then developing LTPGs in the adaptation logic. Yet, it allows the user to delegate supervisory tasks to the adaptation logic, like: monthly credit or data volume allowance or battery level Mgmt, not simply in the sense of monitoring performance or resource use metrics, but of actively intervening to adapt the system behaviour at the successive CSD sessions, transparently to the application and the user, so that the set goal tends to be achieved by the end of its horizon, if the related applications use does not exceed certain limits.

Concerning RQ4, on whether setting LTPGs over finite horizons enables the user to delegate long term resource or quality management to the adaptation logic, the following examples will demonstrate if and to what extent such tasks on a sequence of CSD sessions may be delegated to the policy logic.

In the case study adaptation logic, three LTPGs are set in the global policy, supported also by the video-streaming application policy (see §7.4 and §7.5):

- the Monthly Credit Allowance LTPG, with goal not to exceed the contractual 5 GBP/month by the MNSP on MBs from cloud services, adapting the *costQoSReq* QoS-variable;
- the Monthly Data Volume LTPG, with goal not to exceed the 4000 MBs/month on data downloading through the MNSP on cloud and non-cloud services, adapting the *bitrateQoSReq* QoS-variable;

- the Battery Level LTPG, with goal to retain by the end of each discharge cycle at least 20% battery level, adapting the *bitrateQoSReq* QoS variable.

8.5.1 Simulation Setting

To answer RQ4, several simulated runs of the device operation are derived on different monitoring scenarios, alternative of the monthly context usage model already described in §7.2. five alternative monitoring scenarios on the monthly context usage model are designed on the following parameters with all other parameters intact: priceMax, fixedBW, no constraints:

- Fig. 8-14 with priceMax = 0.0024 GBPs/MB, at the context usage model, it remains as the standard case study reference.
- Fig. 8-15 with priceMax = 0.0016 GBPs/MB, at the context usage model,
- Fig. 8-16 with priceMax = 0.0032 GBPs/MB downloaded from cloud services,
- Fig. 8-17 with fixed bandwidth of 400 KB/s at the basic *priceMax*: 0.0024.
- Fig. 18 with the original context usage model, but with adaptation-rules on the LTPGs overridden by setting wSdat = wScost = wSres = 0, so that only the bitrateQoSReq – bandwidth constraint is followed.

8.5.2 Result Analysis

The Table 8-5 on the simulated outcomes concerning the three LTPGs on these five monitoring scenarios run over the whole monthly context models, shows that the goals have been achieved except at extreme usage cases, as in Fig. 16 (33% higher priceMax), or in Fig. 8-17 (fixed maximum bandwidth 400 KB/s) and Fig. 8-18 which serves for upper bound reference since the LTPG adaptation-rules are overridden.

LTPG failures have been noted on the Credit LTPG (in Figs. 8-16 to 8-17) only and of small deviation. The other 2 LTPGs seem to have been well defended. Only in the extreme scenario of Fig. 8-18, where no LTPGs were in action, the limits have been exceeded.

All three LTPGs are successful in the standard scenario of Fig. 8-14. Fig. 8-15 has a priceMax value 33% smaller than the case study value, leading to an inversion: instead of the Credit LTPG being the bottleneck, in this case the Data Volume LTPG risks of getting out of limit.

Table 8-5 also details the time percentage each service has been bound, from the one with lowest price & QoS (service1) to the ones with successively higher price & QoS (service4). In all scenarios same total monthly duration on the cloud.

Table 8-5 – Performance Metrics for each Monthly Monitoring Scenario

Parameter	Units	Fig. 8-14	Fig. 8-15	Fig. 8-16	Fig. 8-17	Fig. 8-18
Days duration	d	**31**	31	31	31	31
Cloud duration	s	**42999**	42999	42999	42999	42999
CSD sessions		**154**	154	154	154	154
Adaptations		**217**	217	217	217	217
priceMax	GBPs	**0.0024**	0.0016	0.0032	0.0024	0.0024
creditUseratio	%	**95.5%**	75.45%	103.16%	105.85%	254.99%
cloudCost	GBPs	**4.78**	3.77	5.16	5.29	12.75
dataUseratio	&	**87.8%**	98.87%	76.9%	91.10%	174.44%
cloudMBs	MBs	**3510**	3955	3074	3644	11077
Battery Min	%	**30.96**	29.42	32.9	37.62	18.44
Battery Min Avg	%	**26.21**	25.60	26.9	32.06	12.53
service1	%	**31.6%**	9.5%	45.4%	41.8%	4.0%
service2	%	**51%**	67%	44.7%	45.4%	26.2%
service3	%	**15.1%**	21%	8.4%	3.4%	47.4%
service4	%	**2.3%**	2.1%	1.6%	9.5%	27.4%

Figure 8-14 – credit & dataUseratio along the Monthly Usage Model Dataseries_4: 3LTPG_3CEC_USGM&4UPs_AllSWCs_24

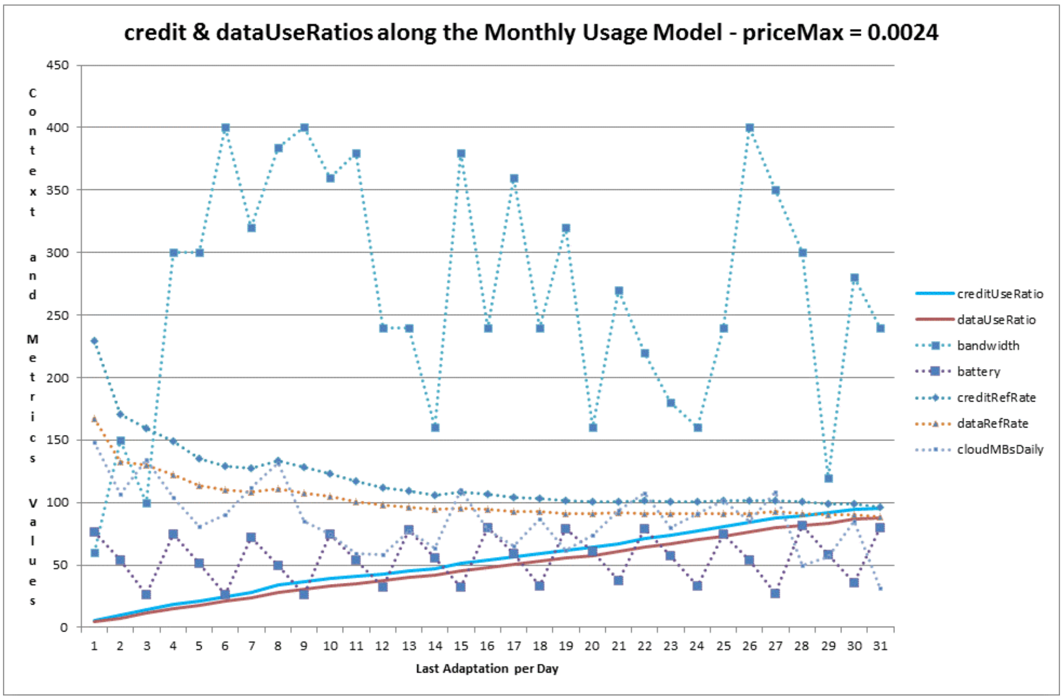


Figure 8-15 – credit & dataUseratio along the Monthly Usage Model Dataseries_5: 3LTPG_3CEC_USGM&4UPs_AllSWCs_16

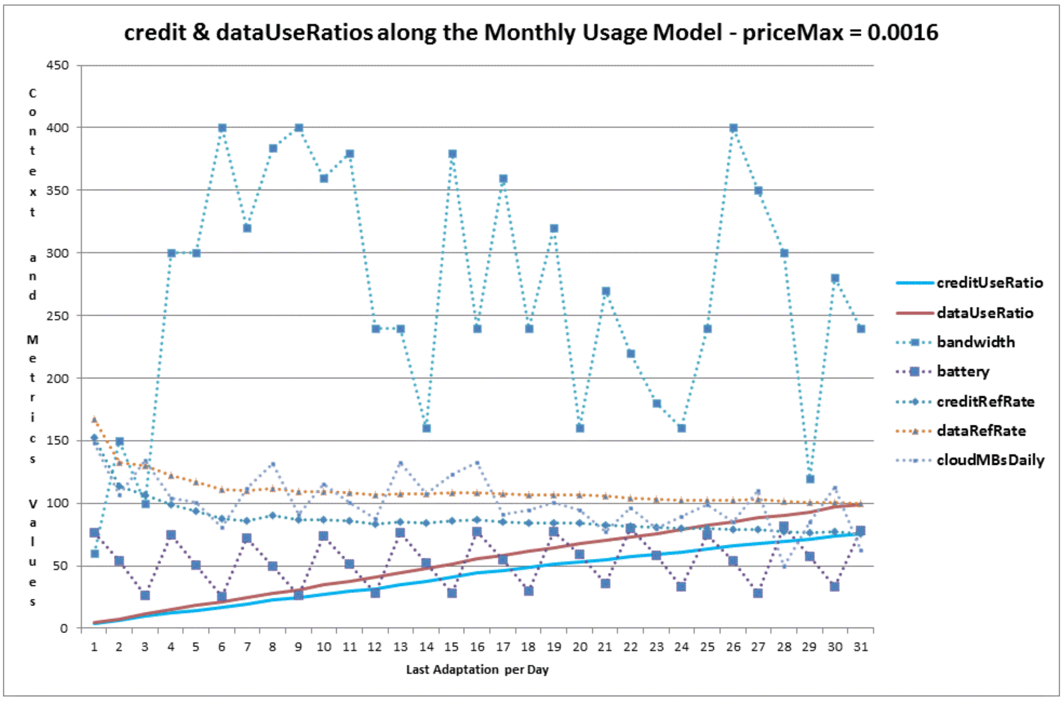


Figure 8-16 – credit & dataUseratio along the Monthly Usage Model

Dataseries_6: 3LTPG_3CEC_USGM&4UPs_AllSWCs_32

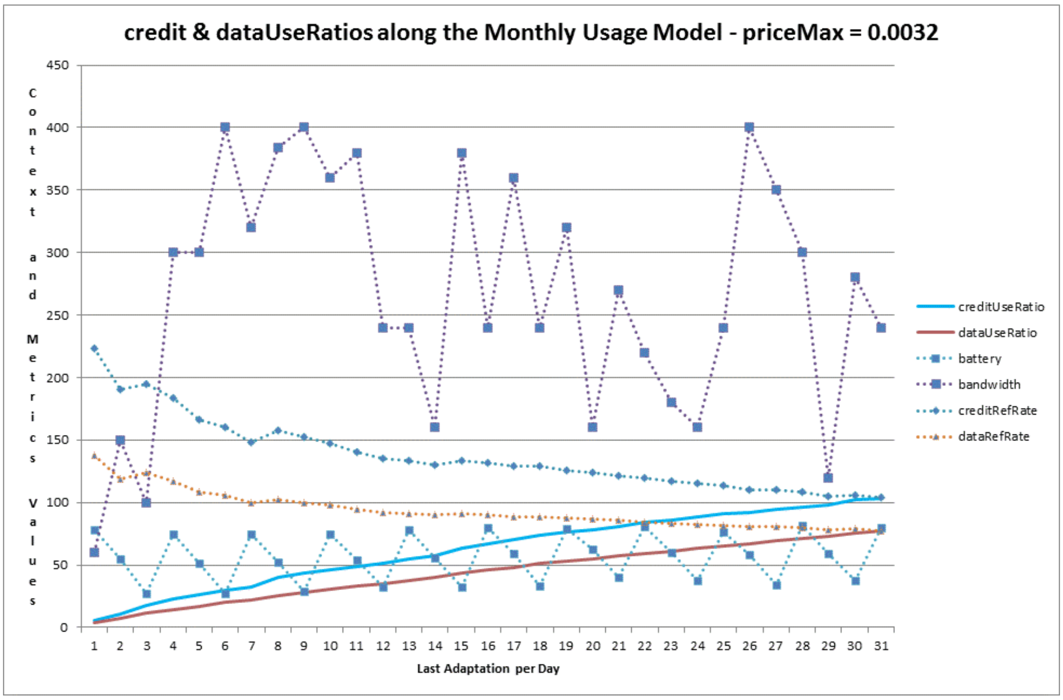


Figure 8-17 – credit & dataUseratio along modified Monthly Usage Model

Dataseries_6: 3LTPG_3CEC_USGM&4UPs_FxdBW400_All24

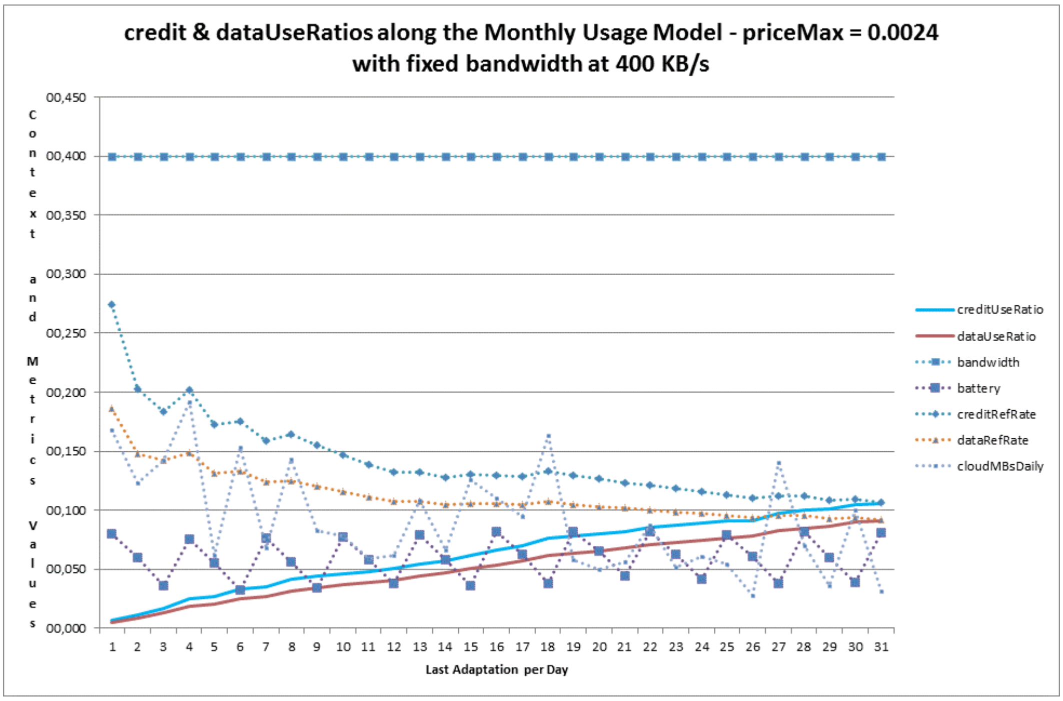
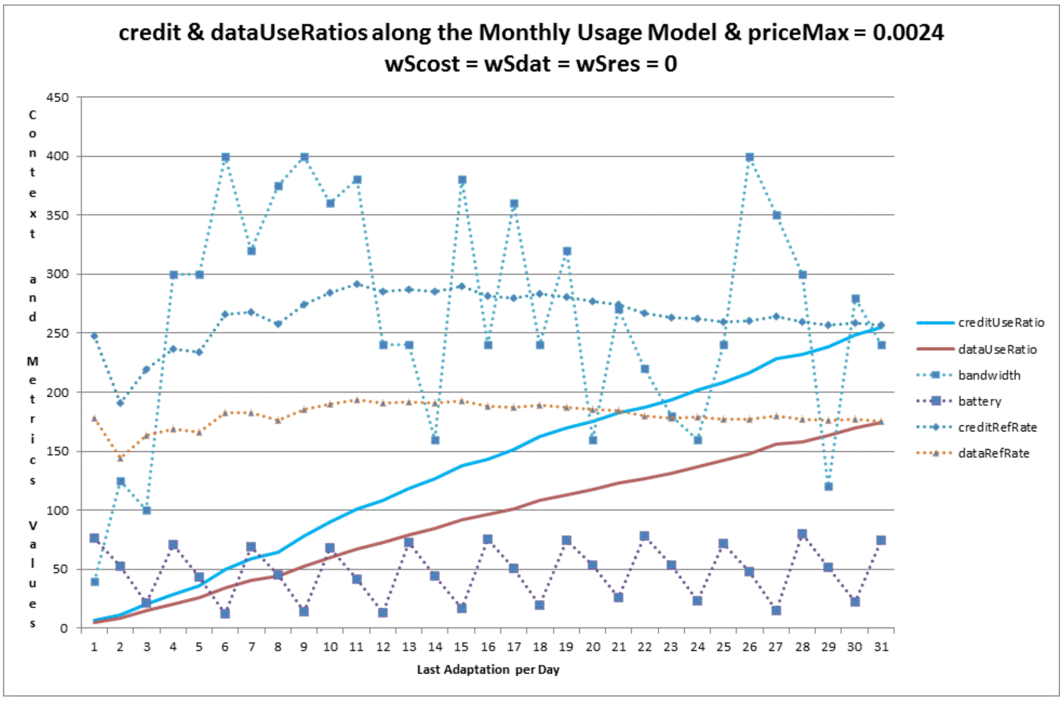


Figure 8-18 – credit & dataUseratio along the Monthly Usage Model

Dataseries_6: 3LTPG_3CEC_USGM&4UPs_AllSWCs=0_24



RQ4a -- Can the user delegate reliably the supervision on such LTPGs to the adaptation logic?

The answer in this RQ, as the Table of results 8-5 indicates, is that there may be extreme scenarios where the duration on CSD applications and/or the contextual situations may exceed the limit goal. For instance the scenario in Fig. 8-16 has a priceMax value 33% higher than the case study normal value. Similarly the scenario in Fig. 8-17, with fixed maximum bandwidth presents the upper limit at the monthly duration of CSD sessions considered.

It should be noted that the policy examined in the case study neither forbids the user to continue utilizing the applications, nor warns her even if the limits tend to be exceeded. The important fact is that the middleware, guided by the adaptation logic, monitors the goal metrics, adapts to keep the system on track as much as possible and even in case of LTPG failure, it reduces the deviation from the target. We note in Figs. 8-16 and 8-17, that even with 33% higher *priceMax* value (0.0032 instead of 0.0024) than the basic value, or with steady maximum bandwidth, violations are small in size. On the other side, in Figure 8-18, a policy with only *bitrateQoSReq* adjustment to *0.8*bandwidth*, the resource use levels on the same context usage model attained were extraordinary, of the order of 254% for credit and 174% for data volume. Consequently and in relation to the threshold values set in the policy, the adaptation logic achieves very efficient management of the LTPG limits, in comparison to shallow adaptation examples.

8.5.3 Implications -- Policy Self-Optimization on LTPGs

RQ4b: Can the developer establish policy self-optimization based on usage history for long term goals over finite horizons?

In addition to the possibility for relieving the user form several supervisory tasks and delegating them to the adaptation logic, LTPGs over finite horizons offer the potential for policy self-optimization in order to better achieve their goals in the future, based on the experience gained. This can be managed by certain supervisory level adaptation-rules that evaluate each LTPG at the end of its horizon for failure or success, or even on how well it performed. For instance, the goal of not exceeding the credit allowance may have been achieved by a large margin, but it was gained by systematically binding to very low QoS services and an optimization of the policy is necessary. Since the middleware has already in place usage data database Mgmt and a minimum of statistic functions support, evaluation of successes or failures at the end of each episode horizon LTPG may lead to usage-based improved monitoring and adaptation metrics that can be developed by policy self-optimization.

Policy Self-Optimization Example

As a simple example in the case study policy, for each LTPG there is an initial percentage value “VLowLevel” that is used to define the High – Low or VLow resource availability for each LTPG and guide the intensity of adaptation. For credit and data volume LTPGs the initial value of creditVLowLevel = dataVLowLevel = 10%. By increasing the value of the parameter VLowLevel, the context sub-domain where the assessment will be Low or VLow increases and the action taken leads to slower parameter increase.

In the case study global policy, three Criteria – Subpolicies at the 1st CEC, one for each LTPG, check at the end of each month on the average success or failure of the corresponding LTPG. In case of failure of one LTPG, the corresponding VLowLevel parameter is immediately increased, so that at the next monthly episode it will defend the goal more efficiently. In the monitoring scenario in Fig. 8-16, at priceMax = 0.0032 GBPs/MB an LTPG failure occurred at the end of the month (see Table 8-5).

The monitoring scenario was run for two consecutive months with identical context data series (the monthly context usage model). As Fig. 8-19 shows, at the end of the second month, the Credit LTPG had succeeded, because the policy-based parameter creditVLowLevel has increased at the beginning of the 2nd month from 10% to 19.9%, and the result for creditUseratio has been 98.3% instead of 103.16%.

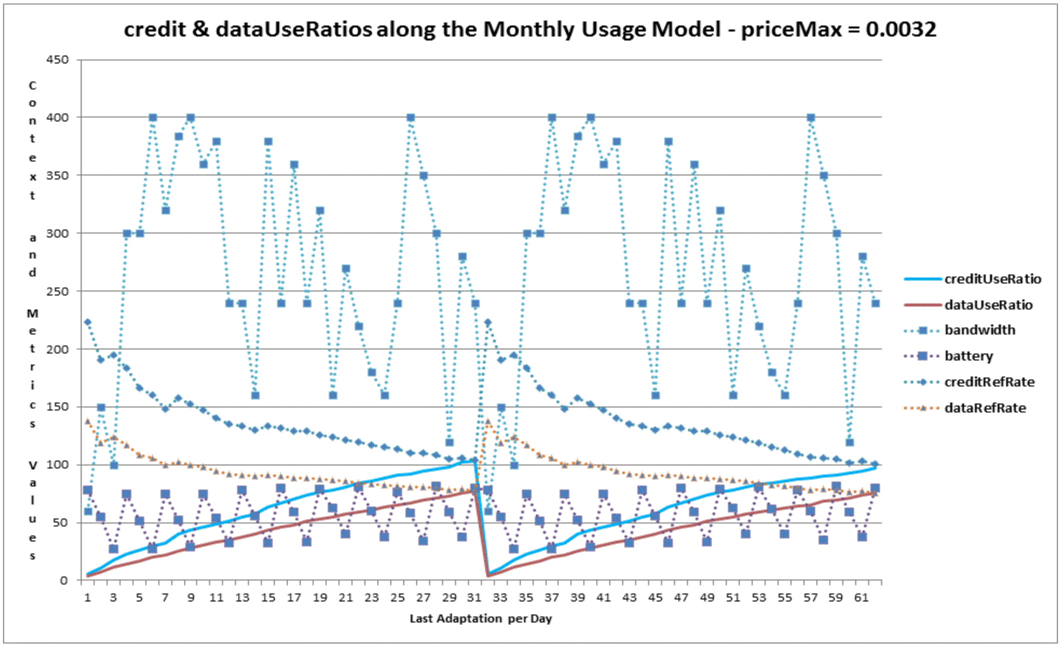
This is a simple but demonstrating example of employing the added features introduced in the APSL, the middleware and the adaptation reasoning methodology for supporting LTPGs, to further explore policy self-optimization on the LTPG performance through learning from successes or failures and drawing conclusions from the usage pattern. Note that this self-optimization takes place transparently to the application and the user at horizon-level time scale.

For this task, supervisory level adaptation-rules are specified, at the CEC have been specified that adapt the algorithm threshold parameters: “Prefixed” or “Usage-based” VLowLevel parameter value, in the adaptation-strategies that evaluate LTPG availability level and impose (or not) adaptation action (reducing the QoS levels at next adaptation). The chart 8-19 depicts the sequence of adaptations on the same monthly context model repeated twice.

