Achieving Service Dependability Through Context-Awareness

Michael Schiffers
Munich Network Management Team
Department of Informatics
Ludwig-Maximilian-University
Oettingenstr. 67, 80538 Munich, Germany
michael.schiffers@ifi.lmu.de

Abstract

By large-scale services (LSS) we understand IT services that are being deployed over unbounded large-scale infrastructures. Typical examples may be found in e-commerce scenarios, in computational Grids, or in ubiquitous computing environments. The more such services will be composed on the fly from other services the more dependability becomes a challenge. Service dependability is about error processing and fault tolerance. There are operational similarities between error processing and context provisioning in ubiquitous computing.

In this paper we are exploring the beneficials of applying context-awareness to the dependable composition of services. Based on an e-commerce scenario we analyze the similarities between error processing and context provisioning, but we also show how the single phases of these processes translate into each other. We address how service dependability may benefit from context-awareness and we finally report on a prototypical implementation of a Web Services based context provisioning framework.

Keywords

context-awareness, context provisioning, dependability, service management, service composition, large-scale service

1. Introduction

In the past few years the popularity of large-scale Internet services such as email services, auction services, and search services has grown enormously. Their scalability and dependability requirements have led to service orientated system architectures that are typically provided over clusters of thousands of geographically distributed computers undergoing frequent reconfiguration and functionality changes [1]. Similarly, Grid computing is turning more and more into a mainstream technology for large-scale distributed resource sharing addressing the increasing demand for collaborative problem solving [2]. Other examples of large-scales ser-

*This work has partly been funded by the Bavarian Government under contract 1312/685 66.

vices (LSS) include Peer-to-Peer applications, vehicular ad-hoc services [3], or such services as envisioned in the Agentcities initiative [4].

By *Large-Scale Services* (LSS) we understand IT services that are being deployed over large-scale infrastructures composed of autonomous systems each belonging to and managed by independent organizations [5, 6]. Typically, the components are existing prior to the configuration of the LSS and are well established in providing useful services to their hosting organizations. In such systems change is no longer rare, rather it is a standard situation without prior notice. Consequently, the components of an LSS are capable of independent behavior and independent failing. Since a global view of the whole infrastructure is not possible, the interoperability between the respective components (belonging to different administrative domains) has to be determined by local conventions. Sometimes these systems are referred to as "unbounded" [6], "federated" [7], or "Systems of Systems" [5].

Dependability of a service is typically defined as that property of a service that makes the service justifiably trustable under all circumstances, even in hostile environments [8]. Thus, dependability encompasses availability, reliability, confidentiality, maintainability, manageability, integrity, privacy and safety. Providing dependable services over unbounded infrastructures raises questions of how to cope with dynamically changing user expectations, flexible infrastructures, domain specific SLAs, or dynamic management policies.

Treating the dependability of composite services (and of the composition process itself) not only requires an adequate presentation of critical information, but also the automatic chaining of services under erroneous conditions. These challenges resemble those being studied in context-aware research. We are interested in understanding these similarities and their limitation. This idea is to look at policies, SLAs, infrastructures, user locations, and the like as context information a service may adapt to.

Context-awareness presents a new paradigm in service provisioning in that context-aware services (CAS) automatically adapt their functionality to the service execution environment in a transparent way. Context is not just there, it has to be provided. This directly leads to the question of how to obtain it. Any necessary context information is derived from raw data delivered by various context sources, like sensors, and iteratively refined by different operators and providers.

Context information can be used to characterize the situation of a person, a place, or any other physical or abstract object, which is of relevance for a particular service [9]. Examples of context information are the user's current activity (working, sleeping, eating, etc.), her spatial environment (indoor, outdoor, at home, at work, etc.), or the technical capabilities of the mobile device she uses. A service is then regarded as context-aware if it uses context to adapt its behaviour to the user's task. For example, an instant messaging service will become context-aware if the message exchanges between its users (the buddies) are adapted to their current location, activities, and mood, and if the appearance of the service, e.g., the representation format of messages (text, pictures, audio, etc.), is adapted to the

capabilities of the respective user's mobile device and the transport protocols of the underlying network [10].

