Reliable Scientific Service Compositions*

Bruno Wassermann and Wolfgang Emmerich

University College London
Dept. of Computer Science
Software Systems Engineering Group
Gower Street, London, WC1E 6BT, UK
{b.wassermann,w.emmerich}@cs.ucl.ac.uk
http://sse.cs.ucl.ac.uk

Abstract. Distributed service oriented architectures (SOAs) are increasingly used by users, who are insufficiently skilled in the art of distributed system programming. A good example are computational scientists who build large-scale distributed systems using service-oriented Grid computing infrastructures. Computational scientists use these infrastructure to build scientific applications, which are composed from basic Web services into larger orchestrations using workflow languages, such as the Business Process Execution Language. For these users reliability of the infrastructure is of significant importance and that has to be provided in the presence of hardware or operational failures. The primitives available to achieve such reliability currently leave much to be desired by users who do not necessarily have a strong education in distributed system construction. We characterise scientific service compositions and the environment they operate in by introducing the notion of global scientific BPEL workflows. We outline the threats to the reliability of such workflows and discuss the limited support that available specifications and mechanisms provide to achieve reliability. Furthermore, we propose a line of research to address the identified issues by investigating autonomic mechanisms that assist computational scientists in building, executing and maintaining reliable workflows.

1 Introduction

Achieving reliability is a key concern in the design of distributed software systems. In this paper we argue that the service-oriented Grid computing infrastructures that have attracted computational scientists as a new set of non-expert users currently only provide inadequate support at both design and runtime to cater for reliability. As we demonstrate, this gap is increased by the fact that scientific service compositions suffer from challenging threats to their reliability. Due to the proliferation of scientific service-oriented applications, it is important to investigate what kind of additional support can be offered to their developers and users.

* This research has been funded by the UK EPSRC through grants GR/R97207/01 (e-Materials) and GR/S90843/01 (OMII Managed Programme)

If computational scientists are not provided with more effective means to tackle issues of reliability, then this will present a serious impediment to the successful use of service-oriented technologies in scientific computing and thereby limit the realisation of its benefits. We therefore want to raise awareness, characterise the problem, and propose a line of research to build autonomic mechanisms that can enable non-expert users to build and execute reliable service compositions by handling failures automatically whenever possible and through meaningful interaction with human users in any other cases.

The main contribution of this paper is the characterisation of scientific service compositions and the environment they operate. We outline the ample threats to their reliability. We do this by introducing the notion of global scientific BPEL workflows (section 2) to get a clearer picture of what computational scientists need to be enabled to deal with. Then, we briefly present a typical instance of a scientific service composition and the failures it suffers from (section 3). In order to demonstrate that current mechanisms and specifications have failed to address the issue of reliability successfully, we review existing approaches (section 4). The second contribution of this paper lies in a proposal outline to investigate the application of autonomic mechanisms to achieve reliability and ways of effective interaction between these mechanisms and human users to achieve better coverage (section 5), before discussing closely related work (section 6).

2 Global Scientific BPEL Workflows

2.1 Scientific Workflows

Computational sciences have increasing demands on compute power, data storage capacity and collaboration across organisational boundaries. These requirements are satisfied by modern Grids, which have evolved into service-oriented computing environments comprised of collections of basic Web services. In order to express scientific experiments, these basic services need to be composed into larger orchestrations. In prior work, we have shown that the Business Process Execution Language (BPEL) as the industry standard for Web service orchestrations has shown to be suitable for this task and it is desirable for faster turnaround of ideas for experiments for computational scientists to take control of their own orchestrations. Computational scientists have been enabled to model scientific workflows through the tool offering developed by the OMII-BPEL project [1] and [2]. In this section, we briefly characterise the key elements of such compositions, or workflows, to identify their impact on reliability.

