Adaptive and Context-Aware Service Composition for IoT-based Smart Cities

A. Urbieta[a,*], A. González-Beltrán[b], S. Ben Mokhtar[c], M. Anwar Hossain[d], L. Capra[e]

[a] *IK4-Ikerlan Technology Research Centre, Information and Communication Technologies Area, P° J.M.Arizmendiarrieta 2, 20500 Arrasate-Mondragón, Spain*
[b] *University of Oxford, Oxford e-Research Centre, 7 Keble Road, Oxford, OX1 3QG, United Kingdom*
[c] *INSA de Lyon, LIRIS, DRIM Research Group, 20 Avenue Albert Einstein, 69621 Villeurbanne, France*
[d] *King Saud University, College of Computer and Information Sciences, Department of Software Engineering, Riyadh 11543, KSA*
[e] *University College London, Department of Computer Science, Gower Street, London WC1E 6BT, United Kingdom*

Abstract

Smart Cities are advancing toward an instrumented, integrated, and intelligent living space, where Internet of Things (IoT), mobile technologies and next generation networks are expected to play a key role. In smart cities, numerous IoT-based services are likely to be available and a key challenge is to allow mobile users perform their daily tasks dynamically, by integrating the services available in their vicinity. Semantic Service Oriented Architectures (SSOA) abstract the environment's services and their functionalities as Semantic Web Services (SWS). However, existing service composition approaches based on SSOA do not support dynamic reasoning on user tasks and service behaviours to deal with the heterogeneity of IoT domains. In this paper, we present an adaptive service composition framework that supports such dynamic reasoning. The framework is based on *wEASEL*, an abstract service model representing services and user tasks in terms of their signature, specification (i.e., context-aware pre-conditions, post-conditions and effects) and conversation (i.e., behaviour with related data-flow and context-flow constraints). To evaluate our composition framework, we develop a novel OWLS-TC4-based testbed by combining simple and composite services. The evaluation shows that our wEASEL-based system performs more accurate composition and allows end-users to discover and investigate more composition opportunities than other approaches.

Keywords: Internet of Things; Smart Environments; Smart Cities; Smart Services; Real-Time and Semantic Web Services; System Design; and Service Modeling

1. Introduction

The smart cities [1, 2] are gradually advancing toward reality, thanks to the advancement of IoT (interconnected RFID, GPS, IR, camera, laser scanners, etc.), mobile technologies and next generation networks (e.g. emerging 5G). In smart cities, various IoT devices and associated services for location, intelligent recognition, tracking, monitoring and management are becoming available for the citizens to consume. One of the challenges of these kind of environments is to allow mobile users to seamlessly consume and often combine functionalities offered by software and hardware resources anywhere and at anytime. In the following, we provide a scenario to better exemplify the context of services, combination of services, and service consumption with respect to a smart city resident.

Carla is taking a long haul flight to Australia, where she has to attend an important seminar. For such a trip, Carla nowadays carries a single smart phone called wEASEL-Com, which is capable of connecting to many different devices and services and coordinate them on a centralized way. Today, exceptionally, Carla arrives early at the airport and when she enters the waiting lounge, it is almost empty. She decides to watch a movie while waiting for the boarding announcement and have a massage in a massage chair. Her wEASEL-Com device uses the wEASEL-Film application, one of the various embedded applications on Carla's tiny smart phone, which is able to discover video services for browsing the content of available

*Corresponding author
Email addresses: aurbieta@ikerlan.es (A. Urbieta), alejandra.gonzalezbeltran@oerc.ox.ac.uk (A. González-Beltrán), sonia.ben-mokhtar@liris.cnrs.fr (S. Ben Mokhtar), mahossain@ksu.edu.sa (M. Anwar Hossain), l.capra@cs.ucl.ac.uk (L. Capra)

video servers. Carla's device is also able to discover services that can select the most appropriate display devices in her surrounding (e.g., the one having the largest display). Furthermore, wEASEL-Film is able to adapt the surrounding environment according to Carla's preferences (e.g., room lighting, film sound level). Hence, wEASEL-Film starts displaying the film selected by Carla (based on a recommended list) on a large LED display that was disseminating the news. Half an hour later, wEASEL-Com informs Carla that she has to go for boarding. After boarding on the plane and paying attention to the security demonstration, Carla is asked by wEASEL-Film whether she would like to continue watching the film on the personal LED panel mounted on the back of the seat in front of her.

Engineering a software system as above involves dealing with challenges peculiar to IoT-based smart environments and cities, namely heterogeneity and dynamics. Indeed, heterogeneous technologies and languages are used by hardware and software resources and the dynamics stem from users' mobility, which implies that the resources and functionalities available may change overtime. In order to develop pervasive software and hardware systems for smart city residents that seamlessly integrate the environment's heterogeneous functionalities and that adapt to their dynamics, it is required to model each resource as an autonomous component, the environment's functionalities as functions of time and to reason about their *context*.

Resources are modelled as services. A service is an autonomous and loosely coupled unit of functionality. Service Oriented Architecture (SOA) is an architectural style that offers a set of design principles and abstractions for the integration of independent services. The SOA style can be implemented using Web Service (WS) standards and specifications, or other service-based technologies. Furthermore, Semantic Service Oriented Architecture (SSOA) or Ontology-Enabled Service Oriented Architecture (OSOA) combines WS standards with the expressive power and the formal ground of Semantic Web (SW) technologies, such as the Web Ontology Language (OWL[1]). An ontology is a formal representation of the concepts and relationships within a domain. Thus, SSOA allows tackling the semantic heterogeneity of service descriptions by relying on established ontology specifications.

In the context of SSOA, a number of semantic service description and composition languages have been proposed (e.g. OWL-S[2], WSMO[3], SWSF[4], SAWSDL[5]). Most of these languages allow to specify services as a set of provided *capabilities*, which are the set of functionalities *provided* by the services of the smart environment or *required* for the realisation of the user tasks[6]. A capability has a *signature* (i.e., the set of inputs consumed and outputs produced by the capability) and a behavioural *specification* (required pre- and post-conditions, and generated effects). *Pre-conditions* are assertions that must be satisfied before a capability can be invoked. An *effect* refers to the result of invoking a capability, where *post-conditions* are the conditions that determine which *effect* is achieved. *Simple capabilities* identify a single functionality. Service capabilities can also be orchestrated in a *conversation*, which determines the order in which the capabilities are executed, to provide *composite capabilities*.

However, current composition approaches that can be applied to smart city environment have a number of limitations:

1. For matching service conversations, most of the existing solutions assume that either the service request or the service advertisement do not have an associated conversation, thus leading to simple capability aggregation mechanism.

2. Description-based service composition algorithms are mainly based on just inputs and outputs, without considering pre-conditions, post-conditions and effects. Consequently, it is not possible to deploy context-aware services. Additionally, they use conversation-driven service selection, where the pervasive services are described by means of simple-capability services.

In this paper, we contribute to present a composition framework based on wEASEL (which stands for contExt Aware web Service dEscription Language), an abstract service model for pervasive software systems that we introduced in [3]. wEASEL supports the specification of services in terms of their semantic signature, context-aware behavioural specification and conversation. As part of our wEASEL-based framework, in

[1]OWL: `http://www.w3.org/2004/OWL/`

[2]Semantic Markup for Web Services (OWL-S): `http://www.w3.org/Submission/OWL-S/`

[3]Web Service Modeling Ontology (WSMO): `http://www.w3.org/Submission/WSMO/`

[4]Semantic Web Services Framework (SWSF): `http://www.w3.org/Submission/SWSF-SWSO/`

[5]Semantic Annotations for WSDL (SAWSDL): `http://www.w3.org/2002/ws/sawsdl/`

[6]Web Service Architecture: `http://www.w3.org/TR/ws-arch/`

[3] we developed a set of signature and context-aware specification-based matching algorithms. However, while signature and context-aware specification matching holds between two services, it still demands a way to match a required service conversation by the integration of a set of provided service's conversations. Therefore, as a natural subsequent step within the wEASEL framework, here we introduce a conversation-based service composition framework that features three levels of composition flexibility:

1. *Flexible conversation integration*, where fragments of the provided services' conversations are integrated towards the realisation of the user task conversation.

2. *Flexible conversation interleaving*, which supports the interleaving of the provided services' conversations by fulfilling the data and control constraints of each individual service.

3. *Adaptive task conversation reshuffling* allows the adaptation of the requested service conversation by reshuffling capabilities. If the capabilities do not have any data or context dependencies among them, reshuffling can occur without restrictions. On the other hand, if there are data or context dependencies among the capabilities, these dependencies constitute constraints to the adaptation process. The adaptive composition algorithm increases the chance of finding service compositions.

We evaluate our composition framework by developing a testbed of simple and composite services. To the best of our knowledge, this is the first available testbed including composite services that has been created based on OWLS-TC4. When evaluating the trade-off between performance and correctness, we select the most appropriate variant of the composition framework to be applied in smart environments.