Example of Policy self-optimization at the next horizon after LTPG failure

Figure 8-19 – credit & dataUseratio along the Monthly Usage Model

Dataseries_6: 3LTPG_3CEC_USG2M&4UPs_AllSWCs_32



8.6 Reducing the combinatorial explosion

A great challenge at adaptation policy development is the combinatorial increase in the number of adaptation-rules to specify adaptation behaviour when there is high variability in both the number of Variation Points (VPs) and the number of mutually exclusive variants (MEVs) of each VP [101]. If k is the number of VPs of a system and nVPi is the number of variants of VPi, then the number of adaptation states tends to increase with the product in relation (1):

Nstates = nVP1 x nVP2 x … x nVPk (1)

Every adaptation state of the system requires a set of adaptation-rules to specify adaptation, at least one state-specific rule on one adaptation-variable. In the weight-based adaptation reasoning technique, ways are explored for by-passing the combinatorial explosion mainly from behavioral variability – in the scope of the Volare approach -- on cross-cutting concerns like optimal performance and resources use. By behavioral variability in this work is meant parameter adaptation mainly on quantitative parameters, in contrast to structural or variability.

The idea consists in building the adaptation-rules concerning behavioral variability not on a contextual “situation” basis, but on wider context sub-domains and have multiple rules expressing different adaptation interests to be activated at any given context simultaneously on the same adaptation-variable and define the adaptation by participative weighted contribution.

8.6.1 Overlapping Sub-Policies Policy Architecture

RQ5: Would utilizing a policy architecture using multiple simultaneously activated adaptation rules per adaptation-variable allow easy incorporation of multiple adaptation-concerns in the policy?

A main characteristic of the Volare approach and specifically of the APSL consists in establishing a policy architecture that allows multiple simultaneously activated adaptation-rules per adaptation-variable. In contrast, the also rule-based Action-based adaptation reasoning technique, allows only one rule per adaptation-variable to be selected at policy execution.

The aforementioned characteristic of the Volare approach allows specifying adaptation on multiple simultaneously activated rules, and permits the building of the adaptation-rules in sub policies, where each one represents a major adaptation interest, called in this work adaptation-concern. In this way, the adaptation policy may be built in sub-policies, each serving an adaptation-concern.

For instance, in the case study policy, 6 major adaptation-concerns were adopted as covering the adaptation-space. As a consequence, the adaptation policy is built in 6 collections of adaptation-rules (the adaptation-strategies), with each one serving its adaptation-concern: generic operational constraints (Sgen), battery Mgmt (Sres), credit Mgmt (Scost), data volume Mgmt (Sdat), performance Mgmt (Sperf), disruption Mgmt (Sdat). This breakdown of the adaptation object in almost independent adaptation-strategies. facilitates the task of the policy developer. The balance for the simultaneously activated rules of the adaptation-strategies at any context on the same adaptation-variable, is established by the Weight Assigning Strategy.

8.6.2 High Behavioral Variability while reducing the Combinatorial Explosion

RQ6: Can we enable policy authoring of adaptation policies with high behavioural variability while reducing the combinatorial explosion by using multiple adaptation rules per adaptation-variable?

A main characteristic of the weight-based approach consists in building the adaptation policy in adaptation-strategies, with each one representing the rules that serve an adaptation-concern.

As described in section 6, we distinguish the Variation Points (VPs) in:

- structural or algorithmic VPs, each of which has a number of mutually exclusive structural or algorithmic variants that define the alternative mutually exclusive system configurations (MESCs),
- behavioural variants are sub-collections of rules that can be simultaneously activated on the same adaptation-variables.

Typically one behavioural variant from each behavioural VP is selected at each context instance, leading to multiple selected adaptation-rules on the same adaptation-variables from all relevant BVs.

Example from the Case Study on the Video-Streaming Application

1. Algorithmic or Structural VPs & MESCs

Suppose, as described in §7.4 and 7.5 in case study, that the adaptive system has 3 algorithmic VPs, each with 2 mutually exclusive algorithmic variants:

- Credit LTPG: prefixed or usage-based threshold parameters
- Data Volume LTPG: prefixed or usage-based threshold parameters
- Battery LTPG: : prefixed or usage-based threshold parameters

This analysis distinguishes 3 VPs of algorithmic character, each with two variants, and consequently the number of alternative mutually exclusive configurations (MESCs) = $N_{\text{MESCs-COMB}}$ = 2 x 2 x 2 = 8.

2. Behavioral VPs and BVs

Suppose that for each MESC there are in total 4 behavioral VPs on cross-cutting concerns, each with 3 mutually exclusive behavioral variants (MEBVs):

- Battery Mgmt LTPG VP with High – Low – VLow
- Monthly Credit Mgmt LTPG VP with High – Low – VLow
- Monthly Data Volume Mgmt LTPG VP with High – Low – VLow
- Performance Mgmt VP with the 4 alternative user preference MEBVs (HighQuality - Normal – SaveBattery – LowCost)

with all the MEBVs of each behavioral VP covering the valid context domain.

In total there are: $N_{\text{BVs-COMB}}$ = 3 x 3 x 3 x 4 = 108 behavioral adaptation states, corresponding to each of the 8 MESCs of algorithmic character.

This leads to 8 x 108 = 864 different adaptation states.

In the Action-based technique, with a single rule selectable at policy execution at a situation for each adaptation-variable, the developer would have to design: $N_{MESCs\text{-}COMB}$ x $N_{BVs\text{-}COMB}$ = 8 MESCs x 128 behavioral adaptation states = 864 adaptation states. Each of these different adaptation states has to be specified by a collection of rules for each relevant adaptation-variable (at least one rule that is state-specific), defining the context sub-domain and the adaptation action.

In the Weight-based technique though, since under each MESC the MEBVs are compatible, the developer would have under each MESC, $N_{BVs\text{-}SUM}$ = 3 + 3 + 3 + 4 = 13 alternative behavioral adaptation states and for each one of them, would need to build a Subpolicy with rules for the relevant adaptation-variables.

In total for the 8 MESCs there would be instead of the 864 states, only: 8 x 13 = 104 alternative adaptation states, i.e. the combinatorial product of the MESCs times the sum of MEBVs: $N_{MESCs\text{-}COMB}$ x $N_{BVs\text{-}SUM}$, instead of the combinatorial product of both factors: $N_{MESCs\text{-}COMB}$ x $N_{BVs\text{-}SUM}$ needed Subpolicies.

Note that in both action- and weight-based techniques usually not all behavioral rules are MSC-specific. So many behavioral rules will be the same for more than one algorithmic variant or MESC, with possibly wider context sub-domains and only few variant-specific rules would need to specify MESC-specific behavior.

3. Policy designed in multiple Consecutive Execution Cycles (CECs)

RQ7: Can the use of multiple execution cycles in the adaptation policy allow for further reduction of the number of rules necessary to model complex adaptation behaviour?

If the weight-based policy design includes multiple CECs, then the stratagem on behavioral adaptation described in the case study in global and application policy (see §7.4 and §7.5) can be applied, assigning to auxiliary-variables the parameters for the “usage-based” vs, “prefixed” algorithmic variants. This is possible if the algorithmic variants refer to alternative threshold parameters or metrics or values to be selected at different context conditions, and which are required to be used in other rules (either in predicates or in adaptation actions).

Analytically, the following procedure is implemented at policy execution:

- At the first CEC part, the adaptation-rules selecting the algorithmic variants would be selected and calculated yielding, say in pseudocode:

  ```
  If conditions1 … Then batteryVLowLevel = “PrefixedBatVLL”
  Else batteryVLowLevel = “Usage-basedBatVLL”
  ```

- At the second CEC, instead of having alternative Subpolicies with "PrefixedBatVLL" and other with "Usage-basedBatVLL", through a policy-driven declared auxiliary-variable **batVLL**, there will be a single rule in each case with batVLL, which will have already been calculated at the 1st CEC.

Then in our example, instead of 128 Subpolicies we would need only 13 BVs + 8 MESCs = 21 Subpolicies, (if all MESCs were representing metrics or values – as in the 8 variants of the case study), which is a distinct policy "shortening" on equivalent adaptation.

The adaptation states to be specified are still 8 X 128 = 864 adapattion states

$N_{MESCs\text{-}COMB}$ x $N_{BVs\text{-}COMB}$ = 8 MESCs x 128 behavioral adaptation states = 864

Now in the example of the case study "Composite Policy", in Appendix A and B, there are 31 Criteria – Subpolicies and a total of 142 adaptation-statements, i.e. 142 adaptation-rules of the form: "If condition … Then action". As a consequence, there is a radical reduction in a Volare "Composite Policy" in the required number of adaptation-rules to specify the adaptation with 3 algorithmic VPs and 4 behavioral VPs, of totally 864 adaptation states.

4. Cost – Benefit on the Number of Policy Rules

We note that since in the weight-based technique there are multiple rules selectable from relevant adaptation-strategies, a weight-based policy may usually have up to 1 rule per adaptation-variable and per Subpolicy in the adaptation-strategy.

At the same time, the number of rules increases quasi-linearly with the sum of MEBVs: $N_{BVs\text{-}SUM}$ instead of the combinatorial size $N_{BVs\text{-}COMB}$ of behavioral adaptation states. When **3 x $N_{BVs\text{-}SUM} < N_{BVs\text{-}COMB}$ (3a)**

the advantage in the number of rules lies with the weight-based technique.

Especially if the advantage above mentioned in (3) may be used by designing the policy in CECs, then (3a) becomes:

3 x ($N_{BVs\text{-}SUM} + N_{MESCs\text{-}SUM}$) < $N_{BVs\text{-}COMB}$ x $N_{MESCs\text{-}COMB}$ (3b),

and the benefit is much greater.

The higher the behavioral variability, the more extensive is the benefit for our technique, since the rules required to be authored and the context sub-domains considered increase quasi-linearly with the sum of BVs instead of combinatorial increase in the action-based adaptation reasoning.

8.7 Key Differences of the Volare APSL to other APSLs

As analytically described in subchapters 2.4, 3.4 and 4.6 [18][22][34], there are three established main techniques on policy-based adaptation reasoning: the Action-based [7][58][69], the Goal-based [74][75] and the Utility-based one [18][19][84], although there are research attempts on hybrid forms of adaptation reasoning [68][77]. The key differences of the Weight-based Adaptation Reasoning Technique to the three other are emphasized below.

1. Multiple Selectable Rules on the same Head Predicate

The most important innovative characteristic of the VARE APSL, in contrast to all other Policy Specification Languages described in Chapter 2, consists in Conflict Resolution Directives allowing at policy execution all the highest priority matched rules to be selected, even more than one on the same head predicate. Thus, possibly more than one rule on the same adaptation-variable may be matched, selected and executed.

2. Participative Weighted Contribution mechanism –

In order to avoid inconsistencies the Volare APSL establishes Conflict Resolution Directives applying the Weighted Participative Contribution Rules for deriving the adaptation results from all the execution-values provided by the selected rules, based on the weight value of each selected rule: the weighted-average rule for numeric adaptation-variables and the majority rule on the sum of weights for the non-numeric adaptation-variables.

3. Multiple Consecutive Execution Cycles at Policy Execution

Another characteristic of the Volare APSL consists in optionally building the adaptation policy in more than one segment, each to be evaluated in Consecutive Execution Cycles. Consequently not all adaptation-rules are evaluated for matching and selection in one Policy Engine cycle, but in consecutive cycles specified in each Criteria-Subpolicy pair cycle declaration number. This characteristic establishes hierarchic policy by first calculating the architectural or operational main mutually exclusive modality variant(s) and in successive cycles the dependent adaptation-variables.

4. Building the Policy in Adaptation-Strategies

Building the policy not on the notion of a single rule selectable at any context instance and adaptation-variable, which is allowed without being the systematic practice, but in **adaptation-strategies** - each being a collection of the rules designed to serve an **adaptation-concern (a targeted adaptation viewpoint)** over the whole valid context-domain. The adaptation-concerns need to cover the whole adaptation space as described in subchapters 4.2 – 4.3 and by the methodology in subchapters 6.1 – 6.2. Thus every global or application policy composed of parallel smaller, independent, specific-goal oriented sub-policies that cover the whole of adaptation interests.

In the *Action-based situation-action methodology* [18][34], the developer considers each contextual “situation” and decides on the corresponding “action”. On the contrary, in the weight-based adaptation reasoning technique, every situation-action rule within an *adaptation-strategy* is built oriented to serve an *adaptation-concern* and may typically be overlapping with other rules on the same adaptation-variable serving other adaptation-concern(s), selectable jointly at policy to drive the adaptation.

Additionally, the design of the policy file in Volare as a synthesis of rules under each MECV takes a top–down approach. Firstly, by horizontally defining the scenario adaptation-concerns & adaptation-strategies, secondly, by identifying existing alternative feasible MECVs of structural or algorithmic character adaptation, and finally under each MECV, specifying behavioral adaptation by developing the corresponding rules within every adaptation-strategy over the whole valid domain.

8.8 Benefits of the Volare Approach

The Volare approach aims at assigning through a mobile middleware DCAA capability to the SR of independent mobile applications launching service discovery, (i) by identifying the most appropriate QoS terms under the current context and policy and adapting the SR, and (ii) at runtime monitoring the context and re-adapting the SR activating rediscovery – if context change justifies it.

The Volare approach provides benefits for the three system stakeholders: the global and the application policy and the mobile device user, outlined in the following paragraphs.

8.8.1 Advantages for the Adaptation Logic Developers

The weight-based adaptation reasoning technique innovative features like the identification of adaptation-concerns and building the adaptation policy logic in the corresponding adaptation-strategies, presents advantages especially in cases of high behavioral variability. Building a policy in the weight-based approach for a scenario that lies within its scope, delivers certain advantages in comparison to the conventional action-based approach.

1. Support of two-level policies

The Volare middleware includes a mechanism for automated merging of the the global policy for the middleware operation and the application policy for the application specific QoS requirements, in one “Composite Policy” driving the adaptation. All middleware supported context-variables, LTPG metrics and global QoS-variables are declared in the global policy.

This constitutes a significant advantage for the application policy developer, who only has to specify application-specific adaptation, conforming of course to the global policy requirements and goals. The application policy may make direct use of all declared variables in the global policy. These characteristics make the application policy developer’s task easier and lower the development cost.

2. The Volare configurable User Preferences Model

The Volare configurable User Preferences Model, supported by the UPI application to communicate dynamically user preferences changes to the middleware, is uniquely integrated in the policy structure not only through the user preferences context elements but also through the Strategy Weight Coefficients. It is very easy for the developer to configure the generic model and adjust it to the scenario, setting the default UCP values for the adaptation logic, while allowing the user to customize or fine-tune the adaptation behaviour.

3. Introducing LTPGs in the Policy

The Volare approach allows the developers to introduce Long Term Performance Goals in their policies, which otherwise would be an impossible task without extensive development if the middleware and the APSL does not support it, and have the possibility to easily review or modify them.

4. Testability of the Volare Adaptation Strategies

An advantage of the WB adaptation technique is the fact that it provides a great testability potential for each adaptation-strategy, either simply on top of the Generic Operational Constraints adaptation-strategy or in parallel to the other ones. The weight-based technique provides the opportunity through the UPI (or the policy) to just set to 0 the SWCs for one or more adaptation-strategies at the User Choices Profile (UCP) and have the relevant adaptation-strategy inactivated without modifying the policy file, just for testing and adaptation behavior verification reasons.

5. Reducing Context Hazards

As described in [55][71], CAAAs face context hazards for the adaptation process due to differences in the sensed context the information represented by propositional context-variables, because of asynchronous notifications of context-variables values. The Volare middleware eliminates this danger by supplying at each policy execution, only the last – most recently updated -- set of context data values for the adaptation values calculation. Thus, the Policy Engine execution prevents context hazards, by prohibiting asynchronous context updates at the Working Memory level.

6. Smaller Number of Rules with wider predicate context sub-domains

In Volare often the contextual "situation" is not defined by a context sub-domain and the selectable rule, but by optionally more than one selectable rules on wider and overlapping context sub-domains. For instance the rule on bitrateQoSReq representing the relevant battery LTPG Subpolicy and the rule representing the data volume LTPG Subpolicy, and the rule on the current user preference mode performance Subpolicy, do have wide and overlapping context sub-domains and jointly specify adaptation for this QoS-variable under the current context. This leads to the result that the predicates of rules in Volare are of wider and usually overlapping context domain coverage.

Thus typically a smaller number of rules in total is required for specifying adaptation than in an action-based equivalent policy , which requires adaptation-rules at each adaptation state (contextual situation), even though in Volare policy there may be more than one rule on the same adaptation-variable selectable at any context, depending on the size of behavioral variability considered.

8.8.2 Benefits of the Volare Approach perceived by the User

1. Adaptation Policy Customization Capability by the User

The user may easily customize the adaptation behavior so that the application/middleware behaves differently on developer set user-customizable parameters of the adaptation logic, without intervening in the policy. For instance, the user may decide to make the binding cost strategy stricter and/or set a more cautious attitude towards a resource like battery power. In Volare, it is easy for the user to modify through the UPI the default User Choices Profile enhancing the relative importance of the relevant adaptation-strategy, thus fine-tuning the adaptation behavior. Additionally the user may modify the LTPGs goal values or other user-customizable provided threshold parameter values.

2. Selection of Alternative User Preference Modes

The user may select a User Preference Mode, selectable on-the-fly at runtime by the UPI among several policy-based alternative adaptation behavior patterns corresponding to real time user mood or needs. In fact, every user preference mode constitutes a different adaptation behavior.

3. Relieving the User from Supervisory Tasks through LTPGs

Long Term Performance Goals over finite horizons introduced in the policy allow the adaptation logic to undertake device operation supervisory tasks and relieve the user from the burden to monitor on constraints like remaining battery level or monthly data volume or monthly credit for binding on the cloud and take relevant actions for reducing resource use. Besides, LTPGs in the policy may manage the task much more effectively and smoothly for the device operation at CSD sessions, than occasional decisions on the part of the user.

4. Savings at the cost of binding

From the charts in §8.5, it can be seen that both the application performance by (re)discovering the more appropriate QoS request values under a frequently changing context and the cost of binding on the cloud are optimized, versus an unadapted or only initially adapting application (static adaptation), especially for long-lasting applications like media-streaming, navigation, etc.

The improvement realized concerns not only application performance but also on the savings on the cost of binding to the cloud service with variable context, since it is annoying for the user to pay additional cost on cloud services when exceeding the monthly allowance.

8.8.3 Benefits in the mobile CSD applications performance

Evaluation, presented in Chapter 8, through RQs on the contributions of this project, using prototype results and simulations, has shown:

Application Performance Benefits

- Noticeable improvements in the performance of mobile applications launching CSD, in the sense that by adapting the SR to the optimal QoS, delays and energy-sink services due to bandwidth higher than required are avoided

- Optimal management of the device resources, as demonstrated by the Battery LTPG charts in 8.6
- Reduction in disruptions due to rediscovery through disruption minimization rules
- Prevention of binding degradation at runtime, by establishing dynamic context-aware adaptation at runtime
- Naturally, due to the domain nature, the efficiency results might be widely different depending on the case study, adaptation policies, context data sequence and the service provision options available to the CSP.

Eventual Benefits for the Service Provider

Additionally, if the economies of scale for a Service Provider on the cloud to its registered users are considered, by providing dynamic adaptation to the mobile applications through a DCAA middleware, it certainly sums up to considerable savings. Of course, on an individual basis, most depends on the network bandwidth variability for each specific user.

8.9 Evaluation Discussion

The evaluation, using prototype results and simulations, has shown noticeable improvements by the Volare policy-based service-level dynamic context-aware adaptation concerning CSD sessions of mobile applications, on several different dimensions that are synoptically outlined below.

As demonstrated in §8.2 a mobile application without context-aware service-level adaptation capability, can suffer significant setbacks in changing context. Long delays would be expected at a video-streaming session, both initially at the beginning of the VSD session as well as during play-back period depending on the bitrate to bandwidth ratio value, in cases where the requested video-encoding bitrate does not match the bandwidth by a safety margin. In the example in Table 8-2, the data volume downloaded during the CSD video-streaming sessions running a shallow adaptation simulation on the monthly duration usage data model, lead to results that were 174% higher that the MNSP contract data volume allowance vs. 87% for the full adaptation policy result. Additionally, when binding at high QoS request values in the same shallow adaptation manner, the cost of binding in the same case reached 255 % of the credit allowance vs. 94% in the full adaptation policy case. The cost of binding on cloud services would be high by selecting services at high QoS without control on price and the capability for the user to control the increasing cost of binding to the cloud and the cost by exceeding the MNSP data volume limit would be to reduce use of cloud services.

From the charts in §8.2 we can see that both the application performance by (re)discovering the more appropriate QoS Request values under a frequently changing context and the cost of binding on the Cloud are optimized, versus an unadapted or only initially adapting application (static adaptation), especially for long lasting applications like media-streaming. For instance the monthly average of min battery level value before recharge time was 18,4% vs. a LTPG target of min 20% (for eventual emergencies), in comparison to an achieved average value of 27.2% for the full adaptation policy run under the same monthly context data matrix.

Without dynamic adaptation, we would have a flat horizontal line corresponding to the unadapted QoS Request value (or the initial discovery value, without dynamic adaptation) for each QoS variable of the Service Request or for each Middleware Configuration variable. On the contrary with a full adaptation policy, note in Table 8-5 comparing alternative monitoring scenarios of Fig. 8-15 against Fig. 8-17, both on the same other context data but Fig. 8.15 on the typical monthly context usage model while Fig. 8-17 on fixed high bandwidth of 400 KB/s – a reference upper bound scenario. The standard reference case in Fig. 8.15 achieved by the end of the monthly period 95.5% of the credit allowance vs. 105.9% of the fixed high bandwidth case (still very near the goal value, under the circumstances) and data use ratio respectively 87.8% vs. 91.1% for the fixed high bandwidth case.

Continuing from a user-perspective, the mobile user is able to enjoy satisfactory performance under variable context and through the User Preferences Interface is entitled to select the User Preference Mode that best suites current, session-specific needs, when launching a mobile application on the Cloud, or customize the adaptation logic. As evidenced by the evaluation in §8.4, different QoS Request values are derived at each alternative user preference mode. Note that the improvement realized by user preference mode selection, not only in application performance but also on the savings on the cost of binding to the cloud service under variable context (mainly bandwidth, battery and user preference), may be significant for the user when viewed on a monthly or yearly basis. It certainly will incite the user to implement case-specific expenses control by selecting the appropriate user preference mode at each session, instead of the default ”Normal” Mode.

Similarly, evaluation in §8.5 on LTPGs introduced in the policy, demonstrates the great advantage to let the adaptation logic take care of the service discovery on battery or credit or data volume Mgmt, by selecting the optimal QoS at each case.

Naturally, due to the nature of the domain, the efficiency results might be widely different depending on the case study, adaptation policies, data series used and the service provisioning options available to the CSP/CSB.

Additionally, if we consider the economies of scale for a Service Provider on the Cloud to its registered users, by providing dynamic adaptation to the mobile applications through a DCAA middleware, it certainly sums up to considerable savings. Of course, on an individual basis, most depends on the network bandwidth variability for each specific user.

The above evaluation shows significant benefits in cost and resource costs by using dynamic adaptation and rebinding of Cloud service bindings.

On qualitative issues from the application developer point of view, evaluation in §8.3 showed that the development burden for the application developer is significantly reduced if all context and global QoS variables are already declared, as well as the global policy generic constraints and goals. All the developer has to do is specify the SR application-specific adaptation, having direct access to the context-variables or declared metrics of the global policy.