Scientific workflows display some interesting properties. They operate on a large scale, both in terms of the number of operations they invoke, the degree of parallelism, the size and number of messages they exchange with service partners and the amount of data they handle. Consequently, and given the nature of the computations they are designed to handle, scientific workflows are resource-intensive and long-running, which makes them prone to resource exhaustion (i.e. memory, threads, file descriptors,) and increases the likelihood of internal

or latent errors from various components materialising themselves as critical failures. Furthermore, scientific workflows often operate and employ resources in a wide-area setting, which introduces further issues with respect to their stability. This state of affairs is not helped by the heterogeneity of the underlying operating systems and hardware and the fact that resource schedulers, such as Condor are explicitly addressing the scavenging of unused CPU cycles, which results in termination or relocation of computation when nodes are beginning to be used again or when nodes are actually switched off.

Computational scientists are certainly computer literate and may posses some excellent programming skills in certain languages (most notably FORTRAN, C and C++). However, they should by no means be regarded as experts in distribution middleware and the underlying technologies used in service-oriented Grid computing and BPEL enactment environments. Therefore, they will benefit from simple to use mechanisms that provide support for ensuring the reliability of scientific service compositions.

2.2 Global Computing

Research collaborations are increasing in size and often involve participating organisations, which are geographically widely dispersed. Wide-area distribution enables such collaboration and increases the capacity to handle larger computational loads. Scientific workflows must integrate resources that are distributed over wide-area settings for various reasons. First, the computations exposed by scientific services are typically resource-intensive and their compositions require the exchange of large numbers of SOAP messages. Provisioning all required resources within a single organisation could easily become prohibitively expensive. Second, some services and the expertise they encapsulate are developed and maintained by individual organisations, which then make such services available for invocation via the Internet, but may not release the source. A third instance of resource sharing arises out of the need to pool Grid compute nodes. In such a setting the resource managers (exposed as Web services) responsible for job scheduling are local to the actual compute nodes and will have to be accessed by their clients over a wide-area network.

This makes scientific workflows expressed in a service-oriented computing environment a prime example of global computing, in which the components of an application are distributed across the Internet. Cardelli asserts that wide-area computing systems are fully asynchronous distributed systems making several new phenomena visible that could previously be hidden to a sufficient extent on LANs [3]. These observables include barriers (e.g., firewall) introduced due to the involvement of separate administrative domains and unpredictably fluctuating network conditions making long delays indistinguishable from failures. This is the category of applications scientific service compositions are a part of.

Being an instance of global computing systems has an impact on the available options for ensuring the reliability of scientific workflows. We cannot rely on timeouts to determine process failures and even if we were to ignore the impossibility result of reaching consensus in an asynchronous distributed system [4],

relying on mechanisms such as fault detectors may be prohibitively expensive. Techniques primarily developed for mobile networks, such as for example probabilistic broadcasts have no direct feasible application as they may rather resemble a DDoS attack (but see [5]).

2.3 Middleware Components

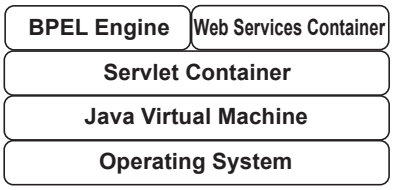

Fig. 1. Stack of high-level middleware components involved in hosting and running BPEL workflows.

Compositions need to be modelled using the tools of a buildtime in such a way as to ensure subsequent reliable execution in a runtime. In OMII-BPEL, this buildtime involves graphical modelling environments whose features assist users in designing, validating, debugging and deploying a workflow in an executable format. There is merit to briefly examine what a typical runtime consists of.

We consider middleware components as they occur in a typical Java environment, shown in Fig. 1. Distributed scientific applications may rely upon some or all of these middleware components in order to provide correct service. However, a considerable degree of complexity arises from the various middleware components and their interactions with each other, which can give rise to various failures. For example, the limitations on the number of sockets, threads or size of memory per process imposed by the operating system can lead to conditions causing the servlet container to crash. This then causes the subsequent failure of one or more parts of a workflow. Or, some problem in the servlet container preventing clients from accessing a particular resource (e.g., an XML Schema), may, via a chain of dependencies, cause a remote service to terminate abnormally. Such failures are extremely difficult to debug as none of the components involved provides much useful information.