The remainder of this paper is structured as follows: Section 2 presents a categorisation of related research efforts for the dynamic realisation of user tasks in smart environments. Then, we describe our service composition framework in Section 3, before evaluating our overall framework both theoretically and practically in Section 4. Finally, our concluding remarks and future work are discussed in Section 5.

2. Related Work

The description-based service composition solutions can be classified into two main categories, depending on whether they are based on the service interfaces or on the service conversations. *Interface-based service composition* assumes that services are described with a list of independent capabilities, without associated conversations. On the other hand, *conversation-based service composition* assumes that services have associated conversations. In both categories, user tasks may be specified with or without an associated conversation. In the following sections, we describe both categories.

2.1. Interface-Based Service Composition

Interface-based service composition (see the second column of Figure 1) is implemented as *service chaining algorithms* or *conversation-driven service selection algorithms*, according to whether the task is specified with or without an associated conversation, respectively.

In service chaining (when both networked services and the target user task have no conversations) [4, 5], including *forward* and *backward chaining*, individual service capabilities are combined with each other based on the conformance of their signatures. The objective is to obtain a composite service that conforms to the signature specification of the user task. Forward chaining starts by selecting services that match the task's provided inputs (and pre-conditions) and chains services forward based on their signature compatibility until all the task's required outputs (and effects) are generated. On the other hand, backward chaining starts by selecting services that generate the task's required outputs (and effects) and chains services backward until all the inputs (and pre-conditions) of the selected services can be satisfied by the task's provided inputs (and pre-conditions). While chaining allows to combine services without any previous knowledge about how services should be chained, its complexity is high as all the possible combinations need to be investigated. Furthermore, as the chaining process is "blind" (i.e., capabilities are chained only on the basis of the compatibility of their signature and/or specification), unexpected capabilities may be employed, which generates uncertainty regarding how user's information is manipulated. Some approaches improve this by providing task decomposition rules in order to orient the service chaining process [6]. Most approaches use a single type of chaining, however Yu and Reiff-Marganiec [7] combine both chaining types, first using forward chaining to determine which are the candidate services for each step and then backward chaining for each step to select the better service that fulfils the defined context criteria.

Conversation-driven service selection assumes user tasks are described with an associated conversation and services described as independent capabilities. This

model has been often employed for dynamic service composition in smart environments [8, 9, 10, 11, 12, 13]. Provided service capabilities are matched against capabilities required in the target user task. The various approaches vary according to the expressiveness of the service description language and its associated matching algorithm. As this approach follows a user task conversation specification, the resulting composition meets the user's requirements without unobtrusively using user provided information through the employment of unexpected capabilities. However, this model does not consider the behaviour of services when integrating them, which does not guarantee their correct composition.

2.2. Conversation-Based Service Composition

Conversation-based service composition assumes that services to be combined have a complex behaviour (see the third column of Figure 1). It is divided in three different cases, namely: *goal-driven conversation selection*, *goal-driven conversation integration* and *conversation-driven conversation integration*. On the former two the user task is specified without an associated conversation and on the latter one both services and tasks are specified with associated conversations).

Goal-driven conversation selection allows the selection of a service conversation that satisfies a user task specified as a single required capability [14]. A process query language, i.e. PQL, is employed to find service conversations that contain a fragment that satisfies the user task. Thus, it is implicitly assumed that the user's request can be performed by a single service as opposed to integrating multiple service conversations. More recently, Kiefer and Berstein [15] proposed to use SPARQL to represent the goal, using similarity measure techniques combined with machine learning techniques.

On the other hand, *goal-driven conversation integration* [16] aims at integrating a set of service conversations to realise a user task described as a single required capability. The conversations of a set of pre-selected services are integrated so that the resulting composition satisfies some properties (e.g. deadlock freedom) and conforms to the target user task by consuming all its provided inputs and generating all its required outputs. The approach by Sirbu *et al.* [17, 18] proposes to use pervasive process fragments, which represent a service-based tool that allows to model incomplete and contextual knowledge. In this way, encoding process knowledge, domain knowledge and goals into an Artificial Intelligence (AI) planning problem, the system is able to automatically compose such fragments into complete processes, according to a specific context and specific goals.

As in chaining algorithms, the last two composition models generate a degree of uncertainty regarding the way networked services are combined. Indeed, verifying that the resulting service composition is deadlock-free does not guarantee that the user's information has not been transformed using unexpected and inappropriate capabilities (e.g., capabilities that a user would not have employed themselves to achieve their objective) just in order to meet the target user task's input/output specification.

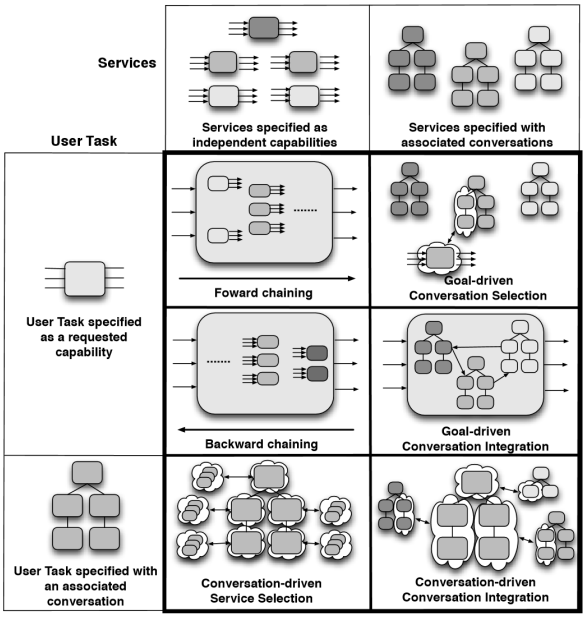


Figure 1: Composition Models

The final composition model, i.e. *conversation-driven conversation integration*, assumes a complex behaviour for both user tasks and services. Conversations of networked services are integrated towards the realisation of the user task's conversation. This composition model is the most comprehensive as it supports maximum expressiveness for task and service functional descriptions. The benefits of this composition model are: 1. The user task's behaviour is used as a basis for service composition, which ensures that the user requirements are satisfied by construction. 2. A valid consumption of the composed services is ensured as their conversations are fulfilled.

Further, as shown in Figure 2, *conversation-driven service integration* allows reconstructing the user task conversation using different composition schemes. In

this figure, a user task, depicted in the middle of the figure, is composed of four different smart environments using four different scenarios. In the first scenario, the task is realised through the integration of individual capabilities of pervasive services. The second scenario describes the case where a single service that conforms to the user task conversation is selected. The third scenario represents the case where the user task is realised through the composition of fragments from two service conversations. The last composition scheme is the most flexible where the realisation of the user task is performed through the interleaving of two service conversations.

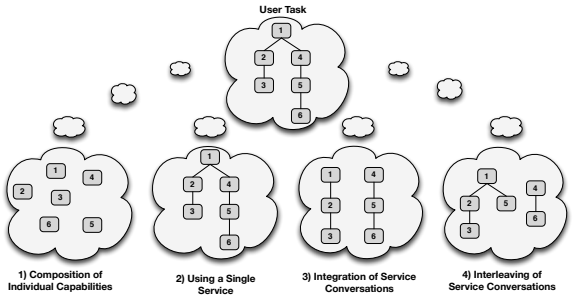


Figure 2: Flexibility of the Conversation-Driven Conversation Integration

Conversation-driven service integration is first investigated in [19], where service conversations are represented as finite-state automata. In this approach, the authors propose an exponential-time algorithm that searches for a possible service composition by reducing this problem to the satisfiability of a Deterministic Propositional Dynamic Logic (DPDL) formula. However, this solution considers neither service semantic specifications nor service and task non-functional properties. Furthermore, this algorithm employs costly formal verification algorithms, the efficiency of which is not assessed for resource-constrained devices. Ben Mokhtar *et al.* [20] also investigate this model in the COnversation-based Service COmposition in PervAsive Computing Environments with QoS Support (COCOA) system: the service composition is carried out by means of automata simulation based on IO-based (Input- and Output-based) semantic service descriptions, where the system offers a high grade of scalability for resource constrained devices. The approach proposed by Benatallah *et al.* [21] also uses automata simulation to find compatibility and replaceability properties between pairs of compositions, however this approach does not use semantics nor is oriented to integrate several compositions into one. A similar work to the one proposed by Ben Mokhtar *et al.* is defined by Hashemian and Mavaddat [22], where a graph-based approach is used to compose OWL-S process models, modeling both services and the client query as interface automata. Finally, Mancioppi *et al.* [23] propose an approach that uses Deterministic Finite Automata (DFA) enriched with time conditions on the transitions, which that in contrast to the approach of Benatallah *et al.* is a multi-party approach. However this approach does not support semantics nor contextual information.

2.3. Discussion

Considering the previous classification of service composition approaches and the survey of the main service composition approaches for smart environments carried out in a previous work [24] and extended by Stavropoulos *et al.* in [25], we conclude that:

- Most of the approaches are purely IO-based without considering contextual information (using PEs: Preconditions and Effects). Thus, it is not possible to deploy context-aware service composition: using just IOs the problem of composition is reduced to a problem of IOs chaining and not a problem that involves contextual changes that can be chained to create complex behaviours, as it is required in smart environments.