Another important issue deriving from the evaluation in §8.6, is the fact that the structure of the policy in multiple eventually simultaneously selectable at policy execution adaptation-rules per variable, allows for shorter implementations of high behavioral variability adaptation behaviors concerning multiple adaptation-concerns, compared to the conventional single rule approaches. This is because the Volare policy language enables the developer to develop separate sub-policies specific to each group of contextual data that can affect the adaptation process, and then bind them all together, instead of having to develop a full separate policy for each possible contextual situation. In this way the combinatorial explosion maybe by-passed, if the rule of thumb:

. When **$3 \times N_{BVs\text{-}SUM} < N_{BVs\text{-}COMB}$ (3a)**

the advantage in the number of rules lies with the weight-based technique. In the case study, with 142 adaptation-rules of the Composite Policy, 864 adaptation states were specified, as demonstrated in §8.6.2.

9 Conclusion and Future Work

In conclusion, the dynamic adaptation of service requests for cloud services in mobile systems is an issue that has yet to be sufficiently addressed by the current state-of-the-art. In this report, we present a solution to this problem via the Volare adaptation reasoning approach, which presents an innovative weight-based approach to adaptation reasoning. We implement this approach with the Volare middleware, a context-aware, client-side, dynamically adaptive middleware, offering DCAA functionality to the service request of applications residing above it on service discovery and binding on the Cloud, without interrupting or changing the code of the applications or the services in any way.

The overview and requirements of the project were presented in chapter 3. In the previous chapters 4, 5, 6, each of the three constituents of the Volare approach are presented and their implementation in the case study of chapter 7. Chapter 8 verifies that these constituents satisfy the requirements identified in subsections §3.4, §3.5 and §3.6 and the approach achieves its target as demonstrated by the quantitated results through prototype and simulations runs in §8.2 to §8.6.

9.1 Conclusions

The central contribution of this project is the “Volare approach”, a client-side policy-based adaptation scheme for implementing dynamic context-aware adaptation to the commercial service request (SR) of a mobile application to an application-selected cloud service broker or provider provisioning alternative services of the same content at different QoS levels.

As demonstrated in §8.2 a mobile application without context aware service-level adaptation capability, can suffer significant setbacks in changing context. Long delays would be expected at a video-streaming session, both initially at the beginning of the VSD session as well as during play-back period depending on the bandwidth value, in cases where the requested video-encoding bitrate does not match the bandwidth by a safety margin. In the example in Table 8-2, the data volume downloaded during the CSD video-streaming sessions running a shallow adaptation simulation on the monthly usage data model, lead to results, in extreme cases, that were 174% higher that the MNSP contract data volume allowance vs. 87% for the full adaptation policy result. In addition, when binding at high QoS request values in an unadapted manner, the cost of binding in the same case reached 255 % of the credit allowance vs. 94% in the full adaptation policy case. The cost of binding on cloud services would be high by selecting services at high QoS without control on price and the capability for the user to control the increasing cost of binding to the cloud and the cost by exceeding the MNSP data volume limit would be to reduce use of cloud services. These ofcourse are rather extreme cases where the available bandwidth was, on average, less than what's needed to receive the QoS levels requested by default. As connection stability increases, and the context changes decrease, the benefit of adaptation becomes significantly less. The evaluation still showed average battery savings of 8.8% on a typical usage model, as well as cost savings closer to 5% on more modest context changes.

The project aims to support the discovery of the most appropriate service found under the current context and the adaptation logic requirements, by performing parameter adaptation on the QoS levels of the service request, then evaluating the offered service QoS levels, and either binding to it or adjusting the SR and launching re-discovery. The case study for this project used Cloud services.

The Volare approach presented in this Thesis is composed of three integrated constituents each of which participates for the contributions of this work, described in §1.5 to be employed successfully.

The mobile DCAA support middleware that implements the adaptation functionalities itself.

The Adaptation Policy Specification Language (APSL) that allows the developer to specify the adaptation behavior required.

The weight-based adaptation reasoning technique (WBART) that guides the adaptation policy development process.

The requirements of each constituent are outlined in §3.3 -- §3.4 -- §3.5 and are analyzed in detail in chapters 4, 5 and 6 respectively.

The **Volare middleware** monitors the context of the mobile device and then proceeds to dynamically perform parameter adaptation on the QoS levels of the service request accordingly. This enables the application to choose the most appropriate services conforming to the current needs of the client. Dynamic monitoring of the QoS levels and available resources allow for rebinding during runtime, which can increase system reliability avoiding long delays and reduce resource costs. Dynamic monitoring of the QoS levels and available resources will allow for rebinding during runtime.

Evaluation, presented in Chapter 8, using prototype results and simulations, has shown noticeable improvements in the performance of mobile applications launching CSD, in the sense that by adapting the SR to the optimal QoS, delays and power draining services due to bandwidth higher than required are avoided.

From the charts in §8.2, it can be seen that both the application performance and the cost of binding on the cloud are optimized by (re)discovering the more appropriate QoS request values under a frequently changing context, versus an unadapted or only initially adapting application (static adaptation), especially for long-lasting applications like media-streaming, navigation, etc. The improvement realized concerns not only application performance but also on the savings on the cost of binding to the cloud service with variable context. With a full adaptation policy, we see in Table 8-5 a comparison of alternative monitoring scenarios of Fig. 8-15 against Fig. 8-17, both on the same other context data but Fig. 8.15 on the typical monthly context usage model while Fig. 8-17 on fixed high bandwidth of 400 KB/s – a reference upper bound scenario. The standard reference case in Fig. 8.15 achieved by the end of the monthly period 95.5% of the credit allowance vs. 105.9% of the fixed high bandwidth case (still very near the goal value, under the circumstances) and data use ratio respectively 87.8% vs. 91.1% for the fixed high bandwidth case.

The middleware is augmented with a User Preferences Interface (UPI) application operating as a user preferences multi-sensor, that enables the user to customize or fine-tune the adaptation logic on cross-cutting concerns for all applications subscribed to the middleware through an application policy. The UPI additionally enables the user to select one among alternative user preference modes, imposing alternative real-life adaptation behaviors, like” “LowCost” or “SaveBattery” or “HighQuality” or “Normal”.

To allow developers to customize the adaptation behaviour of the middleware as appropriate for each application the Volare policy language was created, a simple, declarative **Adaptation Policy Specification Language (APSL)** for policy-based DCAA on mobile systems, utilizing a multiple weightd adaptation rules per QoS variable as well as a novel Conflict Resolution procedure.

The Volare middleware and policy language use a combination of a two level policy architecture, as described §7.4-7.5. It is composed of the platform global policy and the active application policy concerning the SR adaptation. Through the application policy, the width and breadth of adaptation can be allowed to depend also on the platform and the device resources, in cases where the application developer cannot predict the platform his application will be run on, which is fairly common in mobile applications development. This allows the mobile device’s firmware developer (via the global policy) to also influence the adaptation process and to account for each mobile device’s unique resource characteristics, through adaptation-rules on the same adaptation-variable. This approach makes the application policy developer’s task easier and lower the development cost significantly compared to a more typical situation-action architecture, by enabling separate adaptation policies to be developed to specifically address the needs of individual mobile devices and applications. The developer of the application adaptation policy does not need to develop separate adaptation policies for each possible mobile device his application may be deployed on.

To further assist developers in creating the relevant adaptation policies, the **Volare Weight Based Adaptation Reasoning Methodology** was also developed, a process for specifying independent, competing, adaptation-strategies that control the adaptation process, offering better understanding and leveraging access of the adaptation-concerns to the user. This includes a set of generic Models, Procedures and methodology Tools that have been designed, to facilitate in the task of adaptation policy editing. Further methodological tools, procedures and generic models, as well as testing & verification tools, were developed, such as the PEVApp, which including automated test suites generation and offline repeated policy execution and charts presentation on test suite results.

The weight-based adaptation reasoning technique characteristics like the identification of adaptation-concerns and building the adaptation policy logic in the corresponding adaptation-strategies, presents advantages especially in cases of high behavioral variability(see §6.1-6.2) . Building a policy in the weight-based approach for a scenario that lies within its scope, delivers certain advantages in comparison to the conventional action-based approach.

Determination of the major adaptation-concerns around which the policy will be built, and building it as a set of independent adaptation-strategies, each of which is the collection of adaptation-rules that serve a specific adaptation-concern over the valid context domain. The weight-based adaptation reasoning technique may lead, when there is high behavioral variability, to policy authoring significantly shorter in the number of rules and simpler in testing and verification than an equivalent “action-based” policy (see ch. 6 and §8.6 for detailed analysis). Thus, as seen in §8.6, the fact that the structure of the policy in multiple eventually simultaneously selectable at policy execution adaptation-rules per variable, allows for shorter implementations of high behavioral variability adaptation behaviors concerning multiple adaptation-concerns, compared to the conventional single rule approaches. This is because the Volare policy language enables the developer to develop separate sub-policies specific to each group of contextual data that can affect the adaptation process, and then bind them all together, instead of having to develop a full separate policy for each possible contextual situation.

A further advantage of the WB adaptation reasoning is the fact that it provides a great testability potential for each adaptation-strategy, either simply on top of the Generic Operational Constraints adaptation-strategy or in parallel to the other ones. The weight-based technique provides the opportunity through the UPI to selectively deemphasize specific sets of adaptation-rules over other, according to which adaptation strategy they are associated to, without modifying the policy file, facilitating testing and adaptation behavior verification.

The weight-based adaptation reasoning technique and the developed methodology also supports the introduction in the policy of unanticipated at middleware design time quantitative Long Term Performance Goals (LTPGs) with finite horizons spanning over many binding sessions, as discussed in subchapter 6.4, without any hard coded middleware provision, thus allowing for long term resource management. This improves long term resource efficiently as seen in Table 8-5, while it also eliminates the need for user supervision to manage limited resources or monetary budget. For instance the monthly average of min battery level value before recharge time was 18,4% vs. a LTPG target of min 20% (for eventual emergencies), in comparison to an achieved average value of 27.2% for the full adaptation policy run under the same monthly context data matrix.

Finally, the User Preferences Model, supported by the UPI application to communicate dynamically user preferences changes to the middleware integrates user preferences in the policy structure. Thus, user may easily customize the adaptation behavior so that the application/middleware behaves differently on developer set user-customizable parameters of the adaptation logic, without intervening in the policy. As evidenced by the evaluation in §8.4, different QoS Request values are derived at each alternative user preference mode. Note that the improvement realized by user preference mode selection, not only in application performance but also on the savings on the cost of binding to the cloud service under variable context (mainly bandwidth, battery and user preference), may be significant for the user when viewed on a monthly or yearly basis. It certainly will incite the user to implement case-specific expenses control by selecting the appropriate user preference mode at each session, instead of the default"Normal" Mode.   .

Naturally, due to the domain nature, the efficiency results might be widely different depending on the case study, adaptation policies, context data sequence and the service provision options available to the CSP. Additionally, if the economies of scale for a Service Provider on the cloud to its registered users are considered, by providing dynamic adaptation to the mobile applications through a DCAA middleware, it sums up to considerable savings. Of course, on an individual basis, most depends on the network bandwidth variability for each specific user.

9.2 Scope and Limitations

Although the current scenario involves service discovery on the cloud, the Volare approach may support other mobile middleware-based adaptation scenarios as well. The basic constraints of the Volare approach, as defined in the current project, are the following:

1. It is required that the adaptation actions are expressed as values of the adaptation-variables that are distinguished in the pre-defined middleware configuration variables such as operation parameters and modalities of the middleware components, or in the active Service Request QoS variables. By consequence, in addition to the Service Request adaptation, the allowed adaptation actions are limited to the middleware configuration variables values set at middleware design time.
2. It can only support scenarios with adaptation models that correspond to the declared adaptation-variables. Consequently, unanticipated adaptation behaviours that may not be covered by the adaptation-variables already declared and incorporated in the middleware modules design cannot be supported.
3. For Volare APSL-compatible adaptation policies simplicity and clarity, no functions are allowed in the adaptation-rules (in the predicates or the adaptation statements), only algebraic expressions with the operators {+,-,*,/ }, or ^ (power) or abs (absolute value).
4. Volare supports high behavioural variability in the policy and unanticipated introduction of statistic-calculation-variables on the supported by the Statistical Analysis Module. However, the statistical functions for statistic-

calculation-variables and the “periods” are predetermined at middleware design time. Consequently, only LTPGs on supported “horizons” and metrics on supported statistic functions and “periods” can be defined.

5. Currently, the middleware and language only support a single service binding active at a time. Further conflict resolution directives would have to be developed to support multiple simultaneous service bindings, as they need to prioritize using a common pool of resources.

9.3 Vision and Further Perspectives

The research work presented in this Thesis can be further expanded in various directions. In this section, we quickly discuss the ones that in our opinion are the most promising or the more useful.

1. Further Simulation-based Policy Optimization on LTPGs Parameters

On optimization problems where the objective function is not explicitly known (of closed form), the typical optimization methods cannot be applied [85][86][87] [33]. optimization is even more difficult when parameters of stochastic nature intervene, like mobile bandwidth variation. In our case of mobile dynamic context-aware adaptation with Long Term Performance Goals, considering the adaptation process where a Composite Policy through a Policy Engine provides an adaptation values output vector in response to a context vector, assessment of the policy performance on a LTPG can only be evaluated over a sequence of adaptations covering the whole temporal horizon of the LTPG. Additionally, due to the random character of certain parameters like: bandwidth variation, session duration, user preference etc., this sequence of binding cycles is a stochastic sequential decision-making process [87].

Suppose we would like to compare the performance of a Long Term Performance Goal over a specified horizon on two different values of a policy parameter, typically a parameter in the adaptation-rules concerning this LTPG. At first we need a performance assessment criterion for the LTPG. Due to the stochastic nature of the process, we need to collect or design appropriate context data samples containing single or multiple horizons of the LTPG and run repeated simulated execution on each of these samples representing real or virtual sequences of CSD sessions, get the average performance assessment for each parameter value and then compare the two results.

Of course even for one continuous or even discrete parameter, many runs of the above simulation-based optimization procedure will have to be executed, evidently making the computation very intensive for a mobile device of limited resources. In the case of more than one parameter (multi-criteria optimization), or more than one LTPGs, this is even more difficult.

However, the simulated repeated execution could be implemented on a remote component on the web or the cloud, with access to the device context & adaptation history and simulation tools (like a simulated Policy Engine) and application of MLTs. This is a challenge for future work: automated simulation-based policy optimization on LTPG parameter(s) based on the recorded Usage Model.

2. Enlarging the DCAA support to independent CAA Applications

On the middleware side, instead of just adapting the application service request and the middleware components, the middleware may also support Context Aware Adaptive Applications (CAAAs) that may have a Volare compatible policy and use the Volare Policy Engine for adaptation calculation.

This extension would require a few simple functionalities, such as: dynamic dispatching of the application-specific adaptation results, or enriching the middleware Event Service with application-specific events triggering adaptation.

3. Establishing Coordinated Adaptation on concurrent Applications

Coordinated DCAA middleware operation with multiple applications supported by the middleware in parallel [64][42]. In this case, the middleware global policy should also provide rules on coordinated adaptation and the middleware mechanisms for shared resources and bandwidth shaping.

4. Dealing with Uncertainty

On scenarios “dealing with uncertainty” [56][76][34], we consider that our Weight-based approach is in fact provided with a conceptual “relative importance calculus” that may well be applied on reasoning for both the context information trustworthiness provided – thus influencing the gravity of a context change to the corresponding adaptation actions(s) – as well as on multiple adaptation actions. Our Weight-based approach provides the reasoning tools for evaluating adaptation on uncertain context information, resolving to the most fitting adaptation under uncertainty conditions and may be made productive in this perspective.

5. Implementation of the Volare Approach to other Domains

Use of the Volare Weight-based Approach in related research areas like Security, Dynamic Service Composition, Resource Discovery on the Cloud, Fault handling as well as in other fields of mobile CAA would be of great interest.

REFERENCES

[1] Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., ... & Stoica, I. (2009). Above the clouds: A Berkeley view of cloud computing. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, 28, 13.

[2] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., & Zagorodnov, D. (2009, May). The eucalyptus open-source cloud-computing system. In Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on (pp. 124-131). IEEE.

[3] Weiss, A. (2007). Computing in the clouds. networker, 11(4).

[4] Sotomayor, B., Montero, R. S., Llorente, I. M., & Foster, I. (2008, October). Capacity leasing in cloud systems using the opennebula engine. In Workshop on Cloud Computing and its Applications (Vol. 3).

[5] Buyya, R., Yeo, C. S., & Venugopal, S. (2008, September). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on (pp. 5-13). Ieee.

[6] Capra, L., Emmerich, W., & Mascolo, C. (2003). Carisma: Context-aware reflective middleware system for mobile applications. Software Engineering, IEEE Transactions on, 29(10), 929-945.

[7] Keeney, J., & Cahill, V. (2003, June). Chisel: A policy-driven, context-aware, dynamic adaptation framework. In Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on (pp. 3-14). IEEE.

[8] Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J., & Retschitzegger, W. (2003, January). Context-awareness on mobile devices-the hydrogen approach. In System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on (pp. 10-pp). IEEE.

[9] Mukhija, A., Dingwall-Smith, A., & Rosenblum, D. S. (2007, November). Qos-aware service composition in dino. In Web Services, 2007. ECOWS'07. Fifth European Conference on (pp. 3-12). IEEE.

[10] Oreizy, P., Heimbigner, D., Johnson, G., Gorlick, M. M., Taylor, R. N., Wolf, A. L., Rosenblum D. S. & Quilici, A. (1999). An architecture-based approach to self-adaptive software. IEEE Intelligent systems, 14(3), 54-62.

[11] Scowen, R. S. (1998). Extended BNF-a generic base standard. Technical report, ISO/IEC 14977. http://www. cl. cam. ac. uk/mgk25/iso-14977. pdf.

[12] Damianou, N., Dulay, N., Lupu, E., & Sloman, M. (2001). The ponder policy specification language. In Policies for Distributed Systems and Networks (pp. 18-38). Springer Berlin Heidelberg.

[13] Baldauf, M., Dustdar, S., & Rosenberg, F. (2007). A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing, 2(4), 263-277.

[14] Frederick, R., Jacobson, V., & Design, P. (2003). RTP: A transport protocol for real-time applications. IETF RFC3550.

[15] Capra, L., Zachariadis, S., & Mascolo, C. (2005, July). Q-CAD: QoS and context aware discovery framework for mobile systems. In Pervasive Services, 2005. ICPS'05. Proceedings. International Conference on (pp. 453-456) IEEE

[16] Papakos, P., Rosenblum, D. S., Mukhija, A., & Capra, L. (2009). Volare: adaptive web service discovery middleware for mobile systems. Electronic Communications of the EASST, 19.

[17] Papakos, P., Capra, L., & Rosenblum, D. S. (2010, November). Volare: context-aware adaptive cloud service discovery for mobile systems. In Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware (pp. 32-38). ACM.

[18] Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjorven, E., ... & Stav, E. (2009). A comprehensive solution for application-level adaptation. Software: Practice and Experience, 39(4), 385-422.

[19] Amundsen, S. L., & Eliassen, F. (2008). A resource and context model for mobile middleware. Personal and Ubiquitous Computing, 12(2), 143-153.

[20] Chan, A. T., & Chuang, S. N. (2003). MobiPADS: a reflective middleware for context-aware mobile computing. Software Engineering, IEEE Transactions on, 29(12), 1072-1085.

[21] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems, 25(6), 599-616.

[22] Amundsen, S., Lund, K., Eliassen, F., & Staehli, R. (2004, November). QuA: platform-managed QoS for component architectures. In Proceedings from Norwegian Informatics Conference (NIK) (pp. 55-66).

[23] Grace, P., Blair, G. S., & Samuel, S. (2003). ReMMoC: A reflective middleware to support mobile client interoperability. In On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE (pp. 1170-1187). Springer Berlin Heidelberg.

[24] David, P. C., & Ledoux, T. (2002). An infrastructure for adaptable middleware. In On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE (pp. 773-790). Springer Berlin Heidelberg.

[25] Bertolino, A., Emmerich, W., Inverardi, P., Issarny, V., Liotopoulos, F., & Plaza, P. (2008, September). PLASTIC: Providing lightweight & adaptable service technology for pervasive information & communication. In Automated Software Engineering-Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on (pp. 65-70). IEEE.

[26] Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., ... & Scholz, U. (2009). Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In Software engineering for self-adaptive systems (pp. 164-182). Springer Berlin Heidelberg.

[27] Chun, B. G., & Maniatis, P. (2009, May). Augmented Smartphone Applications Through Clone Cloud Execution. In HotOS (Vol. 9, pp. 8-11).

[28] Rukzio, E., Siorpaes, S., Falke, O., & Hussmann, H. (2005, July). Policy based adaptive services for mobile commerce. In Mobile Commerce and Services, 2005. WMCS'05. The Second IEEE International Workshop on (pp. 183-192). IEEE.

[29] Capra, L., Emmerich, W., and Mascolo, C., 2001. Reflective middleware solutions for context- aware applications. In: REFLECTION 2001: Proceedings of the 3rd international conference on metalevel architectures and separation of crosscutting concerns. London, UK: Springer- Verlag, 126–133.

[30] McKinley, P. K., Sadjadi, S. M., Kasten, E. P., & Cheng, B. H. (2004). Composing Adaptive Software. COMPUTER, 56-64.

[31] Frasincar, F., & Houben, G. J. (2002, January). Hypermedia presentation adaptation on the semantic web. In Adaptive Hypermedia and Adaptive Web-Based Systems (pp. 133-142). Springer Berlin Heidelberg.

[32] Gu, T., Pung, H. K., & Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. Journal of Network and computer applications, 28(1), 1-18.

[33] Harman, M., Burke, E., Clark, J., & Yao, X. (2012, September). Dynamic adaptive search based software engineering. In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement (pp. 1-8). ACM.

[34] Kakousis, K., Paspallis, N., & Papadopoulos, G. A. (2010). A survey of software adaptation in mobile and ubiquitous computing. Enterprise Information Systems, 4(4), 355-389.

[35] Friedman-Hill, E. J. (1997). Jess, the java expert system shell. Distributed Computing Systems, Sandia National Laboratories, USA.

[36] Malandrino, D., Mazzoni, F., Riboni, D., Bettini, C., Colajanni, M., & Scarano, V. (2010). MIMOSA: context-aware adaptation for ubiquitous web access. Personal and ubiquitous computing, 14(4), 301-320.

[37] Damianou, N., Dulay, N., Lupu, E., & Sloman, M. (2001). The ponder policy specification language. In Policies for Distributed Systems and Networks (pp. 18-38). Springer Berlin Heidelberg.

[38] Kagal, L., Finin, T., & Joshi, A. (2003, June). A policy language for a pervasive computing environment. In Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on (pp. 63-74). IEEE.

[39] Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., ... & Lott, J. (2003, June). KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on (pp. 93-96). IEEE.

[40] Noble, B. D., & Satyanarayanan, M. (1999). Experience with adaptive mobile applications in Odyssey. Mobile Networks and Applications, 4(4), 245-254.

[41] Amundsen, S. L., Lund, K., Griwodz, C., & Halvorsen, P. (2005, August). QoS-aware mobile middleware for video streaming. In Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on (pp. 54-61). IEEE.

[42] Poladian, V., Sousa, J. P., Garlan, D., & Shaw, M. (2004, May). Dynamic configuration of resource-aware services. In Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on (pp. 604-613). IEEE.

[43] Preuveneers, D., & Berbers, Y. (2005, January). Adaptive context management using a component-based approach. In Distributed Applications and Interoperable Systems (pp. 14-26). Springer Berlin Heidelberg.

[44] Bandara, A. K., Lupu, E. C., & Russo, A. (2003, June). Using event calculus to formalise policy specification and analysis. In Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on (pp. 26-39). IEEE.