Each of the middleware components contributes independent failure modes and each application may exercise different parts of these components under varying conditions. Hence, in order to ensure the reliable execution of a composition, a process modeller must reason about how potential failures in the middleware may influence their applications' ability to provide correct service. This may be a formidable challenge for software engineers, but presents a wholly unacceptable burden on computational scientists.

3 Example: Failures in the Polymorph Search Workflow

Scientific workflows are subject to many different failures. These failures often have no direct obvious cause and can have complex effects, such as cascading failures of components. In this section we briefly present a typical scientific BPEL workflow and discuss some of the failures that it experiences in practice. We refer the reader to [1] for a more detailed account of this workflow.

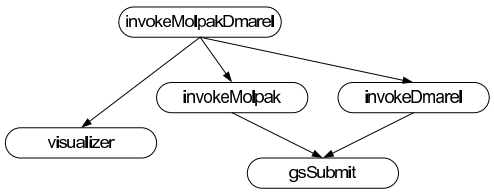

Fig. 2. Abstract overview of the polymorph search workflow illustrating its component sub-workflows. For simplicity, omits details of the services involved and distribution of services and sub-workflows.

The domain of the polymorph search workflow is Theoretical Chemistry and its application is the computational prediction of organic crystal structures. Its characteristics are typical of a realistic scientific workflow. It involves massively parallel computations, at times executing up to 7,600 service invocations concurrently. The individual compute jobs resulting from these invocations take anything between two minutes and several hours to complete. The total data volume resulting from a single polymorph search is in the region of 6 GB and parts of this data will be exchanged among sub-workflows and other services in a large number of SOAP messages.

In order to conquer the size and complexity of the polymorph search workflow, it has been designed and built as several BPEL processes, which are hierarchically composed so that a main process coordinates among several sub-workflows. An abstract overview is shown in figure 2. The invokeMolpakDmarel workflow (top-level) starts by gathering some input data and then invokes a number of instances of the invokeMolpak sub-workflow in parallel. As results become available from this, they are fed to a large number of concurrently executing instances of invokeDmarel. Both of these workflows make use of the gsSubmit sub-workflow, which encapsulates the steps necessary to submit compute jobs to a Grid via the GridSAM job submission and job monitoring services [6]. As the results of individual invokeDmarel invocations are returned, the top-level workflow submits them to the visualizer sub-workflow. Visualizer uses a Web service hosted by Southampton University to present molecule data in a standard tabular format and to render results on a scatter plot.

A selection of the failures experienced with the polymorph search workflow helps to illustrate the brittleness such compositions suffer from.

Omission failures. Some omission failures may go undetected and therefore result in corrupted results. For example, an instance of the visualizer process may fail due to its partner service in Southampton being temporarily unavailable, exhaustion of disk space, etc. As the visualizer sub-workflow provides for one-way invocation through its interface, such a failure will go undetected. A user would be left to manually inspect the resulting data for this omission as no part of the system capable of resolving the issue may have become aware of it.

Cascading failures. There are various scenarios that cause cascading failures to occur. One example for this is when an invoked service tries to reply to its caller after the latter has failed. The invoked service will fail as well and it is easy to see how this may lead to further cascading failures in a system of hierarchically composed sub-workflows. Furthermore, the Web service container of the invoked service may merely report the caller's failure as a broken connection.

Application-specific failures. A service being faced with a compute job which will never complete has little information available to decide how to handle this job and will itself never terminate (unless through resource exhaustion). A user however could in theory inspect this job and decide that it is an irrelevant outlier and should be discarded. However, there is currently no automated mechanism to establish this link from detection of the problem to making a human operator attentive to it and supplying further information.

4 Existing Approaches

In this section we present a brief survey of the key facilities made available by BPEL and various Web service specifications for handling failures.