Our interest is in automatically providing and managing the dependable composition of dependable services from autonomous components. In this paper we address the issue of tolerating *service composition errors*. We propose to apply concepts from context-awareness to achieving this.

The rest of the paper is structured as follows: In sections 2.1 and 2.2 we will give a short introduction into the notions of dependability and context-awareness. In section 3 we introduce a large-scale e-commerce service and address the similarities between error processing and context provisioning. In section 4 we report on a prototypical implementation of a context provisioning framework. Before we conclude the paper in section 6 we relate our work to other contributions in section 5.

2. Basic Notions of Dependability and Context-Awareness

2.1 Dependability

Following [8], dependability is a fundamental property of any system addressing the reliance that can be placed on the services it delivers (see figure 1). The causal relationship between faults, errors, and failures, is a key dependability concept. The basic means for achieving dependability are fault tolerance, fault avoidance, fault removal and fault forecasting [8]. A *fault* is considered as the hypothesized cause of an error. An *error* is that part of a system state that may cause a subsequent failure. A *failure* occurs when a system service deviates from the behaviour expected by the user. *Fault tolerance* is a means for achieving dependability assuming that any system contains faults and is thus inevitably uncertain. *Error processing* typically consists of three steps: error detection, error diagnosis and error recovery. Error detection identifies an erroneous state in the system. Error diagnosis assesses the damage caused by the detected error, or the errors propagated before detection. Error recovery transforms a system state that contains errors into an error free state. Fault treatment consists of fault diagnosis and system repair [8].

A system is dependable to the extent to which its operation is free of failures. Then *dependability* can be defined as that property of a system such that allows placing justifiable trust on the service it delivers. A user is regarded as just another system which interacts with the former. Typical attributes of dependability are availability, reliability, safety, confidentiality, integrity, maintainability. Following [8] security is not considered as a dependability attribute as it involves a combination of availability, confidentiality, and integrity.

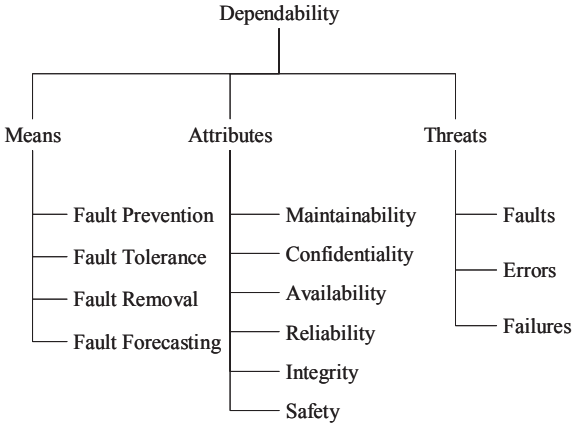

Figure 1: The dependability tree [8]

2.2 Context-Awareness

We call a service *context-aware* if it uses context when providing relevant information and/or services to the user, where relevancy depends on the user's task [9]. *Context* is regarded as any "information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves" [9]. This definition is very broad and lacks the reflection of context as an operational term: "something is context because of the way it is used in interpretation, not due to its inherent properties" [11]. Hence, context is not just there, it is created when needed. However, bear in mind that focusing on specific aspects – those that characterize a situation – implies some prior knowledge of which aspects are more significant than others.

Context-aware services (CAS) can be categorized into three classes: "presenting information and services", "executing a service", and "tagging captured data" [9]. The first class refers to services that either present context information to the user, or use context to propose appropriate selections of actions to the user. The second class describes services that trigger commands, or reconfigure the system on behalf of the user according to context changes. Attaching context information for later retrieval refers to services that tag captured data with relevant context information. For examples of these categories we refer to [12].

Designing and building CASs requires appropriate abstraction mechanisms, context models, and a generic way to provide the necessary context information for a service's functional adaptation. This process of acquiring context information is typically arranged along a value chain (see figure 2 and is called *context provisioning* [10, 13].