- Most of the approaches are based on conversation-driven service selection, where the services are described by means of simple-capability services. We consider that both, user task and pervasive services have to be described by means of conversations. This allows to fulfil the requested contextual requirements taking into account global and local behaviours of each of the capabilities of the conversation. In this way, the composition will be closer to what the user task needs. Therefore, the approach that better fits the smart environments requirements is the conversation-driven conversation integration approach.

Hence, the quest is still open for a solution to service composition that supports the integration of service conversations to realise the conversation of a user task. This functionality should support the semantic specification (PEs) and semantic signature (IOs) of service and task capabilities, and provide the means to adapt its flexibility according to the required efficiency with respect to the resources of thin devices.

3. Dynamic Service Composition

The dynamic realisation of user tasks may involve many networked services from the environment. The service composition process involves selection and coordination (not the composition execution process, i.e., the user task realization execution derived from the composition process is out of the scope of the present manuscript). Given a user task description, the service composition client hosted on the user's mobile device discovers the available services and filters out those that will not be useful for the target user task realisation (Section 3.1). Then, the selected services are coordinated to find possible compositions of the user task. Service coordination can be done using three different flexibility levels (Section 3.2) that differ in the overhead they incur in users' mobile devices.

3.1. Service Selection

The selection process chooses services whose capabilities conform to the user task's capabilities, in terms of signature and specification. More formally, according to our wEASEL model [3], consider a user task T described as a composite capability with an associated automaton $A_T =< Q_T, \Sigma_T, \delta_T, st_{0_T}, F_T >$ where Q_T is a set of states, including an initial state st_{0_T} and F_T a set of final states, δ_T is the transition function and Σ_T is the set of symbols, which contains the capabilities that constitute the task's conversation. These capabilities, called *required* capabilities, are assumed to be simple (not composite). Figure 3 shows the conversation of the wEASEL-Movie User Task, inspired from the scenario introduced in Section 1[7]. The data-flow constraints express the data dependency between two capabilities where an output produced by a capability is used as one of the inputs consumed by another capability.

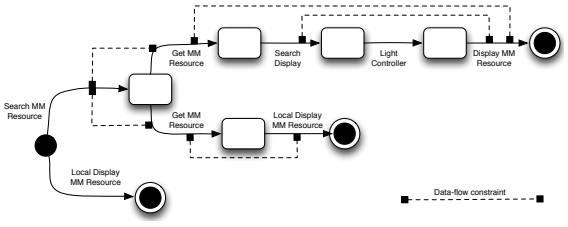


Figure 3: User Task Conversation

Figure 4 depicts the conversations of the pervasive services of the environment, where the context-flow constraints express the dependency between two capabilities where one of the effects produced by the execution of a capability validates one of the pre-conditions or post- conditions necessary to execute another capability.

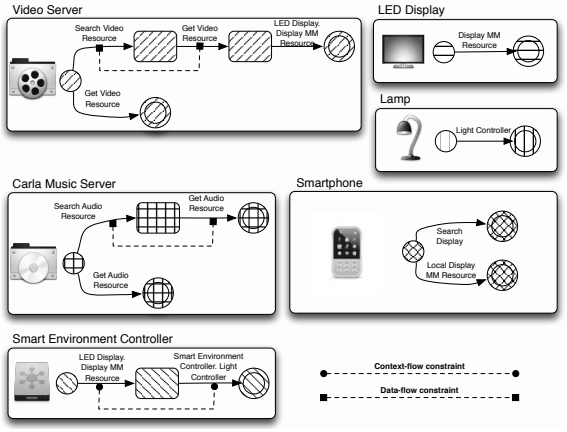


Figure 4: Pervasive Services' Conversations

The service selection returns a list of services to the client application, where each service offers at least one capability semantically conforming to a capability of the user task. More formally, the service selection process can be modelled with the following function:

$$\begin{aligned} ServiceSelection : \mathcal{T} &\longrightarrow 2^{\mathcal{S}} \\ T \in \mathcal{T} &\longmapsto \{s_1, ..., s_I\} \end{aligned} \tag{1}$$

such that:

$$\begin{aligned} &\forall s_i \in \{s_1, ..., s_I\} : \exists c_T \in \Sigma_T, \exists c_{s_i} \in \Sigma_{s_i} : \\ &SigMatch(Sig_{s_i}, Sig_T) \wedge \\ &SpeMatch(Spe_{s_i}, Spe_T) \end{aligned} \tag{2}$$

where $\mathcal{T}$ is a set of tasks, $\mathcal{S}$ is a set of pervasive services defined as $s_1, ..., s_I$, and the pair of relations **SigMatch**(Sig_1, Sig_2) (signature-based matching), **SpecMatch**(Spe_1, Spe_2) (specification-based matching) are defined in [3], where a thorough description of the different matching types is given.

Considering the user task from Figure 3, the **ServiceSelection**(T) function selects all the services in Figure 4, as they offer at least one capability semantically conforming to one of the user task required capabilities (note that if there are several semantically equivalent

[7]For readability, the transitions in the conversations are labelled with capability names.

services in the environment, all of them are selected as candidates). Table 1 shows those services that have been selected during the service selection process.

Table 1: Matching Between Capabilities of the wEASEL-Movie User Task and Selected Pervasive Services

Task Capability	Service Capability	Service
Search MM Resource	Search Video Resource	Video Server
	Search Audio Resource	Carla Music Server
Search Display	Search Display	Smartphone
Get MM Resource	Get Video Resource	Video Server
	Get Audio Resource	Carla Music Server
Local Display MM Resource	Local Display MM Resource	Smartphone
Display MM Resource	Display MM Resource	LED Display
Light Controller	Light Controller	Smart Environment Controller
	Light Controller	Lamp

3.2. Service Coordination

Service coordination is responsible for generating a set of concrete realisations of the user task. Each of these realisations refers to pervasive services' capabilities. We present three composition flexibility levels for the dynamic realisation of user tasks:

- Integrating service conversations in Section 3.2.3: the selected services are integrated without the support of conversation interleaving.
- Conversation interleaving in Section 3.2.4: allows integrating service conversations by enabling the interleaving of their conversations.
- Adaptive user tasks in Section 3.2.5: adjusts the user task's conversation according to its data-flow and context-flow specification to further increase the probability of finding a composition. Besides, this solution can be combined with the previous solutions to obtain a higher number of user task realisations.

By distinguishing between these three alternatives, we can provide the user with the most appropriate solution with respect to the available computing resources available on his/her device. In a resource rich environment, the most flexible solution, which increases the probability of finding a user task realisation, would be employed. However, a less flexible solution would be used in a resource constrained environment. Next, we formally define the problem of task realisation, and then detail each of the three flexibility levels.

3.2.1. Problem Definition

Let A_T be the automaton describing the conversation of the user task T. The set of candidate services for the composition of T are given by the **ServiceSelection**(T) function (Equation 1). The service coordination functionality aims at finding a ranked list of concrete realisations of the user task: $T_1, ..., T_J$ such that $\forall A_{T_j} =< Q_{T_j}, \Sigma_{T_j}, \delta_{T_j}, st0_{T_j}, F_{T_j} >$ associated with T_j:

$$
\begin{aligned}
&- Q_{T_j} = Q_T \\
&- \Sigma_{T_j} \subset \cup \Sigma_{s_i} \ \forall s_i \in ServiceSelection(T) \\
&- \delta_{T_j} = \delta_T \\
&- F_{T_j} = F_T \\
&- ConvDoM(A_T, A_{T_j}) < ConvDoM(A_T, A_{T_{j+1}})
\end{aligned}
\tag{3}
$$

where s_i represents a concrete pervasive service of the set $\mathcal{S} = s_1, ..., s_I$. Furthermore, the concrete realisations of the user task are ranked according to their degree of matching with the initial task using the **ConvDoM**(A_1, A_2) function defined in the following section.

3.2.2. Conversation Matching of Service Capabilities

The matching of service conversations is done using automata compatibility checking algorithms. Specifically, we define the relation **ConvMatch**(A_1, A_2) to compare two service conversations specified using finite state automata. Let $A_1 =< Q_1, \Sigma_1, \delta_1, st0_1, F_1 >$, $A_2 =< Q_2, \Sigma_2, \delta_2, st0_2, F_2 >$ be two automata, **ConvMatch**(A_1, A_2) is defined as:

$$
\begin{aligned}
&\exists\, R \text{ on } Q_1 \times Q_2 \text{ such that :} \\
&\forall < st_1, st_2 > \in R, \forall c_1 \in \Sigma_1 : \\
&\delta_1(st_1, c_1) = st_1' \Rightarrow \exists c_2 \in \Sigma_2 : \\
&SigMatch(Sig_1, Sig_2) \wedge \\
&SpeMatch(Spe_1, Spe_2) \wedge \\
&\delta_2(st_2, c_2) = st_2' \wedge < st_1', st_2' > \in R
\end{aligned}
\tag{4}
$$

where R is a binary relation defined over the set $Q_1 \times Q_2$. R is called *automata simulation*, and it is said that A_2 *simulates* A_1 or that A_1 is *simulated by* A_2.