[45] David, P. C., & Ledoux, T. (2005, November). WildCAT: a generic framework for context-aware applications. In Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (pp. 1-7). ACM.

[46] Mascolo, C., Capra, L., & Emmerich, W. (2002). Mobile computing middleware. In Advanced lectures on networking (pp. 20-58). Springer Berlin Heidelberg.

[47] CLOUDSWITCH: http://www.cloudswitch.com/section/products Last visited 10/8/2013

[48] Petcu, D. (2013). A panorama of cloud services. Scalable Computing: Practice and Experience, 13(4).

[49] AMAZON – EC2 – Pricing, http://aws.amazon.com/ec2/#pricing Last visited 14/8/2013

[50] AMAZON – S3 – Pricing, http://aws.amazon.com/ec2/#pricing Last visited 14/8/2013

[51] SALESFORCE Enterprise Edition CRM – Pricing, http://www.salesforce.com/crm/editions/pricing.jsp?d=70130000000FLyo&internal=true Last visited 14/8/2010

[52] AMAZON CloudFront, http://aws.amazon.com/cloudfront/ Last visited 14/08/2013

[53] Ali, R., Dalpiaz, F., & Giorgini, P. (2010). A goal-based framework for contextual requirements modeling and analysis. Requirements Engineering, 15(4), 439-458.

[54] Issarny, V., Caporuscio, M., & Georgantas, N. (2007, May). A perspective on the future of middleware-based software engineering. In 2007 Future of Software Engineering (pp. 244-258). IEEE Computer Society.

[55] Sama, M., Rosenblum, D. S., Wang, Z., & Elbaum, S. (2008, November). Model-based fault detection in context-aware adaptive applications. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (pp. 261-271). ACM.

[56] Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H., & Malm, E. J. (2003). Managing context information in mobile devices. Pervasive Computing, IEEE, 2(3), 42-51.

[57] Dey, A.K., 2001. Understanding and using context. Personal Ubiquitous Computing, 5 (1), 4–7.

[58] Ouyang, J. Q., Shi, D. X., Ding, B., Feng, J., & Wang, H. M. (2009). Policy Based Self-Adaptive Scheme in Pervasive Computing. Wireless Sensor Network, 1(1).

[59] Garlan, D., Cheng, S. W., Huang, A. C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer, 37(10), 46-54.

[60] Raento, M., Oulasvirta, A., Petit, R., & Toivonen, H. (2005). ContextPhone: A prototyping platform for context-aware mobile applications. Pervasive Computing, IEEE, 4(2), 51-59.

[61] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., & Riboni, D. (2010). A survey of context modelling and reasoning techniques. Pervasive and Mobile Computing, 6(2), 161-180.

[62] Heineman, G. T., Loyall, J., & Schantz, R. (2004). Component technology and qos management. In Component-Based Software Engineering (pp. 249-263). Springer Berlin Heidelberg.

[63] Fahy, P., & Clarke, S. (2004). CASS–a middleware for mobile context-aware applications. In Workshop on Context Awareness, MobiSys.

[64] Efstratiou, C., Friday, A., Davies, N., & Cheverst, K. (2002). A platform supporting coordinated adaptation in mobile systems. In Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on (pp. 128-137). IEEE.

[65] La, H. J., & Kim, S. D. (2010, July). A conceptual framework for provisioning context-aware mobile cloud services. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on (pp. 466-473). IEEE.

[66] Xiao, Y., Hui, P., Savolainen, P., & Ylä-Jääski, A. (2011, June). CasCap: cloud-assisted context-aware power management for mobile devices. In Proceedings of the second international workshop on Mobile cloud computing and services (pp. 13-18). ACM.

[67] Jang, C., Chang, H., Ahn, H., Kang, Y., & Choi, E. (2011). Profile for effective service management on mobile cloud computing. In Advanced Communication and Networking (pp. 139-145). Springer Berlin Heidelberg.

[68] Hussein, M., Han, J., & Colman, A. (2010). Specifying and Verifying the Context-aware Adaptive Behavior of Software Systems (p. 18). Technical Report# C3-516_03, Swinburne University of Technology.

[69] Bettini, C., Pareschi, L., & Riboni, D. (2008). Efficient profile aggregation and policy evaluation in a middleware for adaptive mobile applications. Pervasive and Mobile Computing, 4(5), 697-718.

[70] Chen, T. Y., Tse, T. H., & Zhou, Z. (2002, July). Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing. In ACM SIGSOFT Software Engineering Notes (Vol. 27, No. 4, pp. 191-195). ACM.

[71] Wang, Z., Elbaum, S., & Rosenblum, D. S. (2007, May). Automated generation of context-aware tests. In Software Engineering, 2007. ICSE 2007. 29th International Conference on (pp. 406-415). IEEE.

[72] White, L. J., & Cohen, E. I. (1980). A domain strategy for computer program testing. Software Engineering, IEEE Transactions on, (3), 247-257.

[73] Rutherford, M. J., Carzaniga, A., & Wolf, A. L. (2006, November). Simulation-based test adequacy criteria for distributed systems. In Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering (pp. 231-241). ACM.

[74] Zhang, J., & Cheng, B. H. (2006, May). Model-based development of dynamically adaptive software. In Proceedings of the 28th international conference on Software engineering (pp. 371-380). ACM.

[75] Liaskos, S., McIlraith, S. A., Sohrabi, S., & Mylopoulos, J. (2010, September). Integrating preferences into goal models for requirements engineering. In Requirements Engineering Conference (RE), 2010 18th IEEE International (pp. 135-144). IEEE.

[76] Bishop, C. M. (2006). Pattern recognition and machine learning (Vol. 1, p. 740). New York: springer.

[77] Letier, E. (2001). Reasoning about agents in goal-oriented requirements engineering (Doctoral dissertation, PhD thesis, Université catholique de Louvain).

[78] Papadogkonas, D., Roussos, G., & Levene, M. (2008, July). Analysis, ranking and prediction in pervasive computing trails. In Intelligent Environments, 2008 IET 4th International Conference on (pp. 1-8). IET.

[79] Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 4(2), 14.

[80] Hong, J. Y., Suh, E. H., & Kim, S. J. (2009). Context-aware systems: A literature review and classification. Expert Systems with Applications, 36(4), 8509-8522.

[81] Kapitsaki, G. M., Prezerakos, G. N., Tselikas, N. D., & Venieris, I. S. (2009). Context-aware service engineering: A survey. Journal of Systems and Software, 82(8), 1285-1297.

[82] Byun, H. E., & Cheverst, K. (2001, July). Exploiting user models and context-awareness to support personal daily activities. In Workshop in UM2001 on User Modeling for Context-Aware Applications.

[83] Henricksen, K., & Indulska, J. (2004, March). A software engineering framework for context-aware pervasive computing. In Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on (pp. 77-86). IEEE.

[84] Paspallis, N., Kakousis, K., & Papadopoulos, G. A. (2008, July). A multi-dimensional model enabling autonomic reasoning for context-aware pervasive applications. In Proceedings of the 5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services (p. 56). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[85] Azadivar, F. (1999, December). Simulation optimization methodologies. In Proceedings of the 31st conference on Winter simulation: Simulation---a bridge to the future-Volume 1 (pp. 93-100). ACM.

[86] Gosavi, A. (2003). Simulation-based optimization: parametric optimization techniques and reinforcement learning (Vol. 25). Springer.

[87] Andradóttir, S. (1998, December). A review of simulation optimization techniques. In Proceedings of the 30th conference on Winter simulation (pp. 151-158). IEEE Computer Society Press.

[88] Cheung, T. L., Okamoto, K., Maker III, F., Liu, X., & Akella, V. (2009, October). Markov decision process (MDP) framework for optimizing software on mobile phones. In Proceedings of the seventh ACM international conference on Embedded software (pp. 11-20). ACM.

[89] Suneja, S., Navda, V., Ramjee, R., & Lara, E. D. (2013, June). EnVi: energy efficient video player for mobiles. In Proceeding of the 2013 workshop on Cellular networks: operations, challenges, and future design (pp. 25-30). ACM.

[90] Shu, P., Liu, F., Jin, H., Chen, M., Wen, F., Qu, Y., & Li, B. (2013, April). eTime: energy-efficient transmission between cloud and mobile devices. In INFOCOM, 2013 Proceedings IEEE (pp. 195-199). IEEE.

[91] Donohoo, B. K., Ohlsen, C., & Pasricha, S. (2011, October). AURA: An application and user interaction aware middleware framework for energy optimization in mobile devices. In Computer Design (ICCD), 2011 IEEE 29th International Conference on (pp. 168-174). IEEE.

[92] Carroll, A., & Heiser, G. (2010, June). An analysis of power consumption in a smartphone. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (pp. 21-21).

[93] Mohapatra, S., Dutt, N., Nicolau, A., & Venkatasubramanian, N. (2007). DYNAMO: A cross-layer framework for end-to-end QoS and energy optimization in mobile handheld devices. IEEE Journal on selected areas in communications, 25(4), 722-737.

[94] Balasubramanian, N., Balasubramanian, A., & Venkataramani, A. (2009, November). Energy consumption in mobile phones: a measurement study and implications for network applications. In Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference (pp. 280-293). ACM.

[95] Ra, M. R., Paek, J., Sharma, A. B., Govindan, R., Krieger, M. H., & Neely, M. J. (2010, June). Energy-delay tradeoffs in smartphone applications. In Proceedings of the 8th international conference on Mobile systems, applications, and services (pp. 255-270). ACM.

[96] Cuervo, E., Balasubramanian, A., Cho, D. K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010, June). MAUI: making smartphones last longer

with code offload. In Proceedings of the 8th international conference on Mobile systems, applications, and services (pp. 49-62). ACM.

[97] Dinh, H. T., Lee, C., Niyato, D., & Wang, P. (2011). A survey of mobile cloud computing: architecture, applications, and approaches. Wireless communications and mobile computing.

[98] Bolchini, C., Curino, C. A., Quintarelli, E., Schreiber, F. A., & Tanca, L. (2007). A data-oriented survey of context models. ACM Sigmod Record, 36(4), 19-26.

[99] Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on (pp. 249-262). IEEE.

[100] Brown, G., Cheng, B. H., Goldsby, H., & Zhang, J. (2006, May). Goal-oriented specification of adaptation requirements engineering in adaptive systems. In Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems (pp. 23-29). ACM.

[101] Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., & Jézéquel, J. M. (2009). Modeling and validating dynamic adaptation. In Models in Software Engineering (pp. 97-108). Springer Berlin Heidelberg.

[102] Aksit, M., & Choukair, Z. (2003, May). Dynamic, adaptive and reconfigurable systems overview and prospective vision. In Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on (pp. 84-89). IEEE.

[103] Esfahani N, Kouroshfar E and Malek S. Taming Uncertainty in Self-Adaptive Software. ESEC/FSE’11, September 5 – 9, 2011, Szeged,Hungary. ACM 978-1-4503-0443-6/11/09. Lund, Sweden, September 19th-20th, 2012.

[104] Mell, P., & Grance, T. (2009). The NIST definition of cloud computing. National Institute of Standards and Technology, 53(6), 50.

[105] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., & Leaf, D. (2011). NIST cloud computing reference architecture. NIST Special Publication, 500, 292.

[106] Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. Journal of internet services and applications, 1(1), 7-18.

[107] Sriram, I., & Khajeh-Hosseini, A. (2010). Research agenda in cloud technologies. arXiv preprint arXiv:1001.3259.

[108] Wu, L., & Buyya, R. (2010). Service level agreement (SLA) in utility computing systems. Performance and dependability in service computing: Concepts, techniques and research directions, 1, 1-25.

[109] Yau, S. S., & An, H. G. (2011). Software engineering meets services and cloud computing. IEEE Computer, 44(10), 47-53.

[110] Niu, D., Feng, C., & Li, B. (2012, March). A theory of cloud bandwidth pricing for video-on-demand providers. In INFOCOM, 2012 Proceedings IEEE (pp. 711-719). IEEE.

[111] Tsai, W. T., Sun, X., & Balasooriya, J. (2010, April). Service-oriented cloud computing architecture. In Information Technology: New Generations (ITNG), 2010 Seventh International Conference on (pp. 684-689). IEEE.

[112] Goscinski, A., & Brock, M. (2010). Toward dynamic and attribute based publication, discovery and selection for cloud computing. Future Generation Computer Systems, 26(7), 947-970.

[113] Buyya, R., Ranjan, R., & Calheiros, R. N. (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In Algorithms and architectures for parallel processing (pp. 13-31). Springer Berlin Heidelberg.

[114] Wu, Y., Wu, C., Li, B., Qiu, X., & Lau, F. C. (2011, June). Cloudmedia: When cloud on demand meets video on demand. In Distributed Computing Systems (ICDCS), 2011 31st International Conf. on (pp. 268-277). IEEE.

[115] Venticinque, S., Aversa, R., Di Martino, B., Rak, M., & Petcu, D. (2011, January). A cloud agency for SLA negotiation and management. In Euro-Par 2010 Parallel Processing Workshops (pp. 587-594). Springer Berlin Heidelberg.

[116] Elgazzar, K., Hassanein, H. S., & Martin, P. (2013). DaaS: Cloud-based mobile Web service discovery. Pervasive and Mobile Computing.

[117] Joshi, K. P., Yesha, Y., Finin, T., & Joshi, A. (2012). Policy based Cloud Services on a VCL platform. In Proceedings of the first International IBM Cloud Academy Conference (ICA CON 2012).

[118] Joshi, K. P., Finin, T., Yesha, Y., Joshi, A., Golpayegani, N., & Adam, N. (2012, July). A Policy-based Approach to Smart Cloud Services. In Proceedings of the Annual Service Research and Innovation Institute Global Conf.

[119] Elgazzar, K., Ejaz, A., & Hassanein, H. S. (2013). AppaaS: offering mobile applications as a cloud service. Journal of Internet Services and Applications, 4(1), 1-12.

[120] Ortiz, J., de Almeida, V. T., & Balazinska, M. (2013, June). A vision for personalized service level agreements in the cloud. In Proceedings of the Second Workshop on Data Analytics in the Cloud (pp. 21-25). ACM.

[121] Badidi, E. (2013). A FRAMEWORK FOR SOFTWARE-AS-A-SERVICE SELECTION AND PROVISIONING. International Journal of Computer Networks & Communications, 5(3).

[122] Scandurra, P., Raibulet, C., Potena, P., Mirandola, R., & Capilla, R. (2012). Adapting Cloud-based Applications through a Coordinated and Optimized Resource Allocation Approach. In CLOSER (pp. 355-364).

[123] Wu, Y., Wu, C., Li, B., Qiu, X., & Lau, F. C. (2011, June). Cloudmedia: When cloud on demand meets video on demand. In Distributed Computing Systems (ICDCS), 2011 31st International Conf. on (pp. 268-277). IEEE.

[124] Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I. M., ... & Galan, F. (2009). The reservoir model and architecture for open federated cloud computing. IBM Journal of Research and Development, 53(4), 4-1.

[125] Garg, S. K., Versteeg, S., & Buyya, R. (2011, December). Smicloud: A framework for comparing and ranking cloud services. In Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on (pp. 210-218). IEEE.

[126] Wang, S., Zheng, Z., Sun, Q., Zou, H., & Yang, F. (2011, April). Cloud model for service selection. In Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on (pp. 666-671). IEEE.

[127] Alrifai, M., & Risse, T. (2009, April). Combining global optimization with local selection for efficient QoS-aware service composition. In Proceedings of the 18th international conference on World wide web (pp. 881-890). ACM.

[128] Rao, A., Legout, A., Lim, Y. S., Towsley, D., Barakat, C., & Dabbous, W. (2011, December). Network characteristics of video streaming traffic. In Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies (p. 25). ACM.