The process of "infusing" the context information into an adaptation mechanism and the adaption itself are sometimes referred to as *contextualization*.

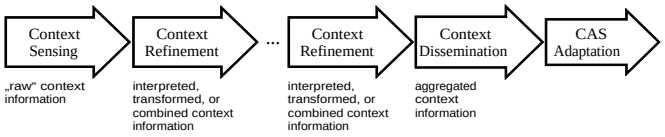

Figure 2: The context provisioning value chain

Notice that context refinement may comprises several very different transformation methods: the transformation between different formats of a representation (e.g., from GPS coordinates to street names and numbers), the augmentation with quality attributes [14], the fusion of several similar context sources [15] in order to overcome the inevitability of imprecision, or the mathematical computation of context information from other ones (e.g., calculating the distance between two objects).

3. Approaching Service Dependability by Context-Awareness

3.1 A Scenario

To provide a reference scenario for subsequent discussions we introduce *GSRM*, a hypothetical service for *Globally Sourcing of Raw Materials*. GSRM assists corporate procurement departments in globally sourcing of, bidding for, and purchasing of raw materials from appropriate spot markets. Typical application scenarios for GSRM would be the sourcing of raw chemicals, stainless steel, wheat, or even investment capital. The GSRM service is provided by HotHouse Corp. either on behalf of their customers as an outsourced service or they act as a Service Provider on request for subscribed customers. GSRM requires from the user a set of static profile parameters (e.g. authentication information) and service specific, mostly dynamic, parameters such as the user's actual preferences for price ranges, the required material quality, the shipping method preferences, time limits, or preferences for financial transactions.

As depicted in figure 3, in operating GSRM HotHouse offers the user to take advantage of other services provided by other corporations. Here are some examples: A *recommender service (RECOMMENDER)* rates trading transactions and recommends potential community partners for joint bidding (similar to the services considered in [16]. The temporary buying community itself could be established using a *community service (COMMUNITY)*. An *auction service (AUCTION)* such as those discussed in [17] could place bids in e-auctions. In case the supplier offers a set of similar products, an intelligent *catalog crawler service (CATALOG)* could walk through a supplier's catalog, probably based on pre-negotiated prices and terms and conditions. Finally, a *logistics service (LOGISTICS)* will be invoked in case the material has to be moved physically. An example of such a service can be derived from the Freightmixer scenario in [17].

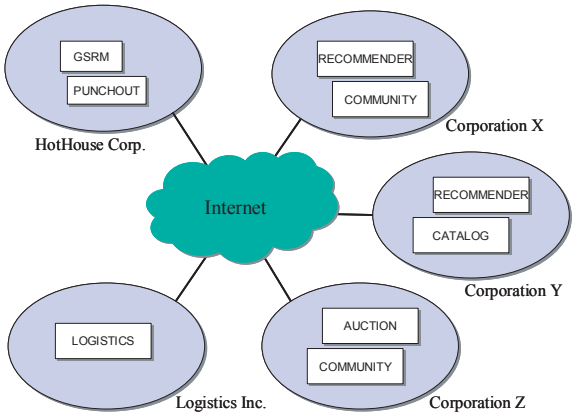

Figure 3: The GSRM domain structure

A typical (simplified) use case for GSRM is given in figure 4:

A user requests sourcing a tonnage of say stainless steel with a certain quality. GSRM first creates a co-operative buying community using recommendations, places bids on behalf of this community, selects the most appropriate one and finally places the order and arranges for shipping. GSRM also supports punchout [18] for "shopping cart" approval, purchase order, and the automatic generation of shipping documents, invoices, and credit notes.