The **ConvDoM**(A_1, A_2) function allows the evaluation of the degree of match between two conforming conversations. It is defined as the sum of the degree of match of each pair of capabilities that match from the first and the second conversation divided by the number of required capabilities. More formally, if we consider a required conversation A_1 and a provided conversation A_2 associated with the sets of capabilities Σ_{A_1} and Σ_{A_2} are ordered

in the form: $\Sigma_{A_1} = \{c_{1_{A_1}}, ..., c_{P_{A_1}}\}$, $\Sigma_{A_2} = \{c_{1_{A_2}}, ..., c_{P_{A_2}}\}$ such that $\forall p \in \{1, ..., P\}$: **SigMatch**$(Sig_{p_{A_1}}, Sig_{p_{A_2}})$ and **SpeMatch**$(Spe_{p_{A_1}}, Spe_{p_{A_2}})$. The **ConvDoM**$(A_1, A_2)$ function is defined as:

$$\frac{\sum_{p=1}^{P} CapabilityDoM(c_{p_{A_1}}, c_{p_{A_2}})}{P} \tag{5}$$

where $P=\left|\Sigma_{A_1}\right|$ is defined as the number of required capabilities of automaton A_1, $\prod_{p=1}^{p=P} CapabilityDoM(c_{pA_1}, c_{pA_2}) > 0$ and **CapabilityDoM**(A_1, A_2) (which is defined in [3]) is the function that returns the degree of match between two service capabilities c_1 and c_2.

3.2.3. Integrating Service Conversations

When integrating service conversations, we first build an automaton that combines the automata of the selected services. The resulting decidable automaton is called *raw automaton* and denoted as RA_{IG}. RA_{IG} contains a new initial state and empty transitions (ϵ) connecting it with the initial states of all selected automata. RA_{IG} also contains empty transitions connecting the final states of each automaton with the new initial state. This allows, if needed, the integration of the same service multiple times in the composition.

More formally, consider a set of services $s_1, ..., s_K$ selected by the **ServiceSelection**(T) and their associated service conversations $A_1, ..., A_K$, where $A_k =< Q_k, \Sigma_k, \delta_k, st0_k, F_k >$. Thus, the *raw automaton* $RA_{IG} =< Q_{RA}, \Sigma_{RA}, \delta_{RA}, st0_{RA}, F_{RA} >$ is generated by the function $\textbf{RawAutomaton}_{IG}(s_1, ..., s_K)$ defined as:

$$\begin{aligned}
&- Q_{RA} = \cup_{k=1}^{K} Q_k \cup st0_{RA} \\
&- \Sigma_{RA} = \cup_{k=1}^{K} \Sigma_k \cup \{\epsilon\} \\
&- \delta_{RA} : Q_{RA} \times \Sigma_{RA} \rightarrow Q_{RA} \\
&\qquad\qquad st, c \mapsto \delta_{RA}(st, c) \\
&- F_{RA} = \cup_{k=1}^{K} F_k
\end{aligned} \tag{6}$$

such that:

$$\delta_{RA}(st, c) = \begin{cases} \delta_k(st, c) & st \in Q_k \ \wedge \ c \in \Sigma_k \\ st0_k & st = st0_{RA} \wedge c = \epsilon \\ st0_{RA} & st \in F_{RA} \wedge c = \epsilon \end{cases} \tag{7}$$

Figure 5 presents the RA_{IG} built from the services selected according to the user task of Figure 3. Based on RA_{IG}, service coordination uses the relations **ConvMatch**(A_1, A_2) defined in Section 3.2.2 to find user task realisations. Specifically, there exists a realisation of the user task if RA_{IG} simulates the task automaton A_T. More formally, this can be represented by the function **ConvInteg**$(T, s_1, ..., s_K)$ as follows:

$$\begin{aligned}
&ConvInteg : \mathcal{T} \times \mathcal{S}^K \longrightarrow 2^{\mathcal{T}} \\
&\qquad < T, s_1, ..., s_K > \longmapsto \{T_1, ..., T_J\}
\end{aligned} \tag{8}$$

such that:

$$\begin{aligned}
&RA_{IG} = RawAutomaton_{IG}(s_1, ..., s_K) \wedge \\
&\forall T_j \in \mathcal{T}, j = 1, ..., J \ \exists A_{T_j} : \\
&SubAutomaton(RA_{IG}, A_{T_j}), \\
&ConvMatch(A_T, A_{T_j})
\end{aligned} \tag{9}$$

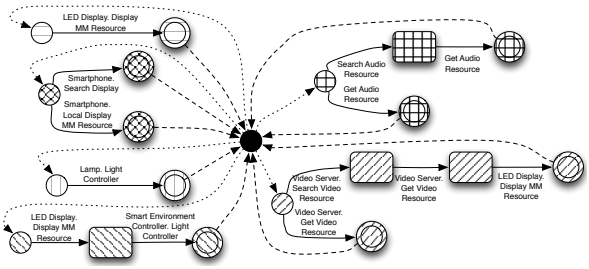


Figure 5: Raw automaton for Conversation Integration

where **SubAutomaton**(A_1, A_2) is a function that defines if an automaton A_2 is a subautomaton of an automaton A_1, which is defined as follows. Let A_1 be described as $A_1 =< Q_1, \Sigma_1, \delta_1, st0_1, F_1 >$ and A_2 as $A_2 =< Q_2, \Sigma_2, \delta_2, st0_2, F_2 >$:

$$\begin{aligned}
&- Q_2 \subseteq Q_1 \\
&- \Sigma_2 \subseteq \Sigma_1 \\
&- \delta_2 : Q_2 \times \Sigma_2 \rightarrow Q_2 \\
&\qquad st, c \mapsto \delta_2(st, c) = \delta_1(st, c) \\
&- \forall st \in Q_2, \forall c \in \Sigma_2 : \delta_2(st, c) = \emptyset \Rightarrow st \in F_2 \\
&- st0_2 = st0_1 \\
&- F_2 \subset F_1
\end{aligned} \tag{10}$$

After generating RA_{IG}, the next step is to define the whole set of user task realisations (Figure 6), i.e., the set of automata that simulates A_T[8] based on the user

[8]When calculating the set of automata that simulates A_T it is possible to obtain realisations that are exact matching to the user task but also realisations that are compatible to the required one.

task (Figure 3) and the pervasive services (Figure 4). As we can see in Figure 6, there are two user task realisations, which means that there are two service compositions that are semantically compatible matching to the user task.

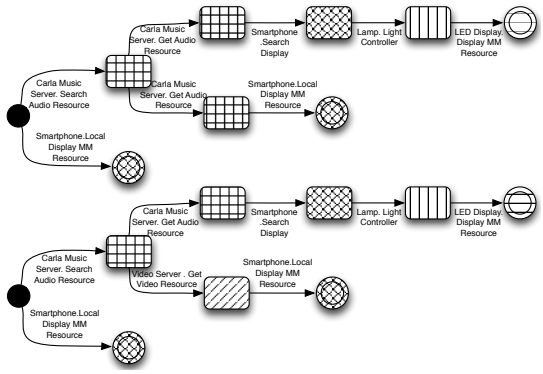


Figure 6: Conversation Integration User Task Realisations

3.2.4. Conversation Interleaving

In this section, we describe the second solution for dynamic user task realisation that supports interleaving of pervasive service conversations. Again, we build a *raw automaton* RA_{IL} from the set of selected services. However, this decidable automaton differs from that of the integration solution. RA_{IL} represents the *asynchronous free product* of the selected services automata. More formally, let $s_1, ..., s_K$ be the set of selected services and $A_1, ..., A_K$ their associated conversations, where $A_k =< Q_k, \Sigma_k, \delta_k, st0_k, F_k >$. $RA_{IL} =< Q_{RA}, \Sigma_{RA}, \delta_{RA}, st0_{RA}, F_{RA} >$ is defined as:

$$\begin{aligned}
&- Q_{RA} = Q_1 \times ... \times Q_K \\
&- \Sigma_{RA} = \cup_{k=1}^{K} \Sigma_i \cup \{\epsilon\} \\
&- \delta_{RA} : Q_{RA} \times \Sigma_{RA} \rightarrow Q_{RA} \\
&\quad (< st_1, ..., st_L >, c) \mapsto \delta_{RA}(< st_1, ..., st_L >, c) \\
&- st0_{RA} =< st0_1, ..., st0_L > \\
&- F_{RA} = \{< st_1, ..., st_L > \in Q_{RA} \mid \\
&\qquad st_1 \in F_1 \wedge ... \wedge st_L \in F_L\}
\end{aligned} \tag{11}$$

such that:

$$\begin{aligned}
&\delta_{RA}(< st_1, ..., st_L >, c) =< st'_1, ..., st'_L > \\
&if\ \exists\, l_1, l_2 \in \{1...L\} : st'_{l_2} = st_{l_2} \\
&\forall l_2 \neq l_1 \wedge \delta_{l_1} st_{l_1}, c = st'_{l_1}
\end{aligned} \tag{12}$$

The previous definition allows to create the *asynchronous free product* automaton of a set of conversations. But it only supports the full execution of all the conversations and it does not allow the full execution of a subset of all them. Thus, in order to support full execution of a subset, it is necessary to redefine which are the final states of the *asynchronous free product* automaton by means of an adaptation in the F_{RA} clause, replacing its definition by the next one: $F_{RA} = \{< st_1, ..., st_L > \in Q_{RA} \mid \forall st_l = F_l \vee st0_l\}$.