4.1 Transactional Mechanisms

The traditional transaction model based on the four properties of atomicity, consistency, isolation and durability (ACID) has been applied with great success in database management systems (DBMS). Its success for short-lived transactions is, to a large extent, due to the fact that it effectively handles concurrency and failures on behalf of a programmer. It would therefore seem to also afford a convenient implementation of backward error recovery in the context of scientific workflows.

However, whilst ACID transactions are necessary and useful for certain cases in service-oriented applications, it is well known that they are of limited use in large-scale, long-running processes [7] and advanced transactions models (ATMs) have been devised that relax some of the properties of ACID transactions. ATMs allow programmers to focus on business logic rather than reason about exceptional executions by providing runtime support for handling failures and concurrency similar to the one afforded by ACID transactions.

Concepts from various ATMs have found application in workflow management systems (WFMS). For example, [8] has applied the concepts developed to preserve reliability in multidatabase systems [9] to WFMSs. In [8] a single

workflow/process represents a global transaction and its individual activities represent sub-transactions. Consequently, process modellers need to equip activities with transactional characteristics, such as compensatable (effects can be undone), retriable (will eventually commit) and pivot (either commits or fails) [9]. This affords the definition of a well-formed process, in which a single pivot activity is preceded only by compensatable activities and followed by retriable activities. In case the pivot activity fails, all previous ones can be undone, and in case it succeeds, all following activities are guaranteed to succeed eventually. Therefore, processes which are structured so as to adhere to the concept of well-formedness, can then achieve semi-atomicity, which affords preservation of the local autonomy of participants, whilst still preserving consistency in the presence of failures.

Although these concepts solve some of the issues we identified with the use of ACID transactions, there are a number issues when applied in our context. First, for computational scientists, the execution characteristics of individual activities in their workflows are far from obvious and reasoning about this is complex. Second, the constraints imposed by the property of well-formedness are too restrictive in practice when applied to large scientific workflows. The primary reason for this is that scientific workflows make use of hierarchical composition of a number of sub-workflows in order to conquer the complexity of large compositions during design and maintenance. Third, even if we were to relax the property of well-formedness, as shown in [10] in the context of MDBMs by introducing flexible transactions with retriable alternatives, the resulting guarantee of *eventual* reliability, that is, a guarantee that an activity will succeed at some point in the future, may introduce considerable delays. Instead, it would be preferable to detect and resolve an issue sooner rather than later, possibly by notifying a human operator.

4.2 BPEL Compensation-Handling

BPEL provides process modellers with various tools to build reliable workflows. It comes equipped with constructs to handle faults similar to the exception handling constructs in modern programming languages. It furthermore offers constructs to carry out compensation.

The concept of compensation as implemented in BPEL is restrictive and complex. In [11] the authors identify the combination of explicit and implicit compensation in BPEL as a main source of this complexity and question whether this added complexity is actually justified by any benefits. The lack of control for steering compensation provided to process modellers is criticised in [12]. It is furthermore noted that there is no support for reasoning about the correctness of an overall workflow in case compensation has been applied. It is also not the case that services usually come equipped with compensating operations, which is a problem in cross-organisational compositions where a developer may have no control over another organisation's services. In our experience, implementing forward error recovery in BPEL is complicated by the assumptions its relevant constructs are based on; immediate termination and backward error recovery.

This restriction is revealed in the BPEL specification, which states that the sole aim of fault handling in BPEL is to undo the effects of an unsuccessful scope. Yet, achieving forward error recovery whenever possible is of utmost importance in scientific workflows and should be made as simple as possible.

4.3 WS-Reliability

A number of specifications have been defined in order to increase the reliability of Web services. A crucial component of reliable distributed systems is reliable message delivery. There are two competing, but rather similar, specifications in the area of WS-Reliability (WS-R) area ([13, 14]). WS-R supports the reliable exchange of SOAP messages between endpoints and allows applications to configure parameters such as message delivery semantics and timeouts. Message queuing systems suggest themselves as an implementation of WS-R.