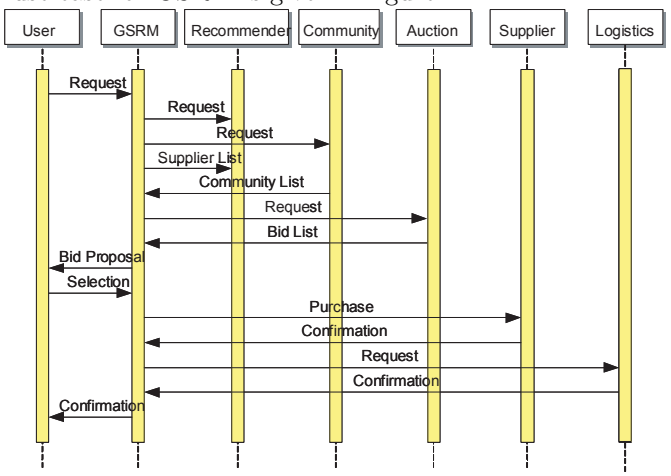

Figure 4: Example of a sequence diagram for GSRM

3.2 Similarities Between Value Chains

Referring back to section 2.1 and to [8], an important objective to attain service dependability is tolerance to both service internal faults and faults related to the composition of services. The phases of fault tolerance can generally be arranged along a dependability value chain as depicted in figure 5.

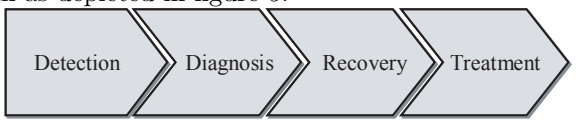

Figure 5: The dependability value chain

The similarities between the value chains in figures 5 and 2 are obvious as both are intrinsically process orientated, both need to acquire appropriate information, both need to analyze the information, and both need to draw adequate conclusions which may or may not result in adaptations. Context provisioning addresses exactly this generic value chain when supporting context-awareness (see section 2.2). Although both areas have developed their own methods and methodologies to address their specific challenges, we think it is worthwhile – because of the aforementioned similarities – to consider mutually leveraging the achievements in the respective other area. In the following sections we will therefore address tolerance to Service Composition Errors (SCE) as an example of exploring the suitability of approaching dependability issues by context-awareness. In doing so, we will follow the intrinsic value chain as explained above.

3.3 Tolerating Service Composition Faults

The dependability of large-scale services like GSRM requires a dependable composition of the diverse component services. For example, if GSRM is configured to use both a RECOMMENDER service and an AUCTION service, this composition must be able to cope with inconsistencies between the GSRM components. The challenge is not only to identify those components that are appropriate enough for the provisioning of a composed service, but also to make sure that this composition does not degrade the dependability of the components.

Service composition faults may occur when either trying to compose services from other services or when executing a composed service. Upon activation they generate Service Composition Errors (SCE). Composition errors are an undesired, though expected, circumstance which in most cases must be tolerated as they cannot be avoided. Consequently, SCEs need to be processed and treated appropriately, otherwise the overall service will fail if the error gets propagated.

Since we cannot expect a generic runtime error handling mechanism, it is necessary to classify SCEs and try to apply class specific detection and diagnosis mechanisms. A major challenge, however, is the capability to distinguish SCEs from other service errors and to distinguish composition problems or even intended behaviour from faults.

Typical SCE-classes range from architectural and protocol mismatches [5] to dependability related faults like incompatible reliability metrics and failure modes,

to wrong sub-service selection (e.g. referring to figure 3: which COMMUNITY service to select).

Detecting Composition Errors

Detecting SCEs means uniquely identifying an error as an SCE. This may require additional information concerning system states or service histories. This information is required at runtime. Notice that the identification of an SCE is not essential if respective provisions are made in the later stages of SCE tolerance. However, the later an error is identified as an SCE, the more costly and the more uncertain the respective processes of recovery and repair will be.

Consider the GSRM session from figure 6 and assume an SCE `AuctionFailed` for whichever reason. Identifying this error as an SCE needs at least some knowledge about the failing sub-service (e.g. AUCTION), the providing organization (e.g. corporation Z), the actual user preferences, the general GSRM service properties (e.g. QoS requirements), and contracted SLAs. Notice that even the user's location and device type may have an impact. Bearing in mind that GSRM is composed from autonomous services, collecting these information on-the-fly raises issues like completeness, speed, consistency, and privacy (e.g., corporation Z may not be interested to "open their books").