Based on this *raw automaton*, service coordination uses the relation **ConvMatch**(A_1, A_2) defined in Section 3.2.2 to find user task realisations. Specifically, there exists a realisation of the user task if the *raw automaton* RA_{IL} simulates the user task automaton A_T. More formally, this can be represented by the function **ConvInter**$(T, s_1, ..., s_K)$ as follows:

$$\begin{aligned}
&ConvInter : \mathcal{T} \times \mathcal{S}^K \longrightarrow 2^{\mathcal{T}} \\
&\qquad < T, s_1, ..., s_K > \longmapsto \{T_1, ..., T_J\}
\end{aligned} \tag{13}$$

such that:

$$\begin{aligned}
&RA_{IL} = RawAutomaton_{IL}(s_1, ..., s_K) \wedge \\
&\forall T_j \in \mathcal{T}, j = 1, ..., J\ \exists A_{T_j} : \\
&SubAutomaton(RA_{IL}, A_{T_j}), \\
&ConvMatch(A_T, A_{T_j})
\end{aligned} \tag{14}$$

Figure 7 presents a simplified version (using only a subset of the selected services) of the *raw automaton* built from the services of the environment that have been previously selected according to the user task of Figure 3. The Figure 7 only shows the *asynchronous free product* automata of the *Smart Environment Controller*, *Lamp* and two *LED Display* pervasive services.

Figure 6 shows the concrete realisations of the user task of Figure 3 for conversation integration and Figure 8 shows the concrete realisations of the user task of Figure 3 for conversation interleaving. While service conversation manages to find only two task realisations, service interleaving offers the possibility to the mobile user of choosing among four realisations (i.e., using either the music service offered by Carla or the video service offered by the environment video server). Thus, we can see that the conversation interleaving method offers more realisations than the integration method.

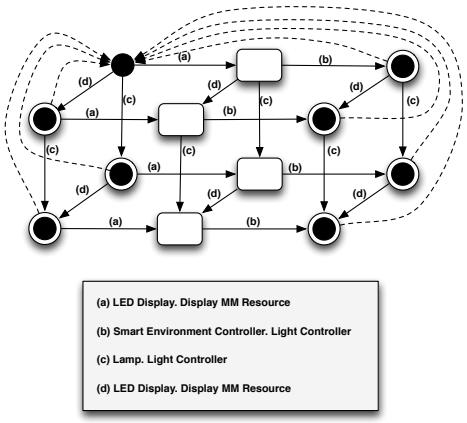


Figure 7: Raw automaton for Conversation Interleaving

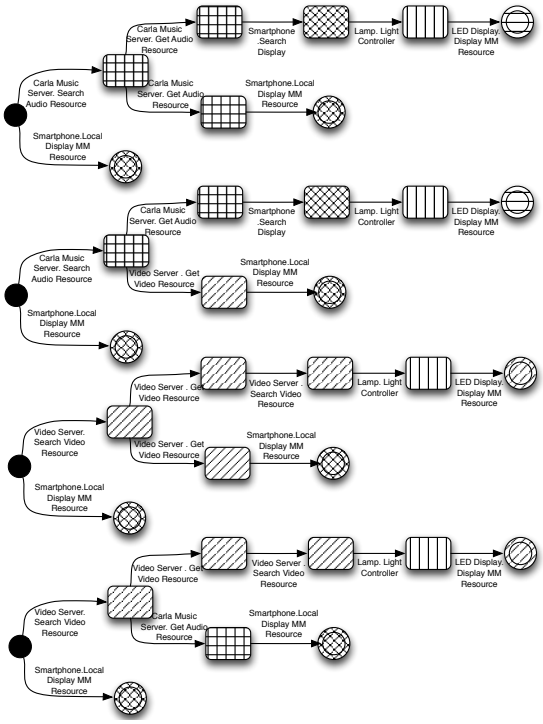


Figure 8: Conversation Interleaving User Task Realisations

3.2.5. Adaptive User Tasks

In this section, we present the last solution to the dynamic realisation of the user task. This solution allows finding a greater number of concrete realisations of the user task than the first two. It is based on the following observation: *The user task is defined by a service developer. Its conversation represents one of the possible ways of satisfying the user's intention*. Specifically, in the dynamic realisation of user tasks it may be possible to change the structure of the user task to increase the probability of finding a composition (this kind of adaptation is performed offline and does not imply any online adaptation technique, because it is not made on execution). The modification in the structure of the task constitutes in changing the order between capabilities, which can be done under three premises:

- Those capabilities that are not related with any data- or context-flow can be moved through the structure of the task, more precisely through the structure of the execution block (an execution block is defined as the path represented between the initial state of the automaton and the first branch; two structural branches without a branch in between; or between the beginning of the last branch and the final state of the automaton) where they are.
- Those capabilities that are part of a data- or context-flow can be moved in the next ways: If the capability is the source of the flow, it can be moved through the execution block, but it cannot be moved to a position that is after the capability that is the target of the flow. If the capability is the target of the flow, it can be moved through the execution block, but it cannot be moved to a position that is before the capability that is the source of the flow.
- The execution blocks cannot be moved through the automaton.

Based on the above definitions, a set of automata is generated from the user task conversation that contains all the rescheduling possibilities fulfilling the task data- and context-flow specifications, which represent different ways to achieve the same user task. These automata, called the *rescheduling automata*, created by the **ReschedulingAutomaton**(T) function, and noted SA in the following, are built based on a dependency graph between capabilities (where the complexity of the rescheduling process is related to the complexity of the user task). In order to create these automatons, it is necessary to follow a set of steps described as follows (using as an example the user task of Figure 3):

1. Define the *Flow Dependency Graph*, Figure 9, extracting from the graphical representation of the data- and context-flow of the user task by removing the automaton states as well as those transitions that are not part of a data- or context-flow.

This graph represents the constraints related with those capabilities that are part of at least a data- or context-flow.

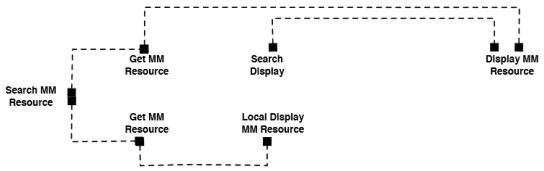


Figure 9: Flow Dependency Graph

2. Define the *Block Dependency Graph*, Figure 10, a graph where the capabilities are grouped by execution blocks. This graph represents the constraints related with the execution blocks, in order to define the range where the capabilities can be moved on.

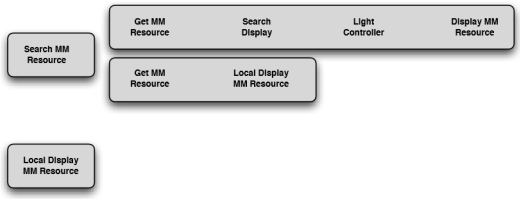


Figure 10: Block Dependency Graph

3. Define the *Integrated Dependency Graph*, Figure 11, a graph that integrates the previous two graphs. This offers a vision of the two types of restrictions (flows and execution blocks).

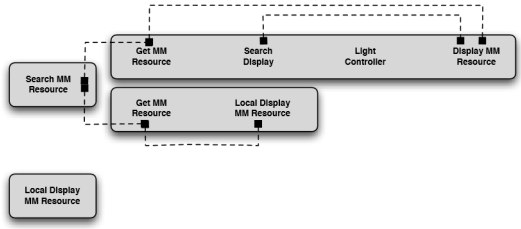


Figure 11: Integrated Dependency Graph

4. Define the *Final Dependency Graph*, Figure 12, representing only the flows whose source and target are located in the same execution block. To create this graph it is necessary to remove from the previous graph those flows whose target or source are located in different execution blocks, because the execution blocks itself are more restrictive than the flows.

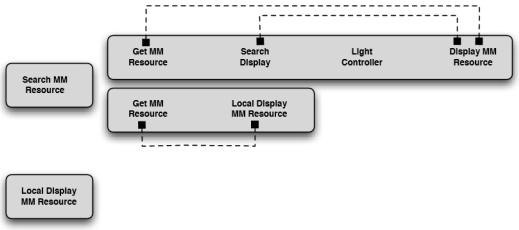


Figure 12: Final Dependency Graph

5. Once the *Final Dependency Graph* is defined, the next step is the generation of the set of possible combinations for each of the blocks where the capabilities can be moved. In this case, the only execution block whose capabilities can be moved is the one where the capabilities *Get MM Resource*, *Search Display*, *Light Controller* and *Display MM Resource* are chained. As a result of this process and taking into account the three premises described before, the graph *Combination Graph* is generated. In the Figure 13 we can see that eight possible combinations are defined based on the block whose capabilities can be moved.