APPENDIX A: The Global PolicyExample

```
// Composite Policy: Global & VSTREAM Application Policy
//  Global Policysetting 3 Policy Consecutive Execution Cycles (CECs) at each policy execution
// for adaptation
// 1st Cycle: 2 Subpolicies for decision-making on the use of Date-based vs. Usage-based
// Criteria
// 2nd Cycle: DEFAULT + HIGHQUALITY + NORMAL + LOWCOST + SAVEBATTERY +
// VLOWBATTERY
// + LOWBUDGET + 2 HIGH - LOW DISRUPTION = 9 Subpolicies
// 3rd Cycle - 2 Subpolicies: Implement Calculated Adaptation or Not
// Application Policy on Video-streaming from services on the Cloud  - 1st & 2nd policy
// execution Cycles  Subpolicies
// 1st Cycle: 2 Subpolicies for decision-making on the use of Date-based vs. Usage-based
// Criteria
// 2nd Cycle: DEFAULT + HIGHQUALITY + SAVEBATTERY + LOWCOST + NORMAl +
// VLOWBATTERY + HIGHBATRES + LOWBATRES
// + HIGHCOSTBUDGET + LOWCOSTBUDGET + HIGHDATABUDGET + LOWDATABUDGET =  12
// Subpolicies.


// BRIEF EXPLANATIONS ON THE STATISTICAL ANALYSIS MODULE (SAM)
// The Statistical Analysis Module (SAM) provides to the Global and Application Policy
// developers a Volare middleware tool for controlled access to cumulative or statistic context
// & adaptation history of values of pre-declared variables, by declaration in the Policy of
// statistic-calculation-variables, based on the following pre-determined characteristics:

// parameterID: representing declared variables in the Global or Application Policy, thus
// taking values at policy execution;
// periodID: representing pre-determined repeated time periods, like: Bindingcycle, Session,
//Daily, Dischargecycle, Monthly;
// statisticTermID : representing pre-determined statistic terms, like: Sum, Max, Min, Avg,
//Stdev, UpperConfLim.
// SAM supports the declaration of statistic-calculation-variables, in the following declaration
//notation:
// typeID  ContextVar varName == StatisticalAnalysisM.parameterID.periodID.statisticTermID;


    // 1. Variables custom-naming convention
// If applicable, atomic context-variables or calculation-variables are custom-named by the
// following suggested naming-convention: varName => parameterIDperiodID
// (e.g. cloudCostSession, durationRecheckcycle).
// If applicable, statistic context-variables or composite calculation-variables are custom-
// named by the  following suggested naming-convention: varName =>
// parameterIDperiodIDstatisticTermID (e.g. batterySessionMax).


    // 2. The atomic context-variables
// The atomic context-variables that form the basis for the cumulative or statistic-calculation-
// variables are either context  parameters like: bandwidth, battery, userPref etc., or
// cumulative quantity parameters over the relevant most generic period, typically the
// “Recheckcycle". These atomic cumulative context-variables are:
```

```
// cloudMBsRecheckcycle = The MBs of data downloaded during the current Recheckcycle;

// cloudDurationRecheckcycle = The duration in seconds of the current Recheckcycle till now;
// cloudCostRecheckcycle = The cost incurred by the binding on the Cloud service during the
// current Recheckcycle.


    // 3. Other computing-environment variables data
// In addition to the above atomic context-variables, all other variables declared in the Global
// or Application Policy (calculation, configuration or adaptation (QoS) variables), get a
// calculated and recorded value at policy execution.
// The APSL provides the formulation and the middleware through SAM supports the
// declaration of new statistic- context-variables for the deduction of useful information from
// the declared variables recorded data.



    // 4. The cumulative or statistically-inferred "statistic-calculation-variables"
// Based on the above atomic context- or calculation-variables, cumulative parameters over
// higher periods are declared, like:
// cloudCostMonthly, cloudDurationSession, cloudMBsDaily etc, as well as statistic parameters
// on them.
// We distinguish "cloudDuration" which is the duration on Cloud Service Discovery (CSD) and
//"overallDuration".
// "overallDuration" within a periodID denotes the time interval from the beginning of the
// current period till now.
// Only the "overallDurationDischargecycle" is declared as statistic context-variable, while
// "overallDurationDaily" or "overallDurationMonthly" are declared as calculation-variables
// using the timeNow  context-variable, while for periods: "Recheckcycle", "Bindingcycle" and
// "Session", cloudDuration = overallDuration (there is no idle time interval in them).


    // 5. The "periods" currently supported by the Statistical Analysis Module:
// Recheckcycle = the period between two consecutive monitorings at launched Cloud Service
// Discovery Request
// Bindingcycle = the period of each discovery at binding on the cloud at a Service Level
// Agreement  QoS values
// Session = period starting when an App launches SR on the Cloud until the end of the
//application SR.
// Daily = period starting from 00:00:01 till now (until the end of the current day).

// Dischargecycle = period from the last battery charging till now (untill the next recharging).

// Monthly = the period starting the fist of the current month till now (until end of the month).

// OverallHistory = the period of the recorded context & adaptation history of the device.


    // 6. The "statisticTerms"  currently supported by the Statistical Analysis Module (SAM):
// Sum = Returns the sum of the values of the numeric parameterID within the specified
// periodID.
// Max = Returns the maximum of the numeric parameterID values within the specified
// periodID.
// Min = Returns the minimum of the numeric parameterID values within the specified
// periodID.

// Avg = Returns the average of the numeric parameterID values within the specified periodID.
// Stdev = Returns the Standard Deviation of the numeric parameterID values within the
// specified periodID.
// UpperConfLim = Returns the Upper Confidence Limit on the parameterID values over the
//specified periodID.



    // 7. The policy-based Long Term Performance Goals (LTPGs)
// Four policy-based LTPGs, each over a finite horizon, are served through the Global or the
// Application Policy A/Rs. Their goals
// are fulfilled over many sessions and require monitoring over cumulative context &
// adaptation history parameters. For each LTPG it is specified: (a) serving Adaptation-strategy
// rules, (b) time horizon, (c) monitoring variable(s), (d) control (SR QoS) variable(s).
// A - Monthly Cost for binding to services on the Cloud LTPG < Monthly Cloud Cost Allowance -
// global policy
// Characteristics: (a) Adaptation-strategy: Served by A/Rs of the Cost Adaptation-strategy,
// (b) Horizon: Monthly, (c) Monitoring variable:
// costUseratio, (d) Control variables: primary costQoSReQ & secondary bitrateQoSReq.
// B - overall Monthly Data (in MBs) for Web and Cloud services LTPG: within Monthly
// Allowance by Mobile SP - Global & Appl. Policy
// Characteristics: (a) Adaptation-strategyategy: Served by A/Rs of the Cost Adaptation-
// strategy, (b) Horizon: Monthly, (c) Monitoring variable:
// dataUseratio, (d) Control variables: primary bitrateQoSReQ.
// C - Battery Power Mgmt LTPG over every battery Discharge period  - Global & Application
// Policy
// Characteristics: (a) Adaptation-strategyategy: Served by A/Rs of the Resource (battery)
// Adaptation-strategyategy, (b) Horizon: Dischargecycle period, (c)
// Monitoring variable: batteryUseratio, (d) Control variable: bitrateQoSReq.
// If the "dataSufficiency" context-variable is "Y", then staistically inferred modifiers are used
// in the resourceUseratio metrics for each LTPG, tuned to the specific usage pattern by the
// Statistical Analysis Module.


     // 8. Gradual Adaptation
// Gradual policy-based adaptation takes place, at least for the major QoS variables like:
// bitrateQoSReq and costQoSReq, not by assigning to adaptation-variables continuous (linear,
// quadratic etc) value expressions as functions of appropriate  continuous context metrics.


    // 9. The Six Adaptation Strategies in the Global and the Application Policy
// The Generic Operational Constraints Adaptation-strategyategy: specifying A/Rs ensuring
// generic operational constraints
// The Performance Optimization Adaptation-strategyategy: In Global & App Policy specifies
// the User Preference adaptation nehavior
// The Cost Optimization Adaptation-strategyategy: In the Global and App Policy handles the
// Monthly Cost Mgmt LTPGs
// The Resource (battery power) Optimization Adaptation-strategyategy: In Gobal & App Policy
// handles the Dischargecycle Battery LTPG
// The Monthly Data Volume LTPG Adaptation-strategyategy: In the Global  & App Policy
// manages the data volume LTPG
// The Disruption Minimization Adaptation-strategyategy: In the Global & App Policy manages
// runtime dusruption minimization
// End of general comments on the Policy


// The Global policy  specifies the settings for the operation of the Volare middleware and the
// SD & Binding Functionality
Policy Global{
Declarations{
    // Context-variables Declarations
    // Data Row No of data storage since the beginning of the History Database
   integer ContextVar dataRowNo == ContextMonitoringM.dataRowNo;
    // The current date with format: "DD/MM/YY"
    date ContextVar dateNow == ContextMonitoringM.dateNow;
    // The current time at which the data row context data are stored - format: hh:mm:ss
    time ContextVar rowStartTime == ContextMonitoringM.rowStartTime;
    // The time at the end of the recheckcycle (till next monitoring) - format: hh:mm:ss
    time ContextVar rowEndTime == SContextMonitoringM.rowEndTime;
    // Usage Model Row No of Cloud Service Discovery from the Usage Model
   integer ContextVar usgRowNo == ContextMonitoringM.usgRowNo;
    // Session No of Cloud Service Discovery since the beginning of the History Database
   integer ContextVar sessionNo == ContextMonitoringM.sessionNo;
    // string value indicating the selected User Preference Mode
    string ContextVar userPref == UPI.userPref;
    // float type value indicating the current bandwidth in KB/s
    float ContextVar bandwidth == ContextMonitoringM.bandwidthSensor;
    // percentage indicating the current battery level percent of the total battery capacity
    percentage ContextVar battery == ContextMonitoringM.batterySensor;
    // percentage value indicating at a new CSD session the battery drop caused by the
    //occurred non-cloud activity
    percentage ContextVar nonCloudbatteryDrop ==
ContextMonitoringM.nonCloudbatteryDrop;
    // percentage value indicating the battery level at the end of the current monitoring row
    percentage ContextVar endRowbattery == ContextMonitoringM.endRowbattery;
    //  Non Cloud MBs of data downloaded from Web referred to the current Recheckcycle
    float ContextVar webMBsRecheckcycle ==
ContextMonitoringM.webMBsRecheckcycleSensor;
```

```
    // The cost coefficient for converting the percentage costQoSReq & costQoSProv values in
    // real cost  ( = 0.0024 pounds/MBs)
    float ContextVar priceMax == UPI.priceMax;
    // float value E [0, 1], indicating the Performance Opt. Adaptation-strategy. Weight Coeff.
   // under the current User Preference
    float ContextVar wSperf == UPI.wSperf;
    // float value E [0, 1], indicating the Resource Opt. Strategy Weight Coefficient (SWC) under
    // the current User Pref.
    float ContextVar wSres == UPI.wSres;
    // float value E [0, 1], indicating the Cost Opt.  Strategy Weight Coefficient (SWC)under the
    // current User Pref.
    float ContextVar wScost == UPI.wScost;
    // float value E [0, 1], indicating the Disruption Minim. Strategy Weight Coeff. (SWC) under
    // the current User Pref.
    float ContextVar wSdisr == UPI.wSdisr;
    // float value E [0, 1], indicating the Data Volume Strategy Weight Coeff. under the current
    // User Pref.
    float ContextVar wSdat == UPI.wSdat;
    // float value, indicating the LTPG value for Monthly cost allowance for services on the
Cloud ( = 5 pounds/month)
    float ContextVar creditAllowance == UPI.creditAllowance;
    // float value, indicating the LTPG value for Monthly data allowance by the MNSP ( = 4000
    // MBs/month)
    float ContextVar dataAllowance == UPI.dataAllowance;

    // string value denoting the communication channel currently active

    string ContextVar commChannel == ContextMonitoringM.commChannel;

    // Cost incurred on CSD during the current Recheckcycle

    percentage ContextVar costQoSProv == ServiceRequest.cloudCostProv;

   // End of context-variables declarations


   // Calculation-variables Declarations

    //  Name of the current day

    string CalcVar dayName == StatisticalAnalysisM.dayName;
    // float value E [0, 1], indicating the Generic Operational Constraints  Adaptation-strategy
    //Weight Coefficient
    float CalcVar wSgen == 1.00;

    // integer value indicating the current Adaptation No
    integer CalcVar adaptationNo ==
StatisticalAnalysisM.sessionAdaptations.OverallHistory.Count;
    // integer value indicating the adaptations occurred during current session (value = 1 at
    // initial SD)
    integer CalcVar sessionAdaptations == StatisticalAnalysisM.adaptationNo.Session.Count;
```

```
// integer value indicating the rediscoveries occurred during current session (rediscoveries = 0
// at initial SD)
      integer CalcVar rediscoveries == sessionAdaptations - 1;

      // integer value indicating the current recheckCycle number within the CSD session

      integer CalcVar inSessionRecordNo == StatisticalAnalysisM.dataRowNo.Session.Count;

      // No of months from beginning of history storage, indicating the No of Monthly Periods

     integer CalcVar monthsNo == StatisticalAnalysisM.monthsNo;
      // integer value indicating the total number of days from the beginning of the data history
      // storage
      integer CalcVar daysNo == StatisticalAnalysisM.daysMonthly.Overallhistory.Count;

      // integer value indicating the current dischargecycleNo
      integer CalcVar dischargecycleNo ==
StatisticalAnalysisM.dischargecycle.overallHistory.Count;
      // Allow or not  usage-based Policy self-optimization at next month, if an LTPG has failed

      string CalcVar allowOptimization == "Y";

      //  Duration (in seconds) on CSD during the current monitoring Recheckcycle

      integer CalcVar cloudDurationRecheckcycle == 86400 * (rowEndTime - rowStartTime);


      // The battery LTPG Parameters & Monitoring Metrics
      // float value, indicating the LTPG value for minimum battery level at every Dischargecycle
      // ( = 20%)
      float ContextVar minBatteryLevel == UPI.minBatteryLevel;
      // The assumed duration of the battery Dischargecycle period for  the date-based criterion
      // (=330000 s)
      float CalcVar overallDurationDischargecycleEstim == 259200;

      //  Overall Duration (in seconds) on CSD during the current Recheckcycle
      integer CalcVar overallDurationRecheckcycle ==
StatisticalAnalysisM.overallDurationRecheckcycle;
      //  Overall Duration (in seconds) on CSD during the current battery Dischargecycle
      integer CalcVar overallDurationDischargecycle ==
overallDurationRecheckcycle.Dischargecycle.Sum;
      // percentage value indicating the Min battery level of the current Dischargecycle (after last
recharging)
     percentage CalcVar batteryDischargecycleMin ==
StatisticalAnalysisM.battery.Dischargecycle.Min;
      // The battery Duration Ratio  over the estimated Discharge cycle time period
     percentage CalcVar batteryDurationRatio == 100 * overallDurationDischargecycle /
overallDurationDischargecycleEstim;
      //  Cloud Duration (in seconds) on CSD during the current battery Dischargecycle
      integer CalcVar cloudDurationDischargecycle ==
StatisticalAnalysisM.cloudDurationRecheckcycle.Dischargecycle.Sum;
      // The asumed Cloud sessions duration within a Dischargecycle period for  the date-based
      // criterion  ( = 7000 s)
      float CalcVar cloudDurationDischargecycleEstim == 7000;

      // The decision-making float type auxiliary variable, modifying  the Battery LTPG Mgmt

     float AuxiliaryVar batteryVLowLevel;
      // decision-making auxiliary variable for Battery LTPG level availability: "HighBat" or
      // "LowBat" or "VLowBat"
     string AuxiliaryVar batteryLTPGLevel;


      // The Monthly Credit LTPG Parameters & Monitoring Metrics

      // float value, indicating the LTPG value for initial credit LTPG VLowLevel ( = 10%)

      percentage CalcVar creditInitialVLowlevel == 20;

      //  MBs of data downloaded on CSD during the current monitoring Recheckcycle

      integer CalcVar cloudMBsRecheckcycle == StatisticalAnalysisM.cloudMBsRecheckcycle ;

      // The cost of binding on a cloud service during the current monitoring Recheckcycle

     float CalcVar cloudCostRecheckcycle == StatisticalAnalysisM.cloudCostRecheckcycle;

      // The cost incurred on CSD  in the current Monthly period till now
      float CalcVar cloudCostMonthly ==
StatisticalAnalysisM.cloudCostRecheckcycle.Monthly.Sum;
      // integer value indicating the current day number  in the current month

      integer CalcVar daysMonthly == StatisticalAnalysisM.daysNo.Monthly.Count;

      // integer value indicating the total number of days in the current month

      integer CalcVar totalDaysOfMonth == StatisticalAnalysisM.totalDaysOfMonth;

      // The monthly Duration Ratio, common for both LTPGs
     percentage CalcVar monthlyDurationRatio == 100 * ((daysMonthly - 1 + rowStartTime) /
totalDaysOfMonth);
      // The usage-based Credit Use Ratio over the current Monthly period, for decision-making
      // on the Monthly Credit LTPG
     percentage CalcVar creditUseratio == 100 * (cloudCostMonthly / creditAllowance);
      // decision-making auxiliary variable for Credit LTPG Level Availability: "HighCredit" or
      // "LowCredit" or VLowCredit"
     string AuxiliaryVar creditLTPGLevel;

      // The decision-making float type auxiliary variable, modifying  Credit VLowLevel

     float AuxiliaryVar creditVLowLevel;


      // The Monthly Data Volume LTPG Parameters & Monitoring Metrics

      // float value, indicating the LTPG value for initial data volume LTPG VLowLevel ( = 10%)

      percentage CalcVar dataInitialVLowlevel == 10;

      // The MBs of data downloaded  on CSD  in the current Monthly period till now
      float CalcVar cloudMBsMonthly ==
StatisticalAnalysisM.cloudMBsRecheckcycle.Monthly.Sum;
```

```
  // The web MBs of data downloaded  through the MNSP but not on CSD  in the current
  // Monthly period
  float CalcVar webMBsMonthly == StatisticalAnalysisM.webMBsRecheckcycle.Monthly.Sum;

  // The sum: cloudMBsMonthly + externalMBsMonthly downloaded through the MNSP

 float CalcVar overallMBsMonthly == webMBsMonthly + cloudMBsMonthly;

  // The date-based Data Use Ratio over the current Monthly period, for decision-making on
  // the Monthly Data LTPG
 percentage CalcVar dataUseratio == 100 * (overallMBsMonthly / dataAllowance);

  // The decision-making float type auxiliary variable, modifying Data Volume VLowLevel

 float AuxiliaryVar dataVLowLevel;

  // Auxiliary variable for Data Volume LTPG level availability: "HighData" or "LowData" or
  //  "VLowData"
 string AuxiliaryVar dataLTPGLevel;



  // Parameters for Prefixed vs. Usage-based LTPG Metrics

  // The duration (in seconds) on CSD  in the current day till now
  float CalcVar cloudDurationDaily ==
StatisticalAnalysisM.cloudDurationRecheckcycle.Daily.Sum;
  // The MBs of data downloaded  on CSD  in the current day till now

 float CalcVar cloudMBsDaily == StatisticalAnalysisM.cloudMBsRecheckcycle.Daily.Sum;

  // The cost incurred on CSD  in the current day till now

  float CalcVar cloudCostDaily == StatisticalAnalysisM.cloudCostRecheckcycle.Daily.Sum;

  // The duration (in seconds) on CSD  in the current Monthly period till now
  float CalcVar cloudDurationMonthly ==
StatisticalAnalysisM.cloudDurationRecheckcycle.Monthly.Sum;
  // The date-based battery use ratio over the current Dischargecycle period, for decision-
  //making  on the battery LTPG
 float CalcVar batteryRefRate == 100 * ((100 - battery) / overallDurationDischargecycle) *
(overallDurationDischargecycleEstim / (100 - minBatteryLevel));
  // The current Monthly date-based Credit Reference Rate, for decision-making on the
  // Monthly Credit LTPG
 float CalcVar creditRefRate == 100 * creditUseratio / monthlyDurationRatio ;

  // The date-based Data Reference Rate Metric over the current Monthly period, for
  // decision-making on the battery LTPG
 float CalcVar dataRefRate == 100 * (overallMBsMonthly / (daysMonthly - 1 +
rowStartTime)) * (totalDaysOfMonth / dataAllowance);
  // the monthly average Min battery level of the Discharge cycles, to compare with the goal
minBatteryLevel
 float CalcVar batteryDischargecycleMinMonthlyAvg ==
StatisticalAnalysisM.batteryDischargecycleMin.Monthly.Avg;

  // Other Parameters and Metrics

  // The duration (in seconds) on Cloud Service Discovery (CSD) during the current Session

 float CalcVar cloudDurationSession ==
StatisticalAnalysisM.cloudDurationRecheckcycle.Session.Sum;
```

```
    // float value indicating the Max value of creditRatio in the current Monthly Period

    float CalcVar creditUseratioMax == StatisticalAnalysisM.creditUseratio.Monthly.Max;

    // float value indicating the Max value of dataRatio in the current Monthly Period

    float CalcVar dataUseratioMax == StatisticalAnalysisM.dataUseratio.Monthly.Max;
    // Runtime Disruption Minimization calculation-variables for restricting non-absolutely
    // necessary rediscoveries
    string CalcVar batteryLTPGLevelLastAdapt ==
StatisticalAnalysisM.batteryLTPGLevel.Session.LastAdaptation;
    string CalcVar creditLTPGLevelLastAdapt ==
StatisticalAnalysisM.creditLTPGLevel.Session.LastAdaptation;
    string CalcVar dataLTPGLevelLastAdapt ==
StatisticalAnalysisM.dataLTPGLevel.Session.LastAdaptation;
    // Runtime Disruption Minimization auxiliary-variable, values: "Current" or
    // "LastAdaptation"
    string AuxiliaryVar selectedLTPGLevel;

    // End of calculation-and auxiliary variables declarations


    // The middleware configuration-variables Declarations
    // The recheck period in seconds for the Context Monitoring Module to update context
    // values
    integer ConfigVar recheckRate == ContextMonitoringM.recheckRate;

    // The allowed max number of renegotiation attempts at CSD

    integer ConfigVar renegotiationAttempts == BindingM.renegotiationAttempts;

    // The allowed max percentage deviation from QoS request value at re-negotiation for CSD

    percentage ConfigVar renegotiationAdjustment == BindingM.renegotiationAdjustment;
    // The allowed max percentage deviation from QoS request to QoS provisioning values for
    // binding to a service
    percentage ConfigVar discoveryAccuracy == BindingM.discoveryAccuracy;

    // The default binding margin for a QoS variable without specific margin value

    percentage ConfigVar defaultBindingMargin == BindingM.defaultBindingMargin;
    // The number of seconds that the middleware will delay monitoring for binding
    // stabilisation
    integer ConfigVar rebindingDelay == BindingM.rebindingDelay;
    // It denotes whether after rebinding the middleware will delay monitoring for the
    // rebindingDelay interval
    string ConfigVar rebindingRecheck == BindingM.rebindingRecheck;
    // The min percentage discrepancy between new and last.adapted QoS values for
    // adaptation implementation
    percentage ConfigVar rediscQoSThreshold == AdaptationM.rediscQoSThreshold;
    // The specified number of Policy Consecutive Execution Cycles at each policy execution for
    // adaptation
    integer ConfigVar cyclesMax == AdaptationM.cyclesMax;
```

```
// The parameter for selection of predetermined options for the middleware, like: "Adapt"
// or "NoAdapt"
      string ConfigVar preferredVariant == AdaptationM.preferredVariant;

     // End of configuration-variables Declarations!


      // The QoS Adaptation-variables Declarations
      // The Service Request cost QoS value for a service on the Cloud as max cost/MB on the
      // service requested
     float QoSVar costQoSReq == AdaptationM.costQoSReq;

      // The Service Request Availability QoS value for the service requested on the Cloud

     float QoSVar availabilityQoSReq == ServiceRequest.activeRequest.availabilityQoSReq;

      // The Service Request Response Time QoS value for the service requested on the Cloud

     float QoSVar responseTimeQoSReq == ServiceRequest.activeRequest.responseTimeQoSReq;
      // The Service Request Reliability QoS value for the service requested on the Cloud
     // End of QoS-variables Declarations!
     // End of Declarations section!
};

Criteria{
     // 1st CEC: Setting default values to configuration-variables
     [1] Criteria G1_DEFAULT{
               default;
     };
     // 1st CEC:  "PREFIXED or UG-BASED CRITERIA FOR METRICS"
     [1] Criteria G1_BATTERYLTPG_USGBASED{
               daysNo > 30;
               monthsNo > 1;
               minBatteryLevel > batteryDischargecycleMinMonthlyAvg;
               allowOptimization = "Y";
     };
     // 1st CEC:  "PREFIXED or USG-BASED CRITERIA FOR CREDIT LTPG MONITORING"
     [1] Criteria G1_CREDITLTPG_USGBASED{
               daysNo > 30;
               monthsNo > 1;
               creditUseratioMax > 100;
               allowOptimization = "Y";
      };
     // 1st CEC:  "PREFIXED or USG-BASED CRITERIA FOR DATA VOLUME LTPG MONITORING"
     [1] Criteria G1_DATALTPG_USGBASED{
               daysNo > 30;
               monthsNo > 1;
               dataUseratioMax > 100;
               allowOptimization = "Y";
     };
     // 2nd CEC: Setting default values to configuration-variables
     [2] Criteria G2_DEFAULT{
               default;
     };
     // 2nd CEC: BATTERY AVAILABILITY LEVEL ASSESSMENT
     [2] Criteria G2_BATTERYLTPG_HIGH{
               battery >= 100 - batteryVLowLevel - 20;
               or
               userPref = "HighQuality";
     };
     [2] Criteria G2_BATTERYLTPG_VLOW{
               battery < batteryVLowLevel;
               userPref <> "HighQuality";
     };
     // 2nd CEC: CREDIT AVAILABILITY LEVEL ASSESSMENT
     [2] Criteria G2_CREDITLTPG_HIGH{
               creditRefRate <= 100;
               creditUseratio <= 100 - creditVLowLevel;
               or
               userPref = "HighQuality";
     };
     [2] Criteria G2_CREDITLTPG_VLOW{
               creditRefRate > 100;
               creditUseratio > 100 - creditVLowLevel;
               userPref <> "HighQuality";
     };
     // 2nd CEC: DATA AVAILABILITY LEVEL ASSESSMENT
     [2] Criteria G2_DATALTPG_HIGH{
               dataRefRate <= 100;
               dataUseratio <= 100 - dataVLowLevel;
               or
               userPref = "HighQuality";
     };
     [2] Criteria G2_DATALTPG_VLOW{
               dataRefRate > 100;
               dataUseratio > 100 - dataVLowLevel;
               userPref <> "HighQuality";
     };
     // 2nd CEC: RESTRICTING NON-ABSOLUTELY NECESSARY RUNTIME ADAPTATIONS
     [2] Criteria G2_RESTRICT_RUNTIME_ADAPTATIONS{
              inSessionRecordNo > 1;
              userPref <> "HighQuality";
      };
     // 3rd CEC: Default Subpolicy with default or generic operational constraints  adaptation-
     // statements
```

```
    [3] Criteria G3_DEFAULT{
            default;
    };
    // 3rd CEC: At User Pref. = "HIGHQUALITY" maximum attainable values are allowed for the
    // A/Vs without reservations
    [3] Criteria G3_HIGHQUALITY{
            userPref = "HighQuality";
    };
    // 3rd CEC: At User Pref. = "NORMAL" normally high attainable values are allowed for the
    // A/V under reservations
    [3] Criteria G3_NORMAL{
            userPref = "Normal";
    };
    // 3rd CEC: At User Pref. = "LOWCOST", costQoSReq 50% reduction under additional
    // reservations
    [3] Criteria G3_LOWCOST{
            userPref = "LowCost";
    };
    // 3rd CEC: At User Pref. = "SAVEBATTERY", increase recheckRate - declrease costQoSReq
    [3] Criteria G3_SAVEBATTERY{
            userPref = "SaveBattery";
    };
    // 3rd CEC: At HighCredit, allow max costQoSReq
    [3] Criteria G3_CREDITLTPG_HIGH{
            creditLTPGLevel = "HighCredit";
    };
    // 3rd CEC: At LowCredit, reduce costQoSReq
    [3] Criteria G3_CREDITLTPG_LOW{
            creditLTPGLevel = "LowCredit";
    };
    // 3rd CEC: At VLowCredit, reduce costQoSReq further
    [3] Criteria G3_CREDITLTPG_VLOW{
            creditLTPGLevel = "VLowCredit";
    };
};
     // End of Criteria section!

Subpolicies{
     // Start Subpolicies section!
    // 1st CEC: Adaptation-rules for specifying structural or algorithmic or configuration
    // parameter values
    Subpolicy G1_DEFAULT{
          defaultBindingMargin = 20 (wSgen * 0.10);
          overridesAsUpperLimit defaultBindingMargin = 50 (wSgen * 0.10);
          yields cyclesMax = 3 (wSgen * 0.10);
          overridesAsUpperLimit cyclesMax = 5 (wSgen * 0.10);
          batteryVLowLevel = minBatteryLevel + 10 (wSgen * 0.10);
```

```
       creditVLowLevel = creditInitialVLowlevel (wSgen * 0.10);
       dataVLowLevel = dataInitialVLowlevel (wSgen * 0.10);
};
Subpolicy G1_BATTERYLTPG_USGBASED{
       batteryVLowLevel = minBatteryLevel + 20 (wSres * 1.00);
};
Subpolicy G1_CREDITLTPG_USGBASED{
       creditVLowLevel = creditInitialVLowlevel + 10 (wScost* 1.00);
};
Subpolicy G1_DATALTPG_USGBASED{
       dataVLowLevel = dataInitialVLowlevel + 10 (wSdat * 1.00);
};
// 2nd CEC: LTPGs AVAILABILITY LEVEL ASSESSMENT
Subpolicy G2_DEFAULT{
      selectedLTPGLevel = "Current" (wSgen*0.10);
      batteryLTPGLevel = "LowBat" (wSgen*0.10);
      creditLTPGLevel = "LowCredit" (wSgen*0.10);
      dataLTPGLevel = "LowData" (wSgen*0.10);
};
Subpolicy G2_BATTERYLTPG_HIGH{
      batteryLTPGLevel = "HighBat" (wSres*0.50);
};
Subpolicy G2_BATTERYLTPG_VLOW{
      batteryLTPGLevel = "VLowBat" (wSres*0.50);
};
Subpolicy G2_CREDITLTPG_HIGH{
      creditLTPGLevel = "HighCredit" (wScost*0.50);
};
Subpolicy G2_CREDITLTPG_VLOW{
      creditLTPGLevel = "VLowCredit" (wScost*0.50);
};
Subpolicy G2_DATALTPG_HIGH{
      dataLTPGLevel = "HighData" (wSdat*0.50);
};
Subpolicy G2_DATALTPG_VLOW{
      dataLTPGLevel = "VLowData" (wSdat*0.50);
};
Subpolicy G2_RESTRICT_RUNTIME_ADAPTATIONS{
      selectedLTPGLevel = "LastAdaptation" (wSdisr*1.00);
      batteryLTPGLevel = batteryLTPGLevelLastAdapt (wSdisr*1.00);
      creditLTPGLevel = creditLTPGLevelLastAdapt (wSdisr*1.00);
      dataLTPGLevel = dataLTPGLevelLastAdapt (wSdisr*1.00);
 };
// 3rd CEC: QoS-variables calculations for SR adaptation
Subpolicy  G3_DEFAULT{
      yields costQoSReq = costQoSReq [-20] (wSgen * 0.10);
```