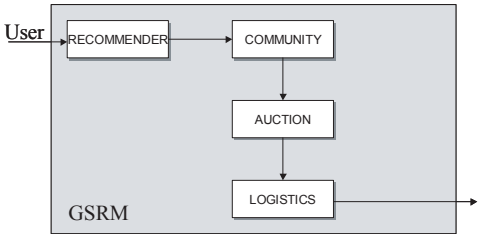

Figure 6: Example of a GSRM session

The first step in all context provisioning is context sensing capturing all relevant context information for all reference objects involved in determining a context information. As a simple example, the location of a reference object may be sensed by a GPS receiver. All actual settings of a reference object are retrieved from very different and heterogeneous context sources during context sensing. Context provisioning implies a goal-oriented co-operation of several actors in the roles of CAS Users, CAS Providers, Context Providers, and Context Owners. Figure 7 depicts this role model, an edge between two roles indicates a contractual or information flow relationship. For a more detailed discussion of this model we refer to [13].

Thus, applying context-awareness to SCE detection means identifying the right context information, identifying the actors and roles, and executing the appropriate interactions. The model automatically takes care of issues like privacy compliance, costs and billing, information quality assurance, and federated cooperation which are all more or less considered relevant for the detection phase.

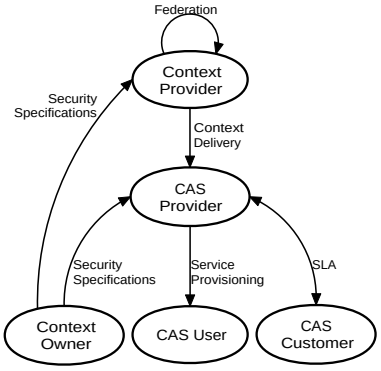

Figure 7: Context provisioning role model

Diagnosis of Composition Errors

The purpose of SCE diagnosis is to assess the damages caused by the detected SCE by identifying all the erroneous states of both the composed and the composing services. This typically requires the identification of the service's dependencies, the availability of all information necessary for damage assessment, and the availability of adequate refinement tools.

Consider again the GSRM session from figure 6 and assume again the SCE `AuctionFailed` exception which now, however, has been identified as an SCE. Notice that `AuctionFailed` is usually thrown after the bidding community has been established using the COMMUNITY service. Assessing the impact of `AuctionFailed` requires the ability to reason about this SCE and to derive from the exception all information adequate for deciding on the recovery strategies. A very simple example is given in figure 8 depicting how the (boolean) indicators `UpAndRunning` and `CommunityTrust` and the `Retry` list are derived from `AuctionFailed`.

Context refinement deals with the question of how to generate a concrete context information from those information that is already available from the previous acquisition phase or which must be captured additionally. For example, a distance could be directly sensed by a distance meter or it could be computed from two geographical coordinates. Context information is refined using a multi-step approach. In every step, previously sensed or computed information is taken, and refinement techniques like fusion, combination, deduction, filtering, transcoding, or inter- and extrapolation are applied to derive "higher level" context information with a certain quality. Conceptually we model the dependencies between context information as required by a CAS using a Petri Net like Context Composition Graph (CoCoGraph). Actually, figure 8 depicts such a graph. For more information on constructing and applying CoCoGraphs see [19]

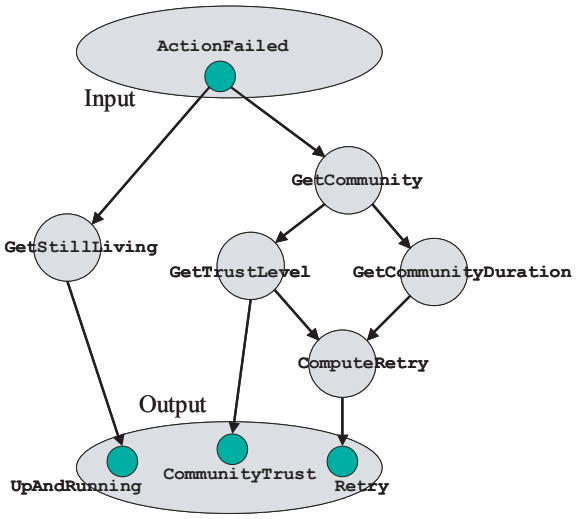

Figure 8: Example of error diagnosis

Once the necessary base information for a successful SCE diagnosis is available, concepts like CoCoGraphs and their construction resp. processing tools could be beneficial to all further diagnostic endeavours, provided the knowledge from where to capture the base information exists.