The rescheduling automata built from the previous process are depicted in Figure 14. A set of eight user tasks, with different order of capabilities invocation but same behaviour, has been generated from a user task, respecting the constraints represented by the data- and context-flows. In this way, instead of having one user task to find its realisations, we have a set of user tasks to use, thus we have more chances to find user tasks realisations.

The rescheduling automata can then be compared using the conversation match method **ConvMatch**(A_1, A_2) with either the *raw automaton* RA_{IG} or RA_{IL} to find user task realisations with or without the support of conversation interleaving, respectively. This generates two additional solutions to the user task realisation that support the adaptation of the user task, defined with the two functions **AdaptInteg**$(T, s_1, ..., s_K)$ and **AdaptInter**$(T, s_1, ..., s_K)$ as follows (where M represents the number of user task combinations generated by the **ReschedulingAutomaton**(T) function). Thus **Adapt-**

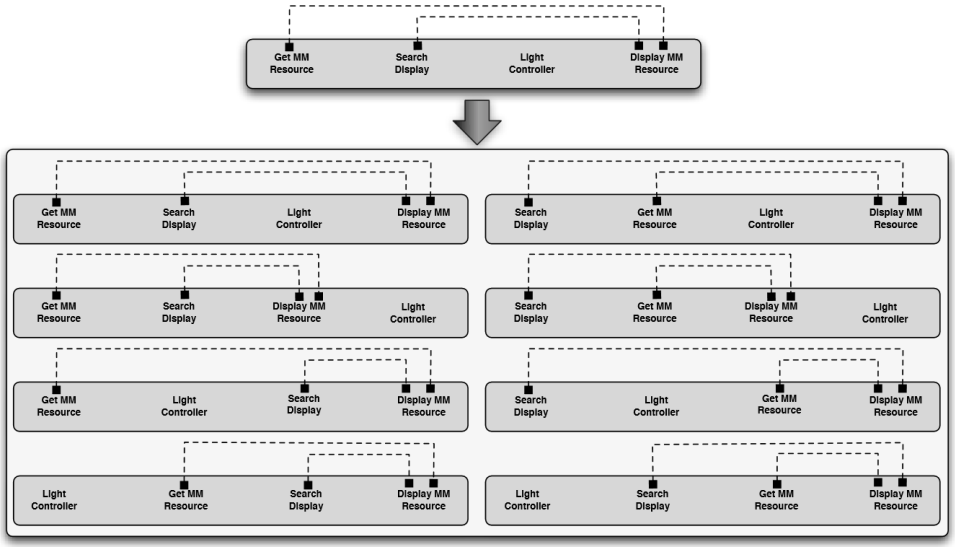


Figure 13: Combination Graph

Integ$(T, s_1, ..., s_K)$ is defined as follows:

$$\begin{aligned} &AdaptInteg : \mathcal{T} \times \mathcal{S}^K \longrightarrow 2^{\mathcal{T}} \\ &\quad < T, s_1, ..., s_K > \longmapsto \{T_1, ..., T_J\} \end{aligned} \tag{15}$$

such that:

$$\begin{aligned} &RA_{IG} = RawAutomaton_{IG}(s_1, ..., s_K) \wedge \\ &SA = ReschedulingAutomaton(T) \wedge \\ &\forall T_j \in \mathcal{T}, j = 1, ..., J \; \exists A_{T_j} : \\ &SubAutomaton(RA_{IG}, A_{T_j}), \\ &ConvMatch(SA_m, A_{T_j}) : m = 1, ..., M \end{aligned} \tag{16}$$

And **AdaptInter**$(T, s_1, ..., s_K)$ as:

$$\begin{aligned} &AdaptInter : \mathcal{T} \times \mathcal{S}^K \longrightarrow 2^{\mathcal{T}} \\ &\quad < T, s_1, ..., s_K > \longmapsto \{T_1, ..., T_J\} \end{aligned} \tag{17}$$

such that:

$$\begin{aligned} &RA_{IL} = RawAutomaton_{IL}(s_1, ..., s_K) \wedge \\ &SA = ReschedulingAutomaton(T) \wedge \\ &\forall T_j \in \mathcal{T}, j = 1, ..., J \; \exists A_{T_j} : \\ &SubAutomaton(RA_{IL}, A_{T_j}), \\ &ConvMatch(SA_m, A_{T_j}) : m = 1, ..., M \end{aligned} \tag{18}$$

Computed realisations will not exactly conform to the initial task conversation, but they will still fulfil the task data-flow and context-flow specification. While this solution increases the probability of finding service compositions, it is more costly than the first and second solutions, as there are more automata used as input of the **ConvMatch**(A_1, A_2) function.

4. Evaluation and Assessment

To evaluate the composition framework and its four composition variants (Integration, Integration + User Task Adaptation, Interleaving and Interleaving + User Task Adaptation), we first describe the methodology used to generate the collection of composite wEASEL services. After that, we introduce the metrics defined for the assessment of our framework and finally, we present the results of the evaluation of the composition mechanisms. The evaluation was carried out on a Raspberry Pi 3 Model B with a 1.2GHz 64-bit quad-core ARMv8 CPU and 1GB of RAM memory.

4.1. Simulation setup: generation of composite wEASEL services

We built a composite service test collection combining simple-capability services[9]. In order to do that, we analysed the data-flows of the simple-capability

[9] Available at http://tinyurl.com/wEASEL-site

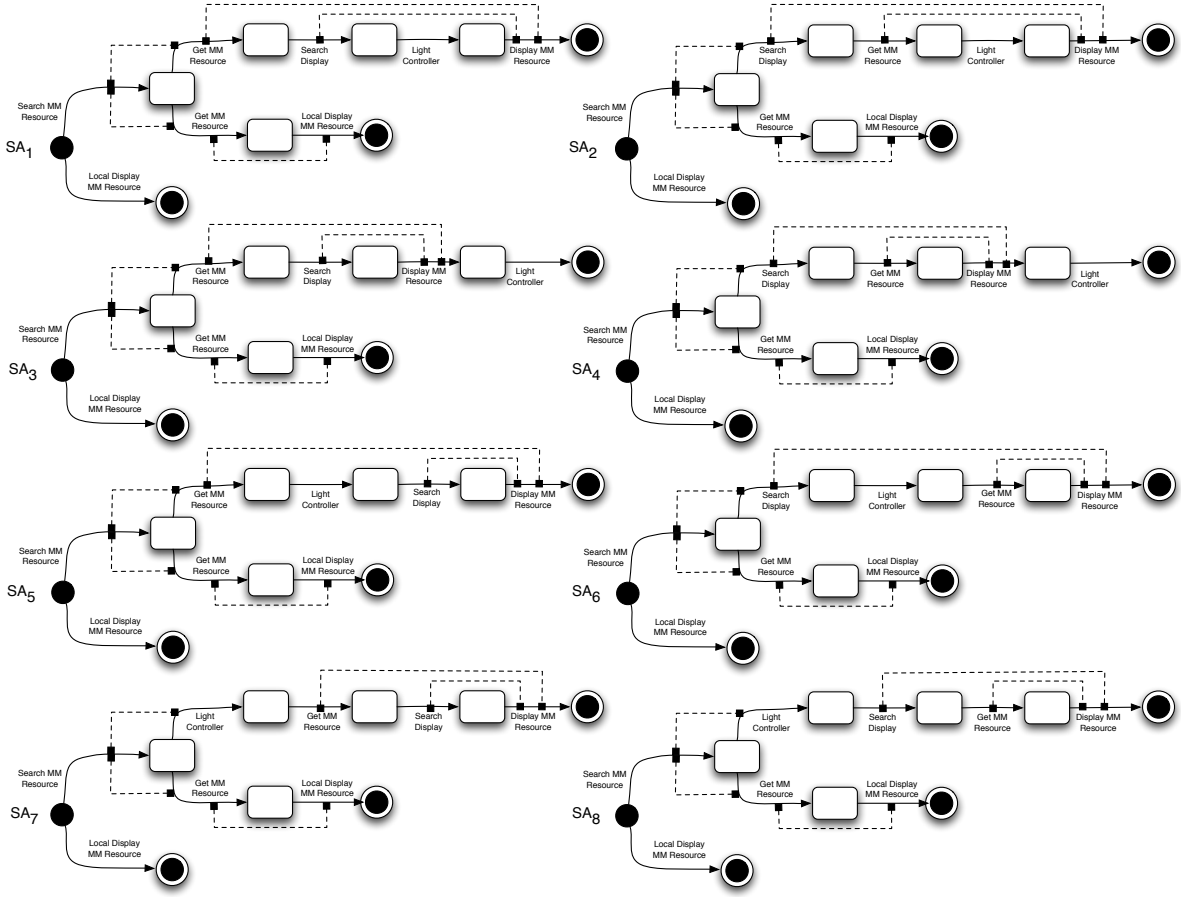


Figure 14: Rescheduling Automatons

wEASEL services converted from the OWLS-TC4 collection[10] in [3]. In other words, we applied our matching framework, defined in [3], to find output-input relationships between every simple-capability pair of services for each domain (matching one by one each of the services of a domain against all of the services of that domain). The IOPE Hybrid-Cosine method was chosen as it offers the best precision/recall results [3]. Besides, we only used the set of descriptions that are in the services folder of OWLS-TC4, rather than the ones that are in the queries folder. As the number of descriptions that are divided in domains is too small, they offer less possible combinations to create composite services for the testbed.