};

```
    overridesAsUpperLimit renegotiationAttempts  = 12  (wSgen * 0.10);
    overridesAsLowerLimit renegotiationAttempts = 0 (wSgen * 0.10);
    overridesAsLowerLimit discoveryAccuracy = 2 (wSgen * 0.10);
    recheckRate = 30 - 10*rediscoveries (wSgen * 0.50);
    overridesAsLowerLimit recheckRate = 5 (wSgen * 0.10);
    overridesAsUpperLimit rediscQoSThreshold = 12 (wSgen * 0.10);
    yields renegotiationAttempts = 10 (wSgen * 0.10);
    yields renegotiationAdjustment = 5  (wSgen * 0.10);
    yields rebindingRecheck = "FALSE"  (wSgen * 0.10);
    yields rebindingDelay = 10  (wSgen * 0.10);
    yields discoveryAccuracy = 5  (wSgen * 0.10);
    yields rediscQoSThreshold = 5 (wSgen * 0.10);
    availabilityQoSReq = 96/100 [30] (wSgen*0.10);
    responseTimeQoSReq = 70/100 [-24] (wSgen * 0.10);
 };
 Subpolicy G3_HIGHQUALITY{
    costQoSReq = costQoSReq [-20] (wSperf*1.00);
    overridesAll renegotiationAttempts = 10 (wSperf*0.20);
    overridesAll renegotiationAdjustment = 5  (wSperf*0.20);
    overridesAll rebindingRecheck = "FALSE"  (wSperf * 0.20);
    overridesAll rebindingDelay = 10  (wSperf*0.20);
    overridesAll discoveryAccuracy = 5  (wSperf*0.20);
    overridesAll recheckRate = 10  (wSperf*0.20);
};
Subpolicy G3_NORMAL{
    costQoSReq = costQoSReq [-20] (wSperf*1.00);
    overridesAll renegotiationAttempts = 10 (wSperf*0.20);
    overridesAll renegotiationAdjustment = 5  (wSperf*0.20);
    overridesAll rebindingRecheck = "FALSE"  (wSperf*0.20);
    overridesAll rebindingDelay = 10  (wSperf*0.20);
    overridesAll discoveryAccuracy = 5  (wSperf*0.20);
};
Subpolicy G3_LOWCOST{
    overridesAsUpperLimit costQoSReq = 0.80 * costQoSReq [-40] (wScost * 1.00);
    costQoSReq = 0.75 * costQoSReq [-20] (wScost * 1.00);
    overridesAll renegotiationAttempts = 10 (wScost*0.20);
    overridesAll renegotiationAdjustment = 5  (wScost*0.20);
    overridesAll rebindingRecheck = "FALSE"  (wScost*0.20);
    overridesAll rebindingDelay = 10  (wScost*0.20);
    overridesAll discoveryAccuracy = 5  (wScost*0.20);
};
Subpolicy G3_SAVEBATTERY{
    costQoSReq = costQoSReq [-20] (wSres*1.00);
    overridesAll renegotiationAttempts = 3 (wSres * 0.20);
    overridesAll renegotiationAdjustment = 6  (wSres * 0.20);
    rebindingRecheck = "FALSE"  (wSres * 0.20);
```

```
        rebindingDelay = 10  (wSres * 0.20);
        overridesAll recheckRate = 30 (wSres * 1.00);
        overridesAll discoveryAccuracy = 7  (wSres * 0.20);
        overrides recheckRate = 30 (wSres * 1.00);
    };
    Subpolicy G3_CREDITLTPG_HIGH{
        costQoSReq = costQoSReq [-20] (wScost*1.00);
    };
    Subpolicy G3_CREDITLTPG_LOW{
        overridesAsUpperLimit costQoSReq = 0.90 * costQoSReq (wScost * 0.50);
        costQoSReq =  2.20 * costQoSReq * (1 / (1 + creditRefRate/100)) * (1 / (1 +
creditUseratio/100)) * (1 / (1 + monthlyDurationRatio/100)) [-12] (wScost*1.00);
    };
    Subpolicy G3_CREDITLTPG_VLOW{
        overridesAsUpperLimit costQoSReq = 0.80 * costQoSReq (wScost * 0.50);
        costQoSReq =  costQoSReq * (1 / (1 + creditRefRate/100)) * (1 / (1 +
creditUseratio/100)) * (1 / (1 + monthlyDurationRatio/100)) [-12] (wScost*1.00);
    };
    // End of Declarations of the Global Policy
};
    // End of the Global Policy
};
```

APPENDIX B: The Application Policy Example

```
// Begin Application Policy
Policy VSTREAM{
Declarations{
    // App Policy context-variables Declarations
     // string value indicating the active appID
     string ContextVar appID == ServiceRequest.activeRequest.appID;
     // string value indicating the serviceID bound to till now
     string ContextVar serviceID == ServiceRequest.serviceID;
     // The corresponding to the bound to serviceID bitrateQoS on CSD during the current
     // Recheckcycle
     integer ContextVar bitrateQoSProv == ContextMonitoringM.bitrateQoSProv;
    // End of context-variables declarations of the Application Policy VSTREAM

     // Calculation-variables Declarations
     // The value of current session rediscoveries expressed per 5 min eq. Session on CSD  in the
     // current Session
    float CalcVar rediscovPer5min == rediscoveries * 300 / (cloudDurationSession + 1);
     // float value indicating the costQoSProv last value
    float CalcVar bitrateQoSProvLast == StatisticalAnalysisM.bitrateQoSProv.Monthly.Last;
    // End of calculation-variables declarations of the Application Policy VSTREAM

     // QoS Variables Declarations
     // QoS Variables that are common with the global QoS of the Global Policyare not declared
again - if they have the same name
     // The following QoS Variables are application-specific QoS variables
     // The Service Request bitrate QoS value for the service requested on the Cloud
    float QoSVar bitrateQoSReq == ServiceRequest.activeRequest.bitrateQoSReq;
     // The Service Request "frames per second" QoS value for the service requested on the
     // Cloud (typically: 23 to 27)
    integer QoSVar fpsQoSReq == ServiceRequest.activeRequest.fpsQoSReq;
    // End of QoS-variables declarations of the Application Policy VSTREAM

    // End of Declarations of the Application Policy
};

Criteria{
    // 3rd CEC Default Subpolicy for default values  and upper/lower limits for QoS variables
    [3] Criteria VSTREAM3_DEFAULT{
            default;
     };
    // 3rd CEC At User Pref. = "HighQuality" max attainable values for the unadapted value are
allowed for QoS variables
    [3] Criteria VSTREAM3_HIGHQUALITY{
            userPref = "HighQuality";
     };
    // 3rd CEC At User Pref. = "Normal" optimal attainable values are allowed for QoS variables
    [3] Criteria VSTREAM3_NORMAL{
            userPref = "Normal";
     };
    // 3rd CEC At User Pref. = "SaveBattery" optimal attainable values of 50%% the unadapted
value are allowed for QoS variables
    [3] Criteria VSTREAM3_SAVEBATTERY{
            userPref = "SaveBattery";
     };
    // 3rd CEC At User Pref. = "LowCost" optimal attainable values of 50%% the unadapted
value are allowed for QoS variables
    [3] Criteria VSTREAM3_LOWCOST{
            userPref = "LowCost";
    };
    // 3rd CEC: At High Battery Use Ratio, high attainable values are allowed for the QoSvars -
mainly bitrateQoSReq
    [3] Criteria VSTREAM3_BATTERYLTPG_HIGH{
             batteryLTPGLevel = "HighBat";
    };
    // 3rd CEC: At Low Battery Use Ratio, gradually reduced attainable values are set - mainly
bitrateQoSReq
    [3] Criteria VSTREAM3_BATTERYLTPG_LOW{
             batteryLTPGLevel = "LowBat";
    };
    // 3rd CEC: At Very Low Battery level, abrubtly reduced attainable values are set for
bitrateQoSReq
    [3] Criteria VSTREAM3_BATTERYLTPG_VLOW{
             batteryLTPGLevel = "VLowBat";
    };
    // 3rd CEC: At High Data Budget, high attainable values are set for QoSvars:  bitrateQoSReq
    [3] Criteria VSTREAM3_DATALTPG_HIGH{
             dataLTPGLevel = "HighData";
    };
    // 3rd CEC: At Low Data Budget, low attainable values are set for QoSvar: bitrateQoSReq
    [3] Criteria VSTREAM3_DATALTPG_LOW{
             dataLTPGLevel = "LowData";
    };
    // 3rd CEC: At Very Low Battery level, abrubtly reduced attainable values are set for
bitrateQoSReq
    [3] Criteria VSTREAM3_DATALTPG_VLOW{
             dataLTPGLevel = "VLowData";
    };
   // End of Criteria section of the Application pPolicy
};
Subpolicies{
     Subpolicy VSTREAM3_DEFAULT{
         overridesAsUpperLimit bitrateQoSReq = 0.8 * bandwidth [-40] (wSgen*1.00);
         yields bitrateQoSReq = 0.8 * bitrateQoSReq [-15] (wSgen*0.10);
         yieldsAll costQoSReq = costQoSReq [-18] (wSgen*0.10);
         yields fpsQoSReq = 0.96*fpsQoSReq [10] (wSgen*0.10);
        availabilityQoSReq = 96/100 [20] (wSgen * 0.80);
        responseTimeQoSReq = 60/100 [-24] (wSgen * 0.50);
        rediscQoSThreshold = 5 (wSgen * 0.50);
     };
     Subpolicy VSTREAM3_HIGHQUALITY{
        bitrateQoSReq = bitrateQoSReq [-10] (wSperf*1.0);
        costQoSReq = costQoSReq [-20] (wSperf*1.00);
        fpsQoSReq = fpsQoSReq [16] (wSperf*0.50);
     };
    Subpolicy VSTREAM3_NORMAL{
        bitrateQoSReq = bitrateQoSReq [-20] (wSperf*1.00);
        costQoSReq = costQoSReq [-20] (wSperf*1.00);
        overridesAsUpperLimit fpsQoSReq = 0.98*fpsQoSReq [20] (wSperf*1.00);
        fpsQoSReq = 0.98*fpsQoSReq [20] (wSperf*1.00);
    };
    Subpolicy VSTREAM3_LOWCOST{
        costQoSReq = 0.78*costQoSReq [-20] (wScost*1.00);
        bitrateQoSReq = bitrateQoSReq [-20] (wScost*1.00);
    };
    Subpolicy VSTREAM3_SAVEBATTERY{
        overridesAsUpperLimit bitrateQoSReq = 0.50*bitrateQoSReq [-12] (wSres*0.50);
        bitrateQoSReq = 0.45 * bitrateQoSReq [-20] (wSres*1.00);
        costQoSReq = costQoSReq [-20] (wSres*1.00);
        overridesAsUpperLimit fpsQoSReq = 0.93*fpsQoSReq [16] (wSres*1.00);
        fpsQoSReq = 0.93*fpsQoSReq [16] (wSres*1.00);
    };
     Subpolicy VSTREAM3_BATTERYLTPG_HIGH{
        bitrateQoSReq = bitrateQoSReq [-20] (wSres*1.00);
        fpsQoSReq = fpsQoSReq [16] (wSres*1.00);
     };
    Subpolicy VSTREAM3_BATTERYLTPG_LOW{
        overridesAsUpperLimit bitrateQoSReq = 0.50 * bitrateQoSReq [-20] (wSres*1.00);
        bitrateQoSReq = 0.60 * bitrateQoSReq * (1 / (1 + batteryRefRate/100)) * (2 -
battery/100) * (1 / (1 + batteryDurationRatio/100)) [-12] (wSres*1.00);
        fpsQoSReq = 0.93*fpsQoSReq [12] (wSres*1.00);
    };
    Subpolicy VSTREAM3_BATTERYLTPG_VLOW{
        overridesAsUpperLimit bitrateQoSReq = 0.20 * bitrateQoSReq [-20] (wSres*1.00);
        bitrateQoSReq = 0.30 * bitrateQoSReq * (1 / (1 + batteryRefRate/100)) * (2 -
battery/100) * (1/(1 + batteryDurationRatio/100)) / 2 [-12] (wSres*1.00);
        fpsQoSReq = 0.90*fpsQoSReq [16] (wSres*1.00);
        overrides recheckRate = 30 (wSres*0.80);
    };
    Subpolicy VSTREAM3_DATALTPG_HIGH{
        bitrateQoSReq = bitrateQoSReq [-20] (wSdat*1.00);
        fpsQoSReq = fpsQoSReq [20] (wSdat*1.00);
        defaultBindingMargin = 24 (wSdat*1.00);
    };
    Subpolicy VSTREAM3_DATALTPG_LOW{
        overridesAsUpperLimit bitrateQoSReq = 0.50 * bitrateQoSReq (wSdat * 1.00);
        bitrateQoSReq =  bitrateQoSReq * (1 / (1 + dataRefRate/100)) * (1 / (1 +
dataUseratio/100)) * (1 / (1 + monthlyDurationRatio/100)) [-12] (wSdat*1.00);
        fpsQoSReq = 0.90*fpsQoSReq [10] (wSdisr*1.00);
    };
    Subpolicy VSTREAM3_DATALTPG_VLOW{
        overridesAsUpperLimit bitrateQoSReq = 0.20 * bitrateQoSReq (wSdat * 1.00);
        bitrateQoSReq =  bitrateQoSReq * (1 / (1 + dataRefRate/100)) * (1 / (1 +
dataUseratio/100)) * (1 / (1 + monthlyDurationRatio/100)) / 4 [-12] (wSdat*1.00);
        fpsQoSReq = 0.85*fpsQoSReq [10] (wSdat*1.00);
    };
    // End of the Subpolicies Part
   };
// End of Application Policy
};
```

APPENDIX C: Middleware & Policy Testing and Verification

When it comes to testing & verification the VOLARE middleware is capable of operating:

a) On real data, providing DCAA support to an application with the User activating a service request for service discovery and binding on the Cloud;

b) On virtual context data on an inserted Test Suite or individual test cases, by-passing Context Monitoring and feeding virtual context data in sequence from a Context Data Matrix.

As part of its functionalities, the middleware keeps record of the Context & Adaptation data in the Context & Adaptation History Database (CAHiD). The CAHiD data may be extracted and then may be used independently of the middleware for static verification, evaluation or validation purposes, since the mobile device cannot have the resources required for a static evaluation of a large data volume and external Automated Static Verification Tools have been designed.

In the following paragraphs we describe the dynamic testing and verification techniques selection for the middleware and the adaptation policy, as well as the strategy for test suites generation on each technique and the adequacy criteria. Based on the dynamic testing context data & results, a static analysis methodology and tools have been designed for detecting policy logic “irregularities” that do not stop the program flow but constitute rule faults.

1. Strategy & Tools for Dynamic & Static Verification

The middleware operation is basically verified through the following steps:

a) Dynamic Testing

Firstly, we test the middleware and the Adaptation Policy observing the middleware CAA operation on real usage data, recording the context & adaptation data and verifying that it operates satisfactorily,

Secondly, Automated Dynamic Testing on appropriately designed Test Suites with adequacy criteria will be implemented.

b) Static Verification of the Context & Adaptation Data derived

Based on the context and adaptation data of the dynamic testing, Automated and/or Tester-guided Static Analysis is implemented, for rules faults and logical irregularities detection and correction.

Additionally, we compare the extracted CAHiD data to an already independently validated application on Policy Engine simulation, called PEVApp, with execution data on the same Composite Policy and on the same real and/or virtual test cases.

c) Testing Techniques Selection

For the Testing & Verification of the middleware and the Policy, we shall select appropriate Grey Box Testing Techniques for the Middleware and White Box Testing Techniques for the Adaptation Policy from the testing repertory proposed by the computing community. Since our purpose is to test the middleware and at the same time verify its adaptation logic, the Policy, we need to combine the two test activity categories.

We shall launch both Dynamic Testing of the middleware and the Policy on appropriately designed Test Suites and Static Analysis of the derived and recorded Context & Adaptation Data.

The choice of the most appropriate testing techniques for the program and the Policy under test as well as the extent and targeting of coverage and adequacy criteria that define the number of test cases is a major strategic decision. We choose as Verification Techniques, those that in combination provide joint validation on the subjects of our Verification and Validation, namely: (a) the Middleware Verification on APSL-compatible policies; (b) The Policy Verification for Rules Faults; (c) the joint Middleware – Policy Validation.

A hybrid solution of several Software Testing & Verification Techniques will be implemented for maximum efficiency, using Automated Testing & Verification Tools for Automated Testing Suites Design & Generation and Automated Static Results Analysis, with developer-guided fine-tuning to focus on suspected output anomalies.

2. Dynamic Testing Techniques

Automated joint Dynamic Testing of the Middleware and the Policy logic will be implemented by the repeated execution on the middleware on context data Test Suites, in addition to real-time operation context & adaptation data.

On Dynamic Testing, we will verify the program for failures, excessive execution duration and unreasonable results.

Testing Techniques Selection

Some of the most widely implemented software testing techniques are adopted, namely: (a) for code coverage, the condition/predicate-coverage, (b) the domain-testing strategy for boundary values of the context sub-domains [ ][ ]. These techniques, based on fine-grained criteria, guide the generation of test suites so that most different paths will execute and weaknesses will become apparent.

a) The Predicate-coverage & Statement-coverage Testing Strategy

We choose from the Code-based techniques the Predicate-coverage & the Statement-coverage Strategies **based on the Adaptation Policy predicates**, as it is the most appropriate for combining at the same time testing of the Middleware and the Policy, not only on dynamic verification where it is the Policy predicates that influence path/branch change, but also in view of the planned input/output Static Verification for Rules Fault Detection and Adaptation Logic inconsistencies.

The testing strategy is designed to detect errors in the control flow of the middleware in application of the Policy. The predicates of the Policy partition the input space into a set of mutually exclusive context sub-domains. Values from each context sub-domain correspond to a particular program path and represent input data points which cause that path to be executed.

As coverage targets and adequacy criteria, we set: 100% for Policy Predicate-coverage and 100% for Policy Statement-coverage. Of course, if some statements or predicates prove “infeasible”, we will try to exercise them though manual test cases or they will be identified as faulty “dead” predicates or statements.

b) The Domain-Testing Strategy

Domain-testing will also be partly adopted as one of the main testing techniques, in combination with the Predicate-coverage Testing technique, for the Dynamic Verification of the Middleware and of the Policy through selection of boundary values test cases of the context sub-domains.

It is essential to orient Dynamic Testing of the middleware with or without the main Policy, to context values equivalence classes that present the higher probability of fault presence. Such equivalence classes include boundary values of context sub-domain, alternative variables types, as well as sub-domains with higher fault percentage.

c) Usage-based Testing Approach

Since it is essential that User generated input will be preferentially tested and in our middleware the User input is fairly limited (in the Policy, typically the User Preference selection only), all four alternative input values for User Preference will be exhaustively used in building the Testing Context Data Matrix.

For the above reasons, the Automated Test Suite Design will include hybrid elements for the above techniques and will be developer-guided to add additional required test cases.

3. Static Testing Techniques

As documented in research papers [55][70][71], the Dynamic Verification techniques cannot discover some program failures, neither is it possible to increase much the number of test cases due to the combinatorial explosion problem with the number of variables and their sub-domains. Thus, we choose a combination of the Dynamic Testing with Static Analysis of the context & adaptation data.

The Static Analysis of the Context & Adaptation Data aims at identifying Rules Faults through indications of "*irregular adaptation behaviour patterns*", like:

- "Dead" policy rules, predicates, statements or adaptation-values not selected over a representative set of context data covering the whole context domain, or eventual live-lock or dead-lock rules,
- Illogical adaptation results that are in conflict with expected relations between context & adaptation data, indicating an anomaly caused by errors in Policy Rules Policy Rules,
- Irregular adaptation results values clustering on several sub-ranges, instead of covering the whole variables ranges.

Such *irregular adaptation results patterns* indicate faults in rules that lead to erroneous adaptation. Two main Static Testing Strategies will be adopted:

4. Rule Faults Detection Strategy

The Rule Faults Detection strategy is described in [55], with algorithms that allow Rule Fault patterns to be detected. However, the Weight-based Policies, due to specific features, are not compatible with the described symbolic testing technique and a different technique is developed, requiring first the Dynamic Testing context & adaptation data, on which appropriate rule fault detection algorithms may run to identify rules with suspected faults.

5. Metamorphic Relations Verification Testing Technique

Metamorphic Relations Testing aims at investigating existing known or expected relations between input and output data produced of the dynamic test cases and verifying that there is no anomaly by violation of these relations [ ][ ]. Of course these relations are scenario-specific but we consider them very important for identifying otherwise hidden logical weaknesses in the Policy, by evaluating if the selected and appropriately sorted sets of results correspond to selected input data.

6. Middleware Verification Implementation

First, we will start with testing and verifying the middleware on Test Policies.

a) Preliminary Functional Testing with Test Policy & Debugging

   a. Build & Verify Test Policies of increasing complexity

   A simple virtual Preliminary Test Policy will be built, with:

   - Declarations statements on all APSL – supported types of variables of all Repositories,
   - Predicates with all logical & arithmetic operators & operands
   - Adaptation-rules on all supported types
   - All other features of the Policy Specification Language.

   This policy will be verified each time it is modified, on syntactic correctness.

   b. Manual/Automated Functional Testing, Debugging

   Then a Manual Dynamic Testing and debugging process will be implemented.

b) Dynamic Testing with the Real Policy on a Test Suite

   a. Build & Verify Real Adaptation Policy

   The real Adaptation Policy will be built and verified on syntactic correctness.

   b. Test Suite Generation

   Based on the real Policy, a Test suite of context data is prepared by the Automated Testing Suite Generator Tool. The Automated Testing Suite Generator applies the developer’s instructions on Testing Strategies selection for the Test Suite generation.

   c. Automated Dynamic Testing

   The Automated Repeated Policy Execution Tool is used and dynamic testing is implemented, based on the Test Suites. The results are stored for further static analysis.

c) Joint Middleware & Policy Verification Strategy

For reasons explained in the following paragraphs, due to the specific features of the VOLARE APSL and unlike the conventional Action-based rules, a Weight-based Policy is difficult to be submitted to Static Verification on Rule Faults Detection on symbolic analysis only without dynamic testing results. Consequently, the real Policy Static Verification will be implemented jointly with the already partly verified Middleware.

Verification of the Real (not Test) Adaptation Policy will be based:

- On Automated Syntactic Correctness Verification by the appropriate Tool;
- On Automated Dynamic Testing by the middleware on an appropriately generated Test Suite, as well as on Real-Time operation;
- On Automated and Tester-guided Static Testing (Analysis) of the extracted Context & Adaptation Data from the Dynamic Testing and from the real-time operation.

7. Automated Dynamic Verification of the Middleware and Policy

a) Automated Test Suites Design & Generation

For each context-variable used in Criteria conditions (or adaptation-statements), all threshold values are automatically identified for the sub-domains referenced in the Criteria conditions (or eventually in adaptation-statements). This concerns:

- the recording of every alternative value referenced for the discrete context-variables Boolean or string-type, and
- the identification of threshold values for each sub-domain referenced for each continuous (numeric) context-variable.

For each context-variable declared and referenced in the Policy File, its domain of values is identified:

- For Boolean context-variables, the alternative states are True and False.
- For every string-type context-variable, the alternative values are identified and stored.
- For numeric context-variables, all its upper and lower sub-domain values are identified. For instance if for a percentage variable, we identify the following three context conditions, then we identify:
    - contextVar1 > val1
    - contextVar1 = val2
    - a * contextVar1 - val3 =< 0 /* a <> 0 */

with sorted constants, say: val1 > val2 > val3/a, and additionally the absolute higher and lower values (say: 0 and 100), we have the following upper/lower boundary values for the sub-domains: 100, val1, val2, val3/a, 0.