Recovery from Composition Errors

The purpose of SCE recovery is to replace an SCE state by an error-free state. The level of difficulty encountered for recovering from SCEs varies with the specific characteristics of an SCE and the service. Typically, there are backward recovery mechanisms and forward recovery mechanisms [20]. The former is more appropriate when dealing with service independence and general approaches for recovery like roll back. Due to the service characteristics we have in mind, however, rolling back may conflict with the un-ability of other services to roll back. This requires co-ordination. For SCEs caused by service specific problems forward recovery techniques like exception handling may be more appropriate. We want to mention though that in all those cases in which not enough information is available for supporting SCE tolerance, error recovery often becomes very complex.

Consider again the GSRM session from figure 6 and assume again the SCE `AuctionFailed` has been diagnosed. Assume further that backward recovery techniques have been applied and that GSRM has been – as a orchestrated action – rolled back. What is missing now in order to make GSRM dependable is an au-

tomatic reconfiguration of GSRM. This does not necessarily always mean a new service implementation, it could also mean replacing sub-services by alternatives (e.g. exchange the COMMUNITY from corporation Z by the one from corporation X).

Contextualization covers both the dissemination of context information and the transparent adaptation of the CAS accordingly. After the refinement process, context information must be delivered to CASs. Generally, context dissemination can be subdivided into a pull and a push mode. Using the pull mode, context information are requested during CAS usage. The pull based context dissemination can be further classified into a polling and a caching mode. In the former case, context refinement is just activated on demand, whereas in caching mode, it has been executed in a proactive manner. Whether pull based context dissemination runs in polling or caching mode, may depend on the update frequency of context information. Using the push mode, context information are automatically delivered if their values reach a predefined range. The push mode is useful to realize context-aware push services.

Translating these concepts to service dependability issues induces support for new surveillance techniques, proactive SLA monitoring, failure situation dependent proactive replication management, or in general proactive policy adjustments.

4. A Prototypical Framework for Context Provisioning

In order to practically explore the suitability of context-awareness to the provisioning of dependable services we developed and implemented as a first step a framework supporting the acquisition and refinement of context information according to the context provisioning value chain as per figure 2 [21]. The prototype is based on Web Services technologies. We implemented parts of the context provisioning role model and the interactions between the roles. An overall conceptual view is given in figure 9.

For the dynamic composition of context information we implemented the aforementioned CoCoGraphs [19]. After initializing the service specific CoCoGraph the dynamic runtime composition could be realized by having the graph control transitions fire which means actually the collection of the respective context information.

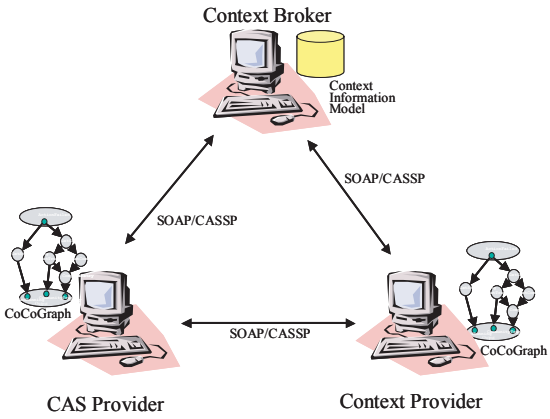

Figure 9: Context Provisioning Architecture

For implementing the interactions between the different roles we used a SOAP-based protocol CASSP (CAS Simple Protocol) which supports both the Context Information Model (CoIM) for modeling context information and locating Context Providers and CoCoGraphs.