Considering the results from the matching framework, we analysed the total number of output-input combinations of the simple-capability services that would produce composite services. Table 2 shows the number of possible composite services per number of capabilities, ranging from 1 (for simple capability) to 4, found for each domain. We decided to set 4 as the maximum length of execution blocks (i.e., the maximum number of paths in a sequence for pervasive services), as this maximum is enough to build a rich set of composite services). As an example from Table 2, in the travel domain, we observed that there are 165 simple-capability services, 546 composite services with two capabilities that can be built using pairs of simple-capability services with an output-input matching, and 14.246 composite services with four capabilities. It is noted that while in some domains, e.g. economy and

[10]OWLS-TC4 is a test collection built to support the evaluation of OWL-S service matchmaking mechanisms. The OWLS-TC4 services are written in OWL-S 1.1. and are divided into nine different domains: communication, economy, education, food, geography, medical, simulation, travel and weapon. The collection is composed of 1.083 services and 42 queries. The queries are associated with relevance sets to allow for performance and correction evaluation experiments.

geography, the number of data-flows increases significantly per number of capabilities, other domains as communication, food and weapon, have no data-flows combining 2 or more capabilities.

Table 2: Number of services per number of capabilities, divided by domain

	Services			
	1 Capab.	***2 Capab.***	***3 Capab.***	***4 Capab.***
communication	58	0	0	0
economy	359	8.540	172.881	4.337.199
education	285	2.817	7.076	8.894
food	34	0	0	0
geography	60	7.773	748.719	63.872.720
medical	73	2.007	138.224	9.570.152
simulation	16	98	34	34
travel	165	546	2.761	14.246
weapon	40	0	0	0

In order to create the semantic service testbed including composite services by taking into account the results in Table 2, we selected the *travel* domain because of the following two reasons:

- It is the one that offers services descriptions that are closest to the scenario described in Section 1.
- It has an equilibrated number of data-flows for each transition length, i.e., from the set of simple-capability services it is possible to create sets of composite services of different lengths that are not as large (thus distributed) as the economy, geography and medical domains nor as small as communication, education, food, simulation and weapon domains.

Considering services from the travel domain, we created a smart environment-oriented testbed with three subsets of service types:

- 20 randomly selected simple services (services with one capability)
- 120 randomly selected composite services with several capabilities in sequence and just one execution branch. These services were built combining the 20 simple services of the travel domain. The 120 services were grouped according to their length, generating 40 services per each of the lengths. Additionally, and for each length, the 40 services were divided in two groups, where 20 of the services were represented with data-flow information and the other 20 were not represented with data-flow information. This is because in smart environments not all the services will be described using data-flow information.
- 240 randomly selected composite services with several capabilities but with more than one execution branch (constituting a choice). These services, in the same way as the ones mentioned before, were constructed combining the 20 services of the travel domain. In this case, the services are grouped according to the total number of capabilities in the composition and to the arity of the composition tree, i.e., the number of execution branches. Based on these two factors, we defined six composition groups: 3-2, 4-2, 4-3, 5-2, 5-3 and 5-4, where the first value represents the number of capabilities and the second value represents the arity of the composition tree. For example, the group 5-3 means that each service has five capabilities and the tree arity is three. For each group, we created 40 composite services, 20 composite services with data-flow information and other 20 without it as before.

Combining all the services of the previously described three subsets, we obtained a testbed of 380 (20+120+240) services. Where we created random subsets of 50 services (simple and composite), assuming that it is not common to find more than 50 services in a smart environment. These random subsets were used as input for the evaluation units described in Section 4.2.

4.2. Metrics

In order to evaluate the situation heterogeneity and dynamism characteristics, we evaluated two aspects of the service composition mechanism: its performance (to evaluate the capacity of the composition engine to respond to the appearance of new services in the environment and thus to calculate if those services can be part of new user task realizations) and its correctness (to evaluate the capacity of the composition engine to deal with heterogeneously described services). The present manuscript deals with the correctness of the conversation matching of user task realizations but also with the semantic correctness of the capabilities of the user tasks. However, the semantic correctness used in this manuscript is presented and thoroughly evaluated in [3], where the best precision/recall balance is achieved by the IOPE Hybrid-Cosine variant, which is also used as the grounding matchmaking algorithm for the composition variants. In the present manuscript the semantic correctness is achieved through the **SigMatch**(Sig_1, Sig_2) and **SpecMatch**(Spe_1, Spe_2) relations that are used in the **ConvMatch**(A_1, A_2) definition, which is presented on Equation 4.

For the performance evaluation we measured the time that the composition mechanism takes to return user task realisations, where we considered the following two aspects:

- We evaluated the average time for each of the four stages in the composition process (i.e., Candidate Selection, Raw Automaton Generation, User Task Adaptation and computation of the user task realisations list, namely Automata Simulation), in order to determine the for each variant which is the most time-consuming stage.
- We evaluated the average total time required by the service composition framework for each of the composition variants, in order to determine the variant that offers better performance.

The evaluation of the correctness consisted on calculating the average number of user task realisations retrieved for each composition variant. Therefore, we could determine which is the variant offering the best performance/correctness rate for smart environments.

We divided the evaluation process in evaluation units composed of:

- Services of the environment: we randomly selected a subset of 50 services (as it has been stated in Section 4.1) from the all services in the testbed.
- User Task: a service was randomly selected from the all services in the testbed.

A total of 100 evaluation units were built, and the combined results of the evaluation units were used to extract the average values shown in Section 4.3.

4.3. Results

In this section, we describe the results of the evaluation of performance and correctness, for a total of 100 units of evaluation of the travel domain. Figure 15(a) shows the results of average time for each of the stages per composition variant, from which we can obtain the following conclusions:

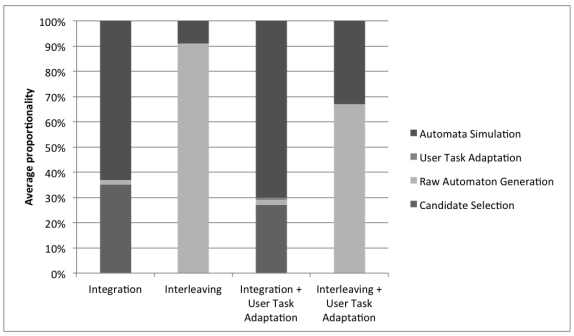
- The stage that takes more time for the variants that use Integration is the process of Automata Simulation, while the Raw Automaton Generation is the subprocess that takes more time for the variants that use Interleaving, as it is a very time-consuming activity.
- The subprocess of Candidate Selection is negligible for the variants that use it, and the subprocess of User Task Adaptation is negligible (due to the simplicity of the subprocess of the adaptation, but mainly due to the low efficiency of the Raw Automaton Generation) for the two variants that use it.
- The subprocess of Candidate Selection is quite costly for the Integration variants, but is negligible for the Interleaving variants.

From the evaluation of the total average time for the composition process per each variant (see Figure 15(b)), we conclude that:

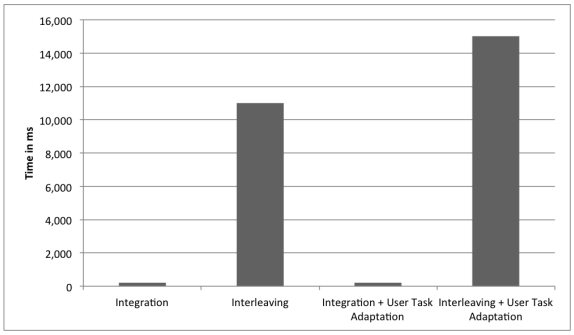
- The variants that use Interleaving are much slower (between 56 to 91 times slower) than the ones that use Integration. This is due to the point of the subprocess of Raw Automaton Generation on the Integration variants.
- However, as the Raw Automaton Generation subprocess is very simple for the Integration variants, it implies that these variants are very fast and that the Automaton Simulation subprocess is much faster than for the variants that use Interleaving. Both of the variants that use Integration offer a very similar performance.

For the correctness evaluation, we considered all the composition variants against the same set of 100 evaluation units, recording the number of user task realisations retrieved for each pair of variant and unit. After that, we calculated the average number of user task realisations obtained by each variant. In Figure 16, we can appreciate that the Interleaving + User Task Adaptation composition variant is the one that offers a higher number of average number of user task realisations but in contrast it offers also the worst performance. On the contrary, the Integration variant is the one that offers the smallest set of average user task realisations but offers the highest efficiency followed by the Integration + User Task Adaptation and Interleaving variants.