Whilst the service offered by WS-R provides an important component to maintain reliability, two issues become apparent in practice. First, the cost that arises from maintaining message queues does not bode well for scientific workflows. This cost arises from persisting messages in some form of database and includes storing additional message histories in the case exactly-once semantics are required. During a single run, a scientific workflow may make thousands of service invocations and may consequently send and receive in the region of tens of thousands of SOAP messages. Second, WS-R guarantees that a message will be delivered eventually. In a loosely-coupled, highly distributed environment involving different administrative domains it becomes difficult to predict for how long a particular set of services may be unavailable to process an incoming message. That is, the delay introduced by the concept of eventual reliability can be significant and an opportunity to detect and resolve a failure is missed.

In summary, we find that even though there are mechanisms to address reliability, they cannot be easily applied to scientific service compositions. The main characteristics of such compositions (long-running, resource-intensive, highly distributed) make the use of ACID transactions impractical. ATMs impose restrictions on the structure of workflows, which are difficult to adhere to in practice. Due to the demand for forward error recovery whenever possible and due the described complexity, which makes it difficult to anticipate all possible failures, BPEL's compensation constructs often fail to provide adequate support. The cost incurred by WS-R may actually be prohibitive and the range of failures encountered by scientific service compositions cannot be solved by reliable messaging alone. This leads us to the question what is actually needed to enable computational scientists to build reliable, fault-tolerant service compositions.

5 Making Reliability Useable

Making service-oriented Grid computing infrastructures directly 'programmable' by computational scientists has a number of benefits to offer that can advance scientific computing.

However, current mechanisms to address reliability are lacking in various respects. This forces computational scientists who develop complex compositions to engage in a lengthy, time-consuming and often frustrating process of trial-and-error where vulnerabilities are discovered through numerous runs of a workflow and protected against by piecemeal modifications. This process of 'design-by-trial-and-error' is not acceptable. The aim of our proposed research is therefore to enable computational scientists to design and execute global scientific BPEL workflows with reasonable trust that the composition will handle any failures and progress forward to completion.

We can derive a number of key features that any reasonable solution should offer. It is desirable to detect failures as soon as possible so that they can be handled and the overall workflow is able to progress forward. This is in contrast to notification of failures by timeouts and undoing a great deal of work in light of failures. In cases where it may be impossible to avoid undoing already completed computations, the least amount of work that needs to be undone in order to proceed to completion should be identified. Of course, any automated handling of failures should be efficient and above all lead to correct behaviour of the system. Achieving correct behaviour is complicated in the case of application-specific failures. Last but not least, it is of crucial importance to allow computational scientists to interact with any autonomic fault handling mechanisms in an intuitive manner so as to be able to indicate desired behaviour and possibly to increase coverage. Amongst other things, this means that autonomic failure handling must be able to operate satisfactorily with the least amount of input from users.

Our proposed solution for achieving our stated objective consists of three major parts.

Failure investigation service. Failure investigation is a crucial element in enabling clever(er) handling of failures. Our experience with scientific BPEL workflows suggests that it may often be possible to handle otherwise fatal failures successfully, if only there was more information available to drive autonomic failure handling mechanisms. The design and implementation of a failure investigation service pose a number of interesting questions to be addressed. One question is what kind of infrastructure is actually needed and what kind of information such an infrastructure should provide. Another issue is to determine what level of support can be achieved without being concerned about providing a global view of system state.

Autonomic recovery strategies. In order to handle failures and enable forward progress, autonomic recovery strategies need to monitor the various components involved in scientific workflows and then take action to prevent the various parts of a workflow from terminating abnormally. There are a number of complications. First, there are many components which may suffer under very different kinds of conditions and therefore require specialised failure investigation and recovery. Second, given the limited degree of software engineering expertise of our users, we cannot expect them to inform such recovery strategies through, for example, sophisticated architectural models. This raises the question of how

such strategies should be expressed? Should they be hard-coded and added to a system by some kind of plug-in mechanism? Or can we enable computational scientists to inform these strategies in an intuitive manner? We are furthermore interested to determine the coverage such strategies can achieve, how to ensure recovery leading to correct system behaviour and to find out limits of such autonomic strategies.