Although we had to extend UDDI to a more flexible repository better serving our needs as far as search capabilities and semantics are concerned, we have found that the basic concepts and the distributed architecture of Web Services are in principle suitable for the kind of context provisioning we have in mind. The big overhead in Web Services protocols due to the XML-format, however, constraints the overall system performance and may require dedicated solutions. Additional concepts are necessary in particular for the dynamic discovery and composition of services, CoIM and CoCoGraph are two such concepts. The drawback is the need for a higher level protocol over SOAP which we provided with CASSP. For a full automation, a semantic solution based on ontology libraries is required. [22] proposes such a solution.

5. Related Work

Research on context-awareness is getting momentum. However, most of the work is still being done for experimental studies covering prototypical implementations in single provider environments. To our best knowledge, context-awareness has not been applied to the questions we addressed in this paper. For a good overview of the research in the field of context-awareness we refer to [9, 12, 23, 24].

Research on dependability on the other hand is not a new topic. A countless number of contributions has addressed lots of issues relating to the threats to, the attributes of, and the means by which dependability is attained (see e.g. [25, 20, 8, 26]. Lots of work has also been performed by the IEEE-CS Technical Committee on Fault-Tolerant Computing (`www.dependability.org/tc`) and IFIP's Working Group 10.4 Dependable Computing and Fault Tolerance (`www.dependability.org/wg10.4`). In spite these many contributions little work has been done in dependably composing large-scale services from autonomous com-

ponents while preserving their dependability. Nearly all approaches assume a prior knowledge of a correct system state or they assume a total management control within well-defined perimeters. The DSoS project (Dependable Systems of Systems [5]) focuses on challenges close to ours but neither do they cover any service management related issues, nor do they take into account role-specific issues as they emphasize on the design of the linking interfaces between component systems only. InfoSpect [27] on the other hand is an approach aiming at managing dependability in large-scale systems. However, they do not consider multi-provider scenarios and their logic language approach (based on Prolog) does not scale well when dealing with fast growing rule bases. Existing middleware approaches (e.g. CORBA) all have the disadvantage that their primary focus is not on dependability issues. Hence, they provide only very specialized solutions. Finally, some contributions from Grid community [28] are promising. However, the Grid community either assumes a central management authority or still fails in transparently adapting to sudden Grid changes. The applicability of Web Services architectures has also been discussed in [29].

6. Conclusion and Future Work

Understanding the concept of context and how to make use of it are central research challenges in the context community. In this paper we proposed to apply the concepts to achieve dependability of large-scale services crossing organizational boundaries. Based on an e-commerce scenario we analyzed dependability issues when composing emergent services from component services. We not only pointed out the similarities between error processing and context provisioning, but we also showed how the phases of these value chains translate into each other. We addressed how service dependability may benefit from context-awareness. We finally reported on a prototypical implementation of a Web Services based context provisioning framework.

Our next steps will focus on the extension of the work done so far with special emphasis on investigating the benefits of treating dependability and other QoS parameters as context information.

Acknowledgement

The author wishes to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of this paper. The MNM Team, directed by Professor Dr. Heinz-Gerd Hegering, is a group of researchers at the University of Munich, the Munich University of Technology, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Their web server is located at `http://wwwmnmteam.informatik.uni-muenchen.de`.

References

[1] David Oppenheimer and David A. Patterson. Architecture and Dependability of Large-Scale Internt Services. *IEEE Internet Computing*, pages 41–49, September/October 2002.

[2] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid Services for Distributed System Integration. *IEEE Computer*, pages 37–46, June 2002.

[3] Christian Cseh, Reinhold Eberhardt, and Walter Franz. Mobile Ad-Hoc Funknetze für die Fahrzeug-Fahrzeug-Kommunikation. In Michael Weber and Frank Kargl, editors, *Mobile Ad-Hoc Netzwerke: Proceedings of the 1. Deutscher Workshop über Mobile Ad-Hoc Netzwerke (WMAN 2002)*, volume P-11 of *Lecture Notes in Informatics (LNI)*, pages 109–120, Bonn, March 2002. Gesellschaft für Informatik (GI). (in German).