In smart environments, it is more important to retrieve the results in a fast way (in order to invoke the selected service) than to retrieve all the possible user task realisations. Thus, according to the previous analysis of the relation between the correctness and the performance of the composition variants, we conclude that the Integration + User Task Adaptation variant is the one that better fulfils those requirements. The Integration + User Task Adaptation variant offers an average of almost 20 user task realisations in less than 200 ms. It shows similar performance to the Integration variant but it offers a higher correctness (an average of 4 times



(a) Subprocesses average proportionality per variant



(b) Average time (ms) for the composition process per variant

Figure 15: Efficiency evaluation results

more realisations). In comparison to the variants that use Interleaving, it is near 75 (for Interleaving + User Task Adaptation) and 56 times faster (for Interleaving) but it only offers an average of seven (for Interleaving + User Task Adaptation) and an average of two (for Interleaving) times less user task realisations.

In the present section we have shown the evaluation results for the travel domain. However, we have also carried out the same evaluation process for the rest of the domains and even if the results are different, the proportionality between the different techniques is equivalent. That is to say, the results are equivalent for different domains, thus, the composition mechanism is domain independent.

5. Conclusion

The large number of IoT devices with increasing computing power and networking capabilities is allowing the smart city vision to become a reality. However, the ability to integrate functionalities offered by IoT devices in the vicinity of mobile users is still an unresolved challenge. To achieve this, the context of the service is of paramount importance. In this paper, we

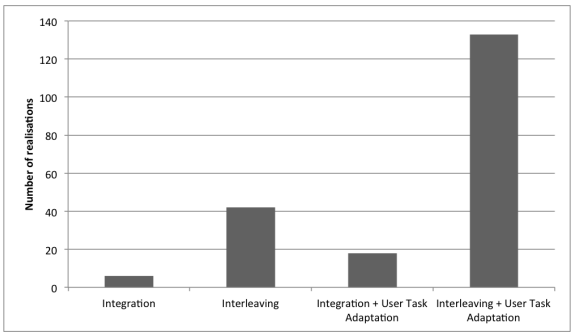


Figure 16: Average user task realisations per variant

proposed a framework that deals with the composition of context-aware services based on our previously published wEASEL abstract service model and matchmaking engine [3], producing the following contributions:

- a novel dynamic and adaptive service composition framework that allows several variants: service conversation integration, service conversation interleaving and user task adaptation,
- a testbed of composite semantic services derived from the single-capability services of OWLS-TC4 and a methodology to build it,
- a thorough evaluation of our service composition framework algorithms, considering both the performance and the quality of the results.

In light of the results and analysis, as part of future work we aim to:

- Develop techniques to create the *raw automaton* on-the-fly instead of building the product automaton for the interleaving technique.
- Extend the composition framework to support user task realizations where the number of their capabilities is greater than the required ones
- Integrate the composition variants with composition engines in order to deploy wEASEL-Com alike devices that are able not only to construct the composition but also to deploy it on the available devices and control them on a centralized way.
- Apply the proposed approach on a different scenario (e.g Industry 4.0 environment), in order to validate its viability. One of these projects is the CREMA[11] H2020 RIA project, where IK4-IKERLAN is participating.

[11] Cloud-based Rapid Elastic MAnufacturing (CREMA): `http://www.crema-project.eu/`

To conclude, we state that our wEASEL-based composition framework allows us to be closer to the smart cities vision, thanks to the different variants of the proposed service composition. More specifically, the combination of the two different composition variants, *Integration + User Task Adaptation*, is the one that better fulfills the smart cities service integration requirements.

Acknowledgements

This work is partially supported by the Commission of the European Union within the CREMA H2020-RIA project (Grant agreement no. 637066) and by the Basque Government's Elkartek program within the LANA II project (Grant agreement no. KK-2016/00052).

References

[1] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. R. Biswas, K. Moessner, Enabling smart cities through a cognitive management framework for the internet of things, Communications Magazine, IEEE 51 (6) (2013) 102–111.

[2] J. M. Hernández-Muñoz, J. B. Vercher, L. Muñoz, J. A. Galache, M. Presser, L. A. H. Gómez, J. Pettersson, Smart cities at the forefront of the future internet, Springer, 2011.

[3] A. Urbieta, A. González-Beltrán, S. B. Mokhtar, J. Parra, L. Capra, M. A. Hossain, A. Alelaiwi, J. I. Vázquez, Hybrid service matchmaking in ambient assisted living environments based on context-aware service modeling, Cluster Computing 18 (3) (2015) 1171–1188.

[4] V. Ramasamy, Syntactical and semantical web services discovery and composition, in: Proceedings of the cec-eee'06 conf., 2006.

[5] R. Masuoka, B. Parsia, Y. Labrou, Task computing - the semantic web meets pervasive computing, in: 2nd International Semantic Web Conference (ISWC2003), 2003.

[6] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau, Automating DAML-S web services composition using SHOP2, in: Proceedings of 2nd International Semantic Web Conference (ISWC'03), 2003.

[7] H. Q. Yu, S. Reiff-Marganiec, A backwards composition context based service selection approach for service composition, in: IEEE International Conference on Services Computing, 2009, 2009, p. 419.
URL `http://oro.open.ac.uk/24964/`

[8] D. Chakraborty, A. Joshi, T. Finin, Y. Yesha, Service Composition for Mobile Environments, Journal on Mobile Networking and Applications, Special Issue on Mobile Services 10 (4) (2005) 435–451.

[9] R. Aggarwal, K. Verma, J. Miller, W. Milnor, Dynamic web service composition in meteor-s, Tech. rep., LSDIS Lab, Computer Science Dept., UGA (2004).

[10] W. L. C. Lee. S. Ko, S. Lee, A. Helal, Context-aware service composition for mobile network environments, in: 4th International Conference on Ubiquitous Intelligence and Computing (UIC2007), 2007.

[11] A. Bottaro, J. Bourcier, C. Escoffier, P. Lalanda, Autonomic context-aware service composition, in: 2nd IEEE International Conference on Pervasive Services (ICPS'07), Istanbul, Turkey, 2007.

[12] N. Ibrahim, F. Le Mouel, S. Frenot, Mysim: a spontaneous service integration middleware for pervasive environments, Proceedings of the 2009 international conference on Pervasive services (2009) 1–10.
URL `http://portal.acm.org/citation.cfm?id=1568201`

[13] B. Lagesse, M. Kumar, M. Wright, Resco: A middleware component for reliable service composition in pervasive systems, 2010 8th IEEE Int. Conf. on Pervasive Computing and Communications Workshops (2010) 486–491.
URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5470620`

[14] M. Klein, A. Bernstein, Toward high-precision service retrieval, IEEE Internet Computing 8 (1) (2004) 30–36.
URL `http://dx.doi.org/10.1109/MIC.2004.1260701`

[15] C. Kiefer, A. Bernstein, The creation and evaluation of isparql strategies for matchmaking, in: 5th European Semantic Web Conference (ESWC 2008), Vol. 5021, Springer, 2008, p. 463.

[16] A. Brogi, R. Popescu, Towards semi-automated workflow-based aggregation of web services, in: Proceedings of Third International Conference on Service Oriented Computing (ICSOC'05), 2005.

[17] A. Sirbu, J. Hoffmann, Towards scalable web service composition with partial matches, 2008 IEEE International Conference on Web Services (2008) 29–36.
URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4670156`

[18] A. Sirbu, A. Marconi, M. Pistore, H. Eberle, F. Leymann, T. Unger, Dynamic composition of pervasive process fragments, 2011 IEEE International Conference on Web Services (2011) 73–80.
URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6009374`

[19] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, M. Mecella, Automatic composition of web services in colombo, in: Proceedings of the 13th Italian Symposium on Advanced Database Systems (SEBD'05), 2003.

[20] S. B. Mokhtar, N. Georgantas, V. Issarny, Cocoa: Conversation-based service composition in pervasive computing environments with qos support, Journal Of System and Software 80 (12) (2007) 1941–1955.

[21] B. Benatallah, F. Casati, F. Toumani, Representing, analysing and managing web service protocols, Data & Knowledge Engineering 58 (3) (2006) 327–357.

[22] S. V. Hashemian, F. Mavaddat, A logical reasoning approach to automatic composition of stateless components, Fundam. Inf. 89 (2008) 539–577.
URL `http://dl.acm.org/citation.cfm?id=1497115.1497123`

[23] M. Mancioppi, M. Carro, W. Van den Heuvel, M. Papazoglou, Sound multi-party business protocols for service networks, Service-Oriented Computing–ICSOC 2008 (2008) 302–316.

[24] A. Urbieta, G. Barrutieta, J. Parra, A. Uribarren, A survey of dynamic service composition approaches for ambient systems, in: ACM First International Conference on Ambient Media and Systems (Ambi-Sys 2008)- ACM First Workshop on Software Organisation and MonIToring of Ambient Systems (Somitas 2008), ACM, ICST, 2008.

[25] T. Stavropoulos, D. Vrakas, I. Vlahavas, A survey of service composition in ambient intelligence environments, Artificial Intelligence Review.