Division of Responsibility. The final part of our research deals with opportunities for interaction between computational scientists and autonomic recovery strategies. There are two main elements to this. For the sake of accountability of autonomic recovery strategies, it will be necessary to make reports available to human users about any incidents and actions taken during a run of a workflow. By identifying failures that cannot be addressed automatically, it will furthermore become possible to determine when human users should be involved in decisions about which actions to take in order to handle such failures successfully. Autonomic strategies could then guide users in resolving issues and make their repository of actions available to be steered by users. The question here is in how far support for dividing responsibility between human users and autonomic mechanisms can be used to overcome any limitations of the latter.

We believe that the combination of informed autonomic recovery strategies and meaningful interaction with human users provides a promising avenue for resolving some, if not many, of the reliability challenges computational scientists are currently confronted with.

6 Related Work

There are a number of related efforts taking place at Cornell. Services to monitor system health in support of high-availability in mission-critical Web service applications have been proposed in [15]. Astrolabe [5] has been proposed as a monitoring standard applications could use to implement autonomic behaviour [16]. And finally, [17] discusses services for tracking of process group membership, failure detection and reaching consensus.

Our work differentiates itself from these efforts in various respects. Our failure investigation service does not aim to support process group semantics or achieve a global view of system state. We prefer to avoid the added cost and complexity of establishing a strong notion of consistency and are instead interested to determine the capabilities and limits of mechanisms built on a simple infrastructure that makes additional system information available on request. In cases where achieving consensus may be required, we will investigate the use of resolution schemes [18]. Defining our main target group to be computational scientists means that we cannot expect them to use the features of a monitoring service directly to implement autonomic behaviour in their workflows and that we must limit the necessary setup and configuration activities. Another difference is our interest in investigating how the coverage achieved by such recovery strategies can be increased through cooperation with human users.

An interesting first step in the context of scientific grid applications is represented by OPERA-G [19]. However, the autonomic behaviour OPERA-G was able to achieve is limited. We propose to develop autonomic recovery strategies based on a richer set of information.

The issue of accountability of autonomic computing mechanisms has been raised in [20] and [21].

7 Conclusion

In this paper we discussed the notion of global scientific BPEL workflows and the environment they operate in. The examples of typical failures of global workflows that occur in practice and their consequences provide an insight into the variety and complexity of failures. This helps to confirm our experience. Namely, that tackling the threats to a scientific workflow's reliability is a challenging task, even for experienced software engineers. Our brief examination of existing reliability mechanisms and language constructs lets us conclude that neither do the proposed techniques address the breadth of threats effectively nor do they provide an interface that allows for sufficiently simple interaction with computational scientists.

For solving many of the identified issues, we proposed a solution consisting of three main parts. We aim to build an environment that can make basic information about failures available in order to enable informed autonomic recovery strategies. Furthermore, we will investigate how to achieve useful interaction between human users and these mechanisms to overcome any limitations. The proposed research seems promising and its components raise a number of interesting questions which we look forward to addressing and examining more closely.

References

1. Emmerich, W., Butchart, B., Chen, L., Wassermann, B., Price, S.L.: Grid Service Orchestration using the Business Process Execution Language (BPEL). J Grid Comp. **3**(3-4) (2005) 283–304
2. Wassermann, B., Emmerich, W., Butchart, B., Cameron, N., Chen, L., Patel, J.: Sedna: A BPEL-based environment for visual scientific workflow modelling. In Taylor, I.J., Deelman, E., Gannon, D., Shields, M.S., eds.: Workflows for eScience - Scientific Workflows for Grids. Springer (2006) To appear.
3. Cardelli, L.: Wide Area Computation. In Widermann, J., van Emde Boas, P., Nielsen, M., eds.: In Proc. of the 26th Intl. Colloquium: Automata, Languages and Programming. Volume 1644 of LNCS., Springer (1999) 10–24
4. Fischer, M., Lynch, N., Paterson, M.: Impossibility of distributed consensus with one faulty process. JACM **32**(2) (1985) 374–382
5. Renesse, R., Birman, K., Vogels, W.: Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Tran. Comp. Sys **21**(2) (2003) 164–206