Use of the **Automated Testing Suite Generator Tool of the PEVApp** is made for the automated Testing Suite generation, based on alternative options provided to the developer for the suite design.

The program implements Test Suite generation for each sub-domain of a context-variable, based on predetermined selectable strategies. A 2-dimensional Context Data Matrix is prepared, with first row header each context-variable and in each next row, each test case.

b) Automated Dynamic Verification based on the Test Suites

Repeated policy execution is implemented, feeding as context data each row of data values of the Test Suite and recording at each Policy Execution, not only the context data & adaptation results as well as the intermediate calculation values, but additionally for further Static Analysis:

- The Policy statements executed
- the Criteria/Subpolicies or individual predicates satisfied,
- the A/Rs selected and executed.

The Dynamic Testing based on the Test Suites will be implemented on the Mobile middleware on the device and – for comparison purpose - on the already independently validated simulated Policy Engine on PEVApp.

8. Automated Static Verification Strategy

Further rules verification is of paramount importance for the correct operation of rule-based CAA middleware-based applications for detecting faults that escaped the Dynamic Testing. We introduce an approach and an Automated Static Verification Tool of the Context & Adaptation data derived by the Dynamic Testing, consisting in rule fault detection through appropriate algorithms, looking for adaptation behavior different from the apparent intention of the policy developer.

The verification of the adaptation policy logic is based on the identification of adaptation rules fault patterns for the VOLARE APSL-compatible Policy Files. The following papers [55] [70] have influenced our work, but we follow a different policy-based DCAA Rules Fault Detection approach. The major differences with the referenced work on rule fault detection for CAAA’s and our work are due the following two main reasons:

- The VOLARE APSL compatible Policy Files at policy execution may match, select and execute multiple A/Rs on the same head predicate, which is considered an inconsistency by the classical verification approaches. In VOLARE the Conflict Resolution Directives allow for participative weighted contribution to the adaptation results. Consequently the inconsistency check is not valid in VOLARE Policy Files.
- The VOLARE approach on adaptation policy logic verification needs to make use of the APSL specific, proprietary features, like (i) rule priority assigning through keywords and (ii) the weight value at each A/R, that require Policy Execution to define which are the selected modalities that define a system state at each context condition. Without Policy Execution, the state matrices cannot be evaluated.

Consequently, in order to launch static verification for rules faults, we need to make use of the Dynamic Testing adaptation data on appropriate test suites and then static verification on the derived data is implemented.

a) Comparison of Dynamic Testing Results with the Simulation

The Context & Adaptation Data of the CAHiD are extracted and evaluated that they are the same with those derived by Repeated Policy Execution by the already validated simulated Policy Engine of PEVApp on the same Test Policy. Otherwise, a fault is detected.

b) Automated Static Rule Faults Detection

The Fault Patterns that have been identified for VOLARE APSL compatible Policy Files based on syntactic and behavioral rule faults are described below. A Fault Pattern is deduced from the “reasonable” demand that in a Policy, each adaptation-rule and each adaptation action encoded in the Policy, should satisfy some requirements of adaptation reasoning “correctness”. Lack of this “correctness” identifies a Fault Pattern for the specific rule, predicate, or adaptation action.

The following “correctness requirements” have been identified for the adaptation-rules in adaptation logic and their lack detects the existence of the relevant rule fault to the verification approach.

a. Criteria Liveness Requirement

There should be at least one set of context data, which leads to the satisfaction of all the predicates of a Criteria conditions group

b. Subpolicy Liveness Requirement

There should be at least one set of context data, which leads to the selection and execution of at least one adaptation-statement in the corresponding Subpolicy.

c. Rule Liveness Requirement

There should be at least one set of context data, which leads to the selection and execution of each Rule in every Subpolicy.

d. Adaptation Action Liveness Requirement

There should be at least one set of context data, which leads to the selection (and execution) of each adaptation-statement of a non-numeric A/R in all its alternative non-numeric values. A Boolean A/V should take both values at different context values, a string-type A/V should take all alternative values identified in the Policy, and a numeric adaptation-variable at least one value.

e. 1st Stability Requirement – Preventing Adaptation Cycles

Adaptation Cycles identified by the execution of the same adaptation at a specific context value/range should be prevented.

f. 2nd Stability Requirement – Preventing Adaptation Races

Adaptation Races, identified through execution of the same sequence of adaptations at a specific context value/range, should be prevented.

The Automated Rule Faults Detection is implemented through the **Automated Rule Faults Detection Tool** of the **PEVApp.**

a) Static Analysis on Metamorphic Relations

Metamorphic Relations are expected relations not in the input/output data of a test case, but between sets of input and output data. Faults are detected if there is anomaly by violation of these relations [ ][ ]. Of course these relations are scenario-specific but we consider them very important for identifying otherwise hidden logical weaknesses in the Policy, by evaluating if the selected and sorted sets of results corresponding to selected input data are “the expected ones” by the developer.

In our scenario, we have **expected relations or trends** that selected context – adaptation data sorted according to certain criteria should exhibit. For instance after multiple Policy Executions on a Test Suite, we may have context & corresponding results appropriately selected and sorted. On these selected and sorted data an automated verification may be implemented on the conformance to expected behaviour:

a. Evaluating bitrate QoS Request values vs. increasing bandwidth values

Keeping all other context fixed, or within specified limits, we would expect a non-negative trend on consecutive results values corresponding to ascending context values. If an anomaly is identified, it requires a careful guided review of the Policy.

b. Evaluation of several A/V values under same context but different User Preference

With all other context values fixed, some numeric A/Vs values calculated under different User Preference, need to follow some relation, like:

$A/V)_{HighQuality}$ >= $A/V)_{Normal}$ >= { $A/V)_{LowCost}$. $A/V)_{SaveBattery}$}
(1)

We intend to identify and use such “reasonable relations” to validate the Policy (and the middleware) on logical as well as eventual functional errors, by Relations Evaluation of the Context & Adaptation data, collected from dynamic testing on appropriately designed test suites and appropriately sorted.

9. Related Literature on Middleware & Policy Verification

Policy Verification

Sama et al [55] have published an approach for static verification of the adaptation logic of CAAAs. VOLARE shares the basic thinking on Rules Fault Detection and has also developed an approach for Policy File verification, but adjusted to the VOLARE APSL and adaptation-rules. The basis of the approach in [55] is based on the consistency algorithm, which is not relevant in VOLARE since multiple adaptation-rules on one head predicate may be selected under a certain context. Consequently a different approach has been adopted for detecting faults and anomalies, based on the analysis of the Context & Adaptation Data derived by the Dynamic Testing, as explained extensively in paragraph 7.5.

Metamorphic Relations are expected relations not in the input/output data of a test case, but between sets of input and output data of the Test Suite results. Faults are detected if an anomaly is detected by violation of these relations [70].

10. Middleware & Policy Results Evaluation & Validation

a) Verification Adequacy Criteria

As the verification activities will be implemented on the middleware and the real Policy, we will need to deduce Verification Adequacy Criteria on the extent and depth of Dynamic and Static Testing.

At the present, we have set as adequacy criteria, 100% predicate-coverage and 100% statement-coverage of the Policy, or as high as it is attainable, to account for eventual “infeasible” predicates or statements, that will be tested manually.

On the real data operation of the mobile, testing will be focused mostly on the other middleware components.

b) Faulty Rules Correction

The errors identified by execution failure will be debugged. The faulty rules are identified directly or indirectly through the detection of faults or “non-correctness” in the results.

Rule correction by modification of the Criteria conditions, priority etc, or elimination of them if they are redundant, is the developer/tester’s responsibility.

Remarks – Discussion on the Verification Challenges

a) The “technical” faults issue

Some of the faults detected may be just “technical” faults in the sense that, as the middleware Global Policy is generic for supporting different alternative applications, some rules may not be activated for the current application or the current Fine-tuned Application Profile (i.e. threshold settings) of the User Choices. It is possible that some adaptation action or rule triggering (selection and calculation) may be reserved for other Apps and since the Global Policy cannot be application specific, a context sub-domain of a rule may not be live for all apps.

b) Complicated Context – Adaptation Model

For scenarios/Policies with complicated Context & Adaptation Model and complex propositional context variables (i.e. predicates) the Automated Test Suite Design & Generation may not be possible. In such cases, Test Suite design or results evaluation requires developer guidance.

c) The Combinatorial Explosion Issue in Dynamic Verification

It is apparent that the test cases number of Testing Suites (i.e. the alternative context data sets of values) is the product of the number of alternative values for each context-variable. As the number of the context-variables and/or the number of alternative context data values generation or the different sub-domains for the numeric variables increase, there may be a combinatorial explosion of data/results. This is why both the dynamic and static Policy File verification takes place at an appropriate workstation.

d) Static Metamorphic Relations Testing – The advantage of the User Preference Model

As it is apparent, our approach on the Adaptation Policy Logic Rule Detection examines also the adaptation domain coverage by the Policy Logic and is not limited only in individual rule fault patterns [55] and “adaptation-action liveness”, but identifies adaptation “results” violating expected adaptation behavior, through the metamorphic testing strategy.

Critical in Static Verification of rules faults is the User Preference Model with the policy-based modes that are designed to impose a *scaled* adaptation behavior on some A/Vs. The conclusion is that for Static Verification on Relations between context & results, it is important to appropriately design the User Preferences Model, so that it can assist a gradual, scaled, adaptation behavior that is more easily verified.

11. Automated Composite Policy Verification Process

A **Policy Execution & Verification Application (PEVApp)** has been designed simulating a VOLARE APSL-compatible Policy Engine, with Automated Verification Tools that assist the developer on Testing & Verification. By introducing relevant context data, Policy Execution is enacted providing and recording the context data and the intermediate and final adaptation results.

PEVApp is operating on a workstation and has been independently validated as operating correctly on Policy Execution against a Test Suite of context data with recorded and verified adaptation results conforming to the VOLARE APSL and the Composite Adaptation Policy.

It also helps validating the middleware, since it is verified that it consistently provides the same results with PEVApp on Policy Execution. PEVApp includes the Verification Tools referenced below.

Since the Dynamic & Static Verification may lead to a large number of data and results with different storage, sorting, evaluation or graphic visualization needs, PEVApp and the Automated Verification Tools are based on MS EXCEL 10 spreadsheets, coded in Visual Basic for Applications (VBA).

Another basic reason for this decision is the fact that policy logic verification is open ended concerning verification and evaluation on non-predefined aspects and EXCEL provides the developer/tester or **the advanced User wishing to test a policy** with an excellent tool with many options for further fine-grained tester-guided evaluation as well as storage of the context & adaptation data for future testing and reference.

12. Policy Testing & Verification Tools

For testing and verification purposes, we have developed a simulated policy engine application, called in this work Policy Execution & Verification Application – PEVApp, for offline simulated policy execution and verification of adaptation policy files compatible to the Volare APSL. This application allows offline testing & verification of every new global or application policy on syntactic correctness, on semi-automated developer-guided generated test suites.

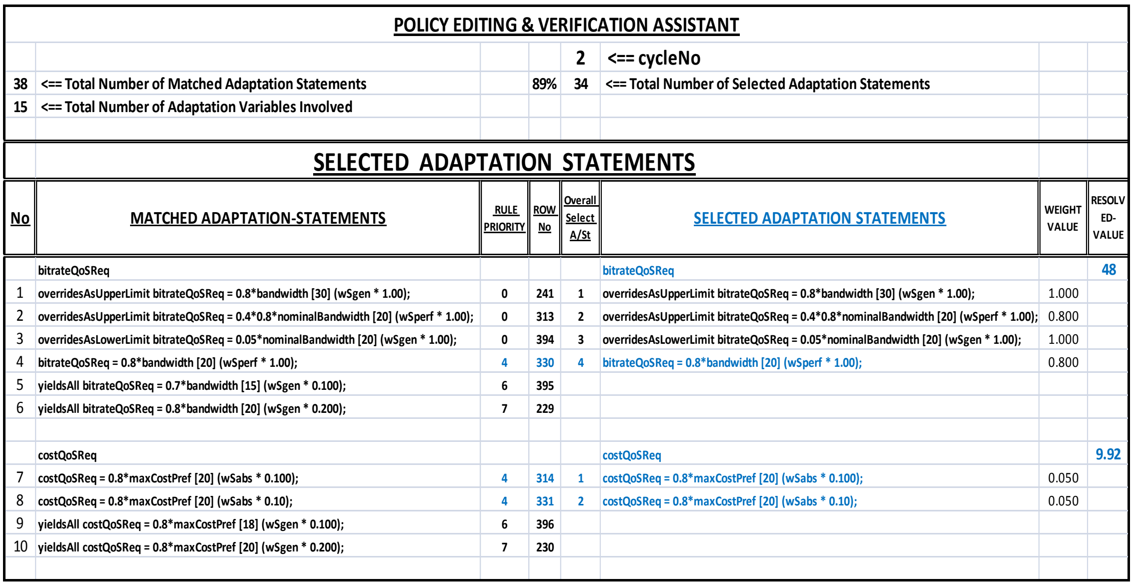
A more complete description on Policy Testing & Verification implemented including adaptation-rules fault detection is given in Appendix C and the PEVApp User Guide in Appendix E.

Several automated or developer-guided testing & verification tools have been developed custom-made to the Volare APSL, and are included in the PEVApp:

a) A Simulated Policy Engine & Rule Verification Assistant

A Simulated Policy Engine for offline policy execution on Volare APSL-compatible policy files has been constructed and independently verified. A Policy Execution and Verification Application (PEVApp) has been developed.

Table 0-1 – A view of the Policy Editing & Verification Assistant Tool



POLICY EDITING & VERIFICATION ASSISTANT

				2	<== cycleNo		
38	<== Total Number of Matched Adaptation Statements		89%	34	<== Total Number of Selected Adaptation Statements		
15	<== Total Number of Adaptation Variables Involved						

SELECTED ADAPTATION STATEMENTS

No	MATCHED ADAPTATION-STATEMENTS	RULE PRIORITY	ROW No	Overall Select A/St	SELECTED ADAPTATION STATEMENTS	WEIGHT VALUE	RESOLVED-VALUE
	bitrateQoSReq				bitrateQoSReq		48
1	overridesAsUpperLimit bitrateQoSReq = 0.8*bandwidth [30] (wSgen * 1.00);	0	241	1	overridesAsUpperLimit bitrateQoSReq = 0.8*bandwidth [30] (wSgen * 1.00);	1.000	
2	overridesAsUpperLimit bitrateQoSReq = 0.4*0.8*nominalBandwidth [20] (wSperf * 1.00);	0	313	2	overridesAsUpperLimit bitrateQoSReq = 0.4*0.8*nominalBandwidth [20] (wSperf * 1.00);	0.800	
3	overridesAsLowerLimit bitrateQoSReq = 0.05*nominalBandwidth [20] (wSgen * 1.00);	0	394	3	overridesAsLowerLimit bitrateQoSReq = 0.05*nominalBandwidth [20] (wSgen * 1.00);	1.000	
4	bitrateQoSReq = 0.8*bandwidth [20] (wSperf * 1.00);	4	330	4	bitrateQoSReq = 0.8*bandwidth [20] (wSperf * 1.00);	0.800	
5	yieldsAll bitrateQoSReq = 0.7*bandwidth [15] (wSgen * 0.100);	6	395				
6	yieldsAll bitrateQoSReq = 0.8*bandwidth [20] (wSgen * 0.200);	7	229				
	costQoSReq				costQoSReq		9.92
7	costQoSReq = 0.8*maxCostPref [20] (wSabs * 0.100);	4	314	1	costQoSReq = 0.8*maxCostPref [20] (wSabs * 0.100);	0.050	
8	costQoSReq = 0.8*maxCostPref [20] (wSabs * 0.10);	4	331	2	costQoSReq = 0.8*maxCostPref [20] (wSabs * 0.10);	0.050	
9	yieldsAll costQoSReq = 0.8*maxCostPref [18] (wSgen * 0.100);	6	396				
10	yieldsAll costQoSReq = 0.8*maxCostPref [20] (wSgen * 0.200);	7	230				

b) Automated Test Suite Generator

A test suite is automatically generated based on several testing strategies for policy predicate coverage, domain testing, context sub-domains boundary values, as well as domain-testing suggested extra values, by letting the developer/tester to select one from several pre-defined alternative options for context data generation.

c) Automated Test Suite Repeated Policy Execution

On the test suites generated, PEVApp enables automated offline execution on the Simulated Policy Engine application and context and adaptation data recording for further automated static analysis.

In the Table 6-4 below Policy Editing & Verification Assistant Tool is depicted, demonstrating which adaptation-rules are matched and selected under a given context data test case, priority settings, weight values and adaptation results. This tool may assist the developer at adaptation policy authoring.

d) Automated Policy Rules Faults Detection

Our approach on rules fault detection verification is implemented as follows. A fault detection algorithm is designed for each fault pattern. These algorithms are executed sequentially on the context & adaptation data derived by simulated policy execution offline, or on real data extracted from the database and identify the specific fault category in each relevant test case, for troubleshooting. More details are provided in Appendix C.

APPENDIX D: The Context Usage Model

The usage model initial data over a one month period, on154 CSD sessions and 217 adaptations (and bindings to cloud services), are listed below for documentation purposes. Based on these data is the regression analysis of Table 8-1 and the monitoring scenarios described in §8.1.3.

Table D-1 – The Context Usage Model Values

usg Row No	session No	dateNow	usg StartTime	usg EndTime	userPref	bandwidth	battery Recharge	nonCloud battery Drop	CSD battery Drop
1	1	01/07/13	9:12:41	9:15:11	Normal	200	94	0	2
2	1	01/07/13	9:15:11	9:17:06	Normal	100		0	1
3	2	01/07/13	9:45:42	9:50:11	LowCost	250		0	2
4	3	01/07/13	13:23:32	13:26:02	HighQuality	180		2	2
5	3	01/07/13	13:26:02	13:27:07	HighQuality	90		0	1
6	4	01/07/13	17:55:11	17:57:11	Normal	320		2	2
7	4	01/07/13	17:57:11	17:59:01	Normal	160		0	1
8	4	01/07/13	17:59:01	18:00:57	Normal	384		0	1
9	5	01/07/13	19:14:43	19:17:13	Normal	120		0	1
10	5	01/07/13	19:17:13	19:18:01	Normal	60		0	0.5
11	6	02/07/13	9:24:11	9:28:04	LowCost	160		8	2
12	7	02/07/13	11:31:46	11:36:10	SaveBattery	260		1	3
13	8	02/07/13	13:46:04	13:50:57	Lowcost	150		1	3
14	9	02/07/13	17:34:16	17:36:46	Normal	200		2	2
15	9	02/07/13	17:36:46	17:38:12	Normal	100		0	1
16	10	02/07/13	18:53:14	18:54:44	Normal	300		0	1
17	10	02/07/13	18:54:44	18:57:06	Normal	150		0	1
18	11	03/07/13	8:54:11	8:58:55	Normal	150		8	3
19	12	03/07/13	11:44:56	11:50:06	LowCost	320		1	2
20	13	03/07/13	13:34:11	13:39:08	SaveBattery	280		1	2
21	14	03/07/13	18:11:13	18:17:44	LowCost	150		2	4
22	15	03/07/13	18:55:11	19:02:07	Normal	240		0	2
23	16	03/07/13	19:46:57	19:52:26	Normal	100		0	3
24	17	04/07/13	9:12:41	9:14:41	Normal	280	91	0	1
25	17	04/07/13	9:14:41	9:16:31	Normal	140		0	1
26	17	04/07/13	9:16:31	9:18:06	Normal	384		0	1
27	18	04/07/13	11:44:56	11:50:06	LowCost	320		1	2
28	19	04/07/13	13:23:20	13:25:50	HighQuality	80		0	2
29	19	04/07/13	13:25:50	13:28:10	HighQuality	50		0	1
30	19	04/07/13	13:28:10	13:30:12	HighQuality	160		0	2
31	20	04/07/13	16:44:37	16:49:11	LowCost	160		2	3
32	21	04/07/13	19:24:40	19:25:56	HighQuality	300		1	1
33	22	05/07/13	9:24:11	9:28:11	Normal	140		8	3
34	22	05/07/13	9:28:11	9:29:15	Normal	280		0	0.5

35	23	05/07/13	11:24:56	11:29:01	LowCost	340		1	2
36	24	05/07/13	13:44:17	13:46:47	Normal	100		1	2
37	24	05/07/13	13:46:47	13:47:16	Normal	50		0	0.5
38	25	05/07/13	18:11:21	18:15:51	Normal	200		2	3
39	25	05/07/13	18:15:51	18:16:44	Normal	384		0	0.5
40	26	05/07/13	19:45:36	19:51:08	LowCost	300		1	2
41	27	06/07/13	8:47:55	8:50:25	Normal	100		8	2
42	27	06/07/13	8:50:25	8:52:55	Normal	50		0	2
43	27	06/07/13	8:52:55	8:53:00	Normal	200		0	0.5
44	28	06/07/13	10:11:56	10:14:16	HighQuality	160		0	2
45	28	06/07/13	10:14:16	10:16:46	HighQuality	80		0	2
46	28	06/07/13	10:16:46	10:17:07	HighQuality	320		0	0.5
47	29	06/07/13	13:44:23	13:49:00	Normal	340		2	2
48	30	06/07/13	17:55:11	18:01:04	SaveBattery	330		2	2
49	31	06/07/13	19:24:44	19:30:56	SaveBattery	400		0	2
50	32	07/07/13	10:33:41	10:35:41	Normal	290	90	0	1
51	32	07/07/13	10:35:41	10:37:31	Normal	145		0	1
52	32	07/07/13	10:37:31	10:39:06	Normal	384		0	1
53	33	07/07/13	12:24:55	12:30:55	LowCost	300		1	3
54	34	07/07/13	17:44:11	17:46:41	Normal	200		3	2
55	34	07/07/13	17:46:41	17:48:03	Normal	100		0	1
56	35	07/07/13	19:35:26	19:37:56	Normal	100		1	2
57	35	07/07/13	19:37:56	19:40:06	Normal	50		0	1
58	35	07/07/13	19:40:06	19:40:33	Normal	200		0	0.5
59	36	07/07/13	20:10:07	20:15:46	LowCost	320		0	2
60	37	08/07/13	11:34:23	11:38:02	LowCost	220		9	2
61	38	08/07/13	12:44:21	12:46:21	Normal	80		0	1
62	38	08/07/13	12:46:21	12:47:05	Normal	50		0	0.5
63	39	08/07/13	13:24:36	13:27:06	Normal	110		0	2
64	39	08/07/13	13:27:06	13:28:11	Normal	55		0	1
65	40	08/07/13	19:10:21	19:12:11	HighQuality	400		3	1
66	40	08/07/13	19:12:11	19:14:11	HighQuality	200		0	1
67	40	08/07/13	19:14:11	19:17:55	HighQuality	384		0	3
68	41	09/07/13	9:11:16	9:12:46	Normal	200		8	1
69	41	09/07/13	9:12:46	9:16:21	Normal	200		0	2
70	42	09/07/13	11:15:10	11:20:01	LowCost	320		1	2
71	43	09/07/13	13:41:25	13:48:58	SaveBattery	280		1	3
72	44	09/07/13	18:11:23	18:17:34	SaveBattery	250		2	2
73	45	09/07/13	19:05:11	19:12:02	LowCost	400		0	2
74	46	10/07/13	8:55:11	9:03:06	Normal	150	91	0	4
75	47	10/07/13	10:11:44	10:14:14	Normal	100		0	2
76	47	10/07/13	10:14:14	10:16:24	Normal	50		0	1
77	47	10/07/13	10:16:24	10:17:33	Normal	200		0	1
78	48	10/07/13	13:31:45	13:32:45	LowCost	320		2	0.5
79	48	10/07/13	13:32:45	13:38:24	LowCost	320		0	2
80	49	10/07/13	18:05:23	18:10:01	Normal	340		2	2