6. Lee, W., McGough, S., Newhouse, S., Darlington, J.: A standard based approach to job submission through web services. In Cox, S., ed.: Proc. of the UK e-Science All Hands Meeting, Nottingham, UK, EPSRC (2004) 901–905 ISBN 1-904425-21-6.
7. Barghouti, N.S., Kaiser, G.E.: Concurrency Control in Advanced Database Applications. ACM Computing Surveys **23**(3) (1991) 269–317
8. Hagen, C., Alonso, G.: Exception handling in workflow management systems. IEEE TSE **26**(10) (2000) 943–958
9. Mehrotra, S., Rastogi, R., Silberschatz, A., Korth, H.: A transaction model for multidatabase systems. In: Proc. of the 12th Intl.Conference on Distributed Computing Systems, IEEE CS Press (1992) 56–63
10. Zhang, A., Nodine, M., Bhargava, B., Bukhres, O.: Ensuring relaxed atomicity for flexible transactions in multidatabase systems. In: Proc. of the 1994 ACM SIGMOD Intl. Conference on Management of Data, New York, NY, USA, ACM Press (1994) 67–78
11. Butler, M., Ferreira, C., Ng, M.: Precise Modelling of Compensating Business Transactions and its Application to BPEL. Journal of Universal Computer Science **11**(5) (2005) 712–743
12. Greenfield, P., Fekete, A., Jang, J., Kuo, D.: Compensation is Not Enough [fault-handlng and compensation mechanism]. In: Proc. of the 7th IEEE Intl. Enterprise Distributed Object Computing Conference, Brisbane, Australia, IEEE CS Press (2003) 232–239
13. Evans, C., Chappell, D., Bunting, D., Tharakan, G., Shimamura, H., Durand, J., Mischkinsky, J., Nihei, K., Iwasa, K., Chapman, M., Shimamura, M., Kassem, N., Yamamoto, N., Kunisetty, S., Hashimoto, T., Rutt, T., Nomura, Y.: Web Services Reliability (WS-Reliability 1.0) (2003)
14. Ferris, C., ed.: Web Services Reliable Messaging Protocol (WS-ReliableMessaging). BEA Systems, IBM, Microsoft Corporation, TIBCO Software (2005)
15. Birman, K., Renesse, R., Vogels, W.: Adding high availability and autonomic behavior to web services. In: Proc. of the 26th Intl. Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society (2004) 17–26
16. Birman, K., Renesse, R., Vogels, W.: Navigating in the storm: Using Astrolabe to adaptively configure web services and their clients. Journal of Cluster Computing **9**(2) (2006) 127–139
17. Vogels, W.: Tracking Service Availability in Long Running Business Activities. In: LNCS, Intl. Conference on Service Oriented Computing. Volume 3628/2005. Springer (2003) 395–408
18. Kermarrec, A.M., Rowstron, A., Shapiro, M., Druschel, P.: The icecube approach to the reconciliation of divergent replicas. In: Proc. of the 20th annual ACM Symposium on Principles of Distributed Computing, New York, NY, USA, ACM Press (2001) 210–218
19. Bausch, W.: OPERA-G : a microkernel for computational grids. PhD thesis, ETH Zürich (2004)
20. Anderson, S., Hartswood, M., Procter, R., Rouncefield, M., Slack, R., Soutter, J., Voss, A.: Making autonomic computing systems accountable: the problem of human computer interaction. In: Proc. of the 14th Intl. Workshop on Database and Expert Systems. (2003) 718–724
21. Ibrahim, M.T., Telford, R., Dini, P., Lorenz, P., Vidovic, N., Anthony, R.: Self-adaptability and man-in-the-loop: A dilemma in autonomic computing systems. In: Proc. of the 15th Intl. Workshop on Database and Expert Systems Applications, Washington, DC, USA, IEEE Computer Society (2004) 722–729