81	50	10/07/13	19:14:44	19:17:45	LowCost	360		0	1
82	51	11/07/13	8:48:12	8:54:23	SaveBattery	150		8	3
83	52	11/07/13	9:23:11	9:28:01	LowCost	300		0	2
84	53	11/07/13	13:24:45	13:29:07	Normal	340		2	2
85	54	11/07/13	18:01:34	18:05:15	LowCost	400		2	1
86	55	11/07/13	19:11:44	19:17:02	LowCost	380		0	2
87	56	12/07/13	8:46:13	8:51:29	LowCost	180		8	2
88	57	12/07/13	9:11:15	9:17:11	LowCost	240		0	2
89	58	12/07/13	13:22:45	13:25:15	Normal	100		2	2
90	58	12/07/13	13:25:15	13:27:01	Normal	50		0	1
91	59	12/07/13	17:49:34	17:55:04	Normal	260		2	2
92	60	12/07/13	19:23:16	19:29:12	LowCost	240		0	2
93	61	13/07/13	8:40:21	8:45:12	Normal	200	92	0	3
94	62	13/07/13	9:11:19	9:15:23	HighQuality	240		0	3
95	63	13/07/13	13:02:11	13:07:00	Normal	340		2	2
96	64	13/07/13	18:02:43	18:06:55	LowCost	350		3	1
97	65	13/07/13	19:19:33	19:23:27	LowCost	240		0	1
98	66	14/07/13	8:49:23	8:53:04	LowCost	150		8	2
99	67	14/07/13	9:24:13	9:26:43	Normal	100		0	2
100	67	14/07/13	9:26:43	9:28:07	Normal	50		0	1
101	68	14/07/13	11:44:25	11:48:17	Normal	380		1	2
102	69	14/07/13	13:34:42	13:37:21	LowCost	180		1	1
103	70	14/07/13	17:33:24	17:37:06	Normal	380		2	2
104	71	14/07/13	19:12:58	19:18:01	Normal	160		1	3
105	72	15/07/13	9:45:11	9:47:11	Normal	280		9	1
106	72	15/07/13	9:47:11	9:49:01	Normal	140		0	1
107	72	15/07/13	9:49:01	9:50:28	Normal	384		0	1
108	73	15/07/13	11:45:52	11:47:52	HighQuality	320		1	2
109	73	15/07/13	11:47:52	11:49:52	HighQuality	160		0	2
110	73	15/07/13	11:49:52	11:51:03	HighQuality	384		0	1
111	74	15/07/13	12:44:17	12:49:02	Normal	200		0	3
112	75	15/07/13	17:55:23	18:02:49	LowCost	380		3	2
113	76	16/07/13	9:05:27	9:10:12	Normal	220	94	0	2
114	77	16/07/13	10:27:49	10:30:19	HighQuality	180		0	2
115	77	16/07/13	10:30:19	10:31:03	HighQuality	90		0	0.5
116	78	16/07/13	13:14:48	13:17:18	Normal	120		1	1
117	78	16/07/13	13:17:18	13:19:48	Normal	60		0	1
118	78	16/07/13	13:19:48	13:19:55	Normal	240		0	0.5
119	79	16/07/13	17:55:34	18:01:36	Normal	200		2	3
120	80	16/07/13	19:12:24	19:16:46	LowCost	240		0	2
121	81	17/07/13	9:11:15	9:15:49	Normal	140		8	3
122	82	17/07/13	10:31:56	10:33:46	HighQuality	280		0	1
123	82	17/07/13	10:33:46	10:34:48	HighQuality	140		0	1
124	83	17/07/13	13:12:36	13:17:10	Normal	340		1	2
125	84	17/07/13	18:06:44	18:08:44	Normal	120		3	1
126	84	17/07/13	18:08:44	18:10:37	Normal	60		0	1

127	85	17/07/13	19:11:42	19:15:38	LowCost	360		0	1
128	86	18/07/13	8:42:23	8:46:16	Normal	230		8	2
129	87	18/07/13	9:11:13	9:13:33	HighQuality	80		0	2
130	87	18/07/13	9:13:33	9:16:03	HighQuality	50		0	2
131	87	18/07/13	9:16:03	9:17:55	HighQuality	160		0	1
132	88	18/07/13	13:24:39	13:29:47	Normal	320		2	2
133	89	18/07/13	18:23:14	18:28:09	LowCost	340		3	2
134	90	18/07/13	19:11:15	19:17:40	Normal	240		0	2
135	91	19/07/13	8:47:23	8:50:06	LowCost	140	93	0	2
136	92	19/07/13	9:54:11	9:55:41	Normal	320		0	1
137	92	19/07/13	9:55:41	9:57:51	Normal	160		0	2
138	92	19/07/13	9:57:51	9:59:46	Normal	384		0	1
139	93	19/07/13	13:18:11	13:20:41	Normal	80		2	2
140	93	19/07/13	13:20:41	13:22:03	Normal	50		0	1
141	94	19/07/13	18:05:44	18:09:17	LowCost	220		3	1
142	95	19/07/13	19:11:26	19:16:03	SaveBattery	320		0	2
143	96	20/07/13	9:41:54	9:45:18	LowCost	200		9	1
144	97	20/07/13	10:35:41	10:39:57	Normal	260		0	2
145	98	20/07/13	11:58:12	12:01:56	LowCost	300		0	1
146	99	20/07/13	13:12:24	13:15:49	Normal	240		0	1
147	100	20/07/13	15:33:40	15:37:12	Normal	160		1	2
148	101	21/07/13	9:36:41	9:41:11	LowCost	200		11	2
149	102	21/07/13	11:15:22	11:16:52	Normal	300		1	1
150	102	21/07/13	11:16:52	11:19:02	Normal	150		0	1
151	102	21/07/13	11:19:02	11:21:28	Normal	384		0	1
152	103	21/07/13	13:11:21	13:18:54	LowCost	280		1	3
153	104	21/07/13	15:44:23	15:49:07	Normal	270		1	2
154	105	22/07/13	8:40:12	8:42:42	Normal	180	92	0	2
155	105	22/07/13	8:42:42	8:44:09	Normal	90		0	1
156	106	22/07/13	9:10:15	9:14:11	LowCost	260		0	2
157	107	22/07/13	13:28:11	13:31:07	Normal	240		2	2
158	108	22/07/13	18:11:46	18:14:55	Normal	260		2	2
159	109	22/07/13	19:07:49	19:10:52	HighQuality	220		0	2
160	110	23/07/13	8:46:13	8:50:58	Normal	150		8	3
161	111	23/07/13	9:11:55	9:14:46	LowCost	180		0	1
162	112	23/07/13	13:25:49	13:29:11	LowCost	220		2	1
163	113	23/07/13	18:20:11	18:24:14	Normal	360		3	3
164	114	23/07/13	19:18:44	19:22:16	Normal	180		0	2
165	115	24/07/13	8:51:23	8:57:08	LowCost	180		8	3
166	116	24/07/13	9:30:34	9:35:23	Normal	240		0	3
167	117	24/07/13	13:18:11	13:22:55	Normal	200		2	3
168	118	24/07/13	18:11:34	18:16:55	LowCost	230		3	2
169	119	24/07/13	19:11:43	19:17:36	LowCost	160		0	2
170	120	25/07/13	8:52:11	8:55:23	Normal	150	90	0	2
171	121	25/07/13	9:36:44	9:39:14	Normal	160		0	2
172	121	25/07/13	9:39:14	9:41:24	Normal	80		0	2

173	121	25/07/13	9:41:24	9:42:35	Normal	320		0	1
174	122	25/07/13	13:20:17	13:26:45	LowCost	200		2	2
175	123	25/07/13	18:07:56	18:11:38	Normal	220		2	2
176	124	25/07/13	19:11:45	19:15:37	LowCost	240		0	1
177	125	26/07/13	8:49:53	8:52:23	Normal	200		8	2
178	125	26/07/13	8:52:23	8:53:22	Normal	100		0	1
179	126	26/07/13	9:24:12	9:27:16	LowCost	180		0	2
180	127	26/07/13	13:11:49	13:15:03	Normal	360		2	2
181	128	26/07/13	18:12:39	18:16:12	Normal	380		3	1
182	129	26/07/13	19:11:44	19:15:04	LowCost	400		0	1
183	130	27/07/13	9:18:44	9:24:11	LowCost	200		8	3
184	131	27/07/13	10:35:22	10:37:22	Normal	340		0	1
185	131	27/07/13	10:37:22	10:39:12	Normal	170		0	1
186	131	27/07/13	10:39:12	10:41:06	Normal	384		0	1
187	132	27/07/13	12:45:17	12:50:06	LowCost	240		1	2
188	133	27/07/13	15:03:36	15:06:06	Normal	120		1	1
189	133	27/07/13	15:06:06	15:08:11	Normal	60		0	1
190	134	27/07/13	17:11:23	17:13:53	HighQuality	80		1	2
191	134	27/07/13	17:13:53	17:16:13	HighQuality	50		0	1
192	134	27/07/13	17:16:13	17:17:34	HighQuality	160		0	1
193	135	27/07/13	18:05:28	18:11:57	Normal	350		0	2
194	136	28/07/13	9:30:33	9:32:33	HighQuality	300	89	0	1
195	136	28/07/13	9:32:33	9:33:56	HighQuality	150		0	1
196	137	28/07/13	12:11:34	12:13:04	Normal	300		1	1
197	137	28/07/13	12:13:04	12:15:06	Normal	150		0	1
198	138	28/07/13	14:11:05	14:14:43	LowCost	290		1	2
199	139	28/07/13	15:33:45	15:37:01	LowCost	300		0	1
200	140	29/07/13	8:51:23	8:55:08	LowCost	210		10	2
201	141	29/07/13	9:30:34	9:33:04	Normal	180		0	2
202	141	29/07/13	9:33:04	9:34:23	Normal	90		0	1
203	142	29/07/13	13:18:11	13:22:55	Normal	340		2	2
204	143	29/07/13	18:11:34	18:15:55	LowCost	240		3	1
205	144	29/07/13	19:11:43	19:16:36	LowCost	120		0	2
206	145	30/07/13	8:50:34	8:56:12	Normal	200		8	2
207	146	30/07/13	9:11:23	9:17:55	LowCost	240		0	2
208	147	30/07/13	13:23:14	13:28:03	Normal	320		2	2
209	148	30/07/13	18:08:55	18:10:45	HighQuality	340		2	1
210	148	30/07/13	18:10:45	18:12:55	HighQuality	170		0	2
211	148	30/07/13	18:12:55	18:13:09	HighQuality	384		0	0.5
212	149	30/07/13	19:11:23	19:15:29	Normal	280		0	1
213	150	31/07/13	8:47:15	8:51:07	LowCost	100	90	0	2
214	151	31/07/13	9:14:45	9:17:03	Normal	200		0	1
215	152	31/07/13	13:20:17	13:24:11	LowCost	350		2	1
216	153	31/07/13	18:09:23	18:13:34	Normal	260		2	1
217	154	31/07/13	19:18:34	19:22:56	Normal	240		0	2

APPENDIX E: The Weight-Based Adaptation Mathematical Model

The Volare policy-based DCAA Model is expressed through the different conceptual components of the Policy Logic and the Context & Adaptation Profile for the middleware and an application. We abstract the values of the Boolean variables as 0 or 1 and the respective values of the discreet string type variables as integer values: 1, 2, 3 etc corresponding to the respective string values.

Referring to the scenario Context & Adaptation Profile, we have:

- The vector of the context-variables:

$CV = \{cv_1, \ldots cv_i, \ldots cv_m\}$, $1 =< i =< m$, i and $m \in N$ (1)

where: $cv_i \in \mathbf{CVi} \subset R$ and $CV \subset CV_1 \; X \; CV_2 \; X \; CV_m \forall$ i,m $\in N$, where CV is the context-space

- The vector of the adaptation-variables:

$AV = \{av_1, \ldots, av_j \; av_n\}$, j and $n \in N$ (2)

where: $av_j \in \mathbf{AVj} \subset R$ and $AV \subset AV_1 \; X \; AV_2 \; X \ldots AV_n$ $1 =< j =< n$ and j, $n \in N$

Policy File Representation

The scenario Adaptation Space Segmentation Model identifies:

- The vector of adaptation-strategies $\mathbf{S_X}$ of the policy file X (X = G or A)

$XP = \{S_{X1}, S_{X2}, \ldots, S_{Xs}\}$, s E N

(for example in the scenario, the number of A/S of the Global Policy i**s** is 8).

- The vector of MESCs represented in the policy file XP, $\mathbf{X} \in \mathbf{\{G, A\}}$, corresponding to the Global or the Application **A** policy file:

$MESC = \{MESC_{X1}, \ldots, MESC_{Xd}, MESC_{Xv}\}$, $1 =< d =< v$, v & d $\in$ (3a)

- The vector of adaptation-rules for each adaptation-strategy:

$S_{Xi} = \{AR_{Xi1}, AR_{Xi2}, \ldots, AR_{Xik}\}$, k E N, $1 =< i =< k$ (3b)

- Each adaptation-rule $\mathbf{AR_{Xi}}$ of the policy file **GP** or **AP**, represents a single row matrix of dimensions 1x5:

$AR_{Xi} = \{pr_{Xi}, AVid_{Xj}, ev_{Xi}, bm_{Xi}, w_{Xi}\}$, $1 =< i =< r$ i, $r \in N$ (4)

where **r** denotes the number of A/Rs in the policy file and $\mathbf{AVid_i}$ denotes the corresponding A/V (head predicate)

$\mathbf{pr_{Xi}}$ is an optional priority assigning keyword (“overrides”, “yields”, etc),

AVid$_{Xj}$ is the A/V id of the A/V concerning the **A/R$_{Xi}$,**

ev$_{Xi}$ is the execution-value assigned to adaptation-variable **AVid$_{Xj}$** by the A/R,

bm$_{Xi}$ is an optional value for the binding margin if the adaptation-variable is a QoS adaptation-variable and **w$_{Xi}$** is the obligatory weight function value for the A/R **AR$_{Xj}$** of the policy file **X**.

The DCAA relation is expressed as follows: AV = f(CV, GP, AP, UP) (5)

where: **CV** stands for the context-variables vector (except "User Preference"),

GP stands for the Global Policy adaptation-rules,

AP stands for the Application Policy adaptation-rules,

UP stands for the current User Preference Mode,

where UP = {"Normal", "LowCost", "SaveBattery", "HighestBitrate"} AV stands for the vector of the adaptation-variables.

Policy Execution

1. At policy execution, the Policy Files Manager creates the **Composite-Policy** composed of all the A/Rs of the **p** A/Ss of the Global and the **q** A/Ss of the Application Policy File, which may be represented by the **CP matrix** of **p+q X 5** dimensions with subscript GA indicating the G and A policy files:

   CP$_{GA}$ Matrix$_{p+qX5}$ = {PR$_{GA}$, AV$_{GA}$, EV$_{GA}$, BM$_{GA}$, W$_{GA}$}, with some elements or whole rows of elements equal to zero (6)

   where: PR, AV, EV, BM and W stand for vertical vectors of p+q elements, as follows:

   PR$_{GA}$ = {pr$_{GA1}$,… pr$_{GAj}$, …, pr$_{GAp+q}$} where 1 =< i =< p+q (6a)

   AVid$_{GA}$ = {AVid$_{GA1}$, …, AVid$_{GAi}$, … AVid$_{GAp+q}$} where AVid denotes each adaptation-variable

   id **(6b)**

   EV$_{GA}$ = {ev$_{GA1}$, …, ev$_{GAi}$, ,,, ev$_{GAp+q}$} where **ev$_{GAi}$** denotes the execution-value of the

   respective A/R **(6c)**

   BM$_{GA}$ = {bm$_{GA1}$, … , bm$_{GAi}$, … , bm$_{GAp+q}$} where **bm$_{GAi}$** denotes the binding margin if

   the respective adaptation-variable is a QoSVariable, otherwise it is 0.
   (6d)

W_{GA} = {w_{GA1}, … , w_{GAi}, … , w_{GAp+q}} where **w_{GAi}** denotes the weight value for the

respective A/R **i**. **(6e)**

2. The following ***Execution Data Matrix*** of dimensions **r X 4** of adaptation-rule elements is derived from the **r** selected under the current context and executed A/Rs, around the **m** adaptation-variables, after the ***Policy Engine Manager*** through the Conflict Resolution Directives has eliminated all the lower priority A/Rs:

$$\text{Execution Data Matrix r X 4} = \begin{pmatrix} \boldsymbol{AVid1} & \boldsymbol{ev1} & \boldsymbol{bm1} & \boldsymbol{w1} \\ \boldsymbol{AVid2} & \boldsymbol{ev2} & \boldsymbol{bm2} & \boldsymbol{w2} \\ \dots & \dots & \dots & \dots \\ \boldsymbol{AVidj} & \boldsymbol{evi} & \boldsymbol{bmi} & \boldsymbol{wi} \\ \dots & \dots & \dots & \dots \\ \boldsymbol{AVidr} & \boldsymbol{evr} & \boldsymbol{bmr} & \boldsymbol{wr} \end{pmatrix} \quad (6)$$

where: **AVid**$_j$ denotes the adaptation-variable AVid of the selected A/R **j** 1 =< **j** =< **r** and **j, r** ∈

N

3. The ***Policy Engine Manager*** restructures the ***Execution Data Matrix*** into **m** sub-matrices, one for each adaptation-variable, of dimensions **3Xk$_{QoS}$,** if the adaptation-variable is a QoSVariable of the service request where **k$_{QoS}$** is the number of selected A/Rs: 1 =< j =< **k$_{QoS}$** =< p+q

$$\text{AVid}_i \text{ Data Matrix } k_{QoS}\text{X3} = \begin{pmatrix} \boldsymbol{ev1} & \boldsymbol{bm1} & \boldsymbol{w1} \\ \boldsymbol{ev2} & \boldsymbol{bm2} & \boldsymbol{w2} \\ \dots & \dots & \dots \\ \boldsymbol{evj} & \boldsymbol{bmj} & \boldsymbol{wj} \\ \dots & \dots & \dots \\ \boldsymbol{ev}\mathbf{kQoS} & \boldsymbol{bm}\mathbf{kQoS} & \boldsymbol{w}\mathbf{kQoS} \end{pmatrix} \quad (7a)$$

For the each of the rest adaptation-variables that are not QoS Variables, a data matrix of dimensions **2 x p** is derived:

$$\text{AVid}_i \text{ Data Matrix p x 2} = \begin{pmatrix} \boldsymbol{ev1} & \boldsymbol{w1} \\ \boldsymbol{ev2} & \boldsymbol{w2} \\ \dots & \dots \\ \boldsymbol{evj} & \boldsymbol{wj} \\ \dots & \dots \\ \boldsymbol{ev}\mathbf{p} & \boldsymbol{w}\mathbf{p} \end{pmatrix} \quad (7b)$$

Consequently, at policy execution if there are **p adaptation-strategies** of the Global policy file, then there are (up to) **p execution-values** for each adaptation-variable (and especially for the QoSVariables, if there are also q adaptation-strategies in the Application policy file, then we have **p+q execution-values**).

Two vectors (or one matrix px2) of adaptation results are provided for each adaptation-variable on the execution-values **ev_i**, and the weight values **w_i (and concerning the QoSVariables** the binding margin values **bm_i)**:

- For every adaptation-variable **$AVid_i$** the Policy Engine Manager forms the following three execution data values vectors:

- $EV_i = [ev_{i1}, ev_{i2},\ldots, ev_{ij}, \ldots ,ev_{ik}]$ (8a)

- $W_i = [w_{i1}, w_{i2}, \ldots, w_{ij}, \ldots, w_{ik}]$ (8b)

- and if it is a QoSVariable, $BM_i = [bm_{i1}, bm_{i2},\ldots, bm_{ij}, \ldots ,bm_{ik}]$ (8c)

where i denotes the adaptation-variable **$AVid_i$** and 1 =< i =< k and i, k ∈ N

- For every numeric adaptation-variable, we have:

$\boldsymbol{avi} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{k}} (\boldsymbol{ev_j} * \boldsymbol{w_j})$ / $\sum_{\boldsymbol{j=1}}^{\boldsymbol{k}} \boldsymbol{wi}$ ***where*** j = 1 to k, with j and k ∈ N (9a)

1 =< j =< k with j, k ∈ N

- Additionally, for every numeric QoS adaptation-variable, concerning the binding margin we have: $\boldsymbol{bmi} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} (\boldsymbol{bm_i} * \boldsymbol{w_i})$ / $\sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} \boldsymbol{wi}$, i = 1 to k, with I, k ∈ N (9b)

- However, for every non-numeric (Boolean or string type) adaptation-variable, and any variant values ***var_1, var_2,…, var_k*** we have the sums of weight values over each variant value:

$\sum wij)_{\text{over var1}}$, $\sum wij)_{\text{over var2}}$, …, $\sum wij)_{\text{over vark}}$

The maximum sum of weight values over a variant, specifies the resolved-value for this Boolean or string-type adaptation-variable, i.e. **max {$\sum w_{ij})_{\text{over var1}}$, $\sum w_{ij})_{\text{over var2}}$, …, $\sum w^{ij})_{\text{over vark}}$}** defines the variant Boolean or string-type resolved-value, i.e. the selected variant value.

Weight Assigning Strategy

The vector of weight function values for each adaptation-rule **ARid** of an adaptation-variable **AVj** (1 =< **j** =< n, with j, n ∈ N), from the (Global or Application) policy file **XP**:

W_{Xrj} = {w_{xr1}, w_{xr2}, ..., w_{xrn}}, where in the scenario X = G or A (for the application A).

The weight function **w_{ARid}** for the A/R: **ARid** is a function of the following variables:

w_{Arid} = g(Sid, UP, CV, ARid) (10)

where UP and CV are defined above and

Sid denotes the Adaptation-strategy to which the A/R belongs,

ARid denotes the A/R represented uniquely by the relevant adaptation-variable and its Criteria conditions,

AVid denotes the adaptation-variable assigned value by the A/R.

The Weight Assigning Strategy of the Weight-based Methodology defines three weight coefficients:

a) The Adaptation-strategyategy weight coefficient under each User Preference Mode, denoted as ws, with $wSid_{UP} = f(Sid, UP)$

(11a)

b) The A/R Adaptation Rule weight coefficient within the A/S, denoted as wAR, with

$wARid = h(Sid, AVid)$ (12)

in the sense that within a Policy, two A/Rs on the same adaptation-variable have possible different relative importance due to the Adaptation-strategy they belong to and the Criteria Conditions they represent.

c) The Strategy Weight Coefficient Modifier under each User Preference mode, denoted as **uSid,** representing a value typically between 0.80 – 1.25, that the User may set at the User Choices Profile UCP through the User Interface (UI), customizing the default Strategy Weight Coefficients values:

The default value for each uSid is: **uSid = 1.00** **(13)**

d) The weight function for each A/R ARid is given by the product of the three weight coefficients:

$weight_{ARid} = wSid)_{UP} * uSid)_{UP} * wARid$ (14)

since the term $wSid)_{UP}$ * $uSid)_{UP}$ is provided as a context-variable by the User

Interface, denoted as **$wSid)_{correlated}$,** equation (14) becomes equivalent to the

equation: $weight_{ARid} = wSid)_{correlated} * wARid$ (14a)

with **$wSid)_{correlated}$** provided as context-variable value.