[4] S. Willmott, J. Dale, B. Burg, P. Charlton, and P. O'Brien. Agentcities: A Worldwide Open Agent Network. *The Agentlink Newsletter*, (8):13–15, November 2001.

[5] DSoS Project. Final Version of DSoS Conceptual Model. Technical Report CS-TR-782, University of Newcastle upon Tyne, April 2003.

[6] R. Ellison, D. Fisher, R. Linger, H. Lipson, T. Longstaff, and N. Mead. Survivable Network Systems: An Emerging Discipline. Technical Report CMU/SEI-97-TR-013, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA., USA, 1997, revised 1999.

[7] Gabrijela Dreo Rodošek. *A Framework for IT Service Management.* Habilitationsschrift, Ludwig-Maximilians-University, Munich, Germany, 2002.

[8] A. Avižienis, J. Laprie, and B. Randell. Fundamental Concepts of Dependability. Research Report N01145, LAAS-CNRS, Toulouse, France, April 2001.

[9] A.K. Dey. *Architectural Support for Building Context-Aware Applications.* PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[10] I. Hochstatter, A. Küpper, M. Schiffers, and L. Köthner. Context Provisioning In Cellular Networks. In *Proceedings of the 8th International Workshop on Mobile Multimedia Communications (MoMuc 2003)*, Munich, Germany, 2003.

[11] Terry Winograd. Architectures for Context. *Human-Computer-Interaction*, 16(2), December 2001.

[12] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dartmouth, USA, November 2000.

[13] H.G. Hegering, A. Küpper, C. Linnhoff-Popien, and H. Reiser. Management Challenges of Context-Aware Services in Ubiquitous Environments. Proceedings of the 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSCOM 2003), Heidelberg, Germany 2003.

[14] T. Buchholz, A. Küpper, and M. Schiffers. Quality of Context: What It Is And Why We Need It. In *Proceedings of the 10th Workshop of the HP OpenView University Association: HPOVUA'03*, Geneva, Switzerland, July 2003.

[15] Guanling Chen and David Kotz. Solar: Towards a Flexible and Scalable Data-Fusion Infrastructure for Ubiquitous Computing. In *UbiTools Workshop at UbiComp*, 2001.

[16] B.M. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender Systems for Large-Scale E-Commerce: Scalable Neighborhood Formation Using Clustering. In *Proceedings of the Fifth International Conference on Computer and Information Technology (ICCIT 2002)*, East Western University, Bangladesh, December 2002.

[17] Chris Preist, Andrew Byde, Claudio Bartolini, and Giacomo Piccinelli. Towards Agent-Based Service Composition Through Negotiation in Multiple Auctions. Technical Report HPL-2001-71R1, HP Laboratories Bristol, Bristol, UK, May 2001.

[18] D. M. Dias, S. L. Palmer, J. T. Rayfield, H. H. Shaikh, and T. K. Sreeram. E-commerce Interoperability with IBM's WebSphere Commerce Products. *IBM Systems Journal*, 41(2):272–286, 2002.

[19] M. Krause and I. Hochstatter. Strategies for On-the-Fly Composition of Context Information Services. In *Proceedings of the 11th Workshop of the HP OpenView University Association: HPOVUA'04*, Paris, France, June 2004.

[20] DSoS Project. State of the Art Survey. Technical Report CS-TR-708, University of Newcastle upon Tyne, April 2000.

[21] Yingfan Lei. An Architecture for Context Provisioning Over Web Services. Masters thesis, Ludwig-Maximilian University, Munich, Germany, 2003.

[22] Thomas Strang, Claudia Linnhoff-Popien, and Matthias Roeckl. Highlevel Service Handover through a Contextual Framework. In *Proceedings of the 8th International Workshop on Mobile Multimedia Communications (MoMuc 2003)*, Munich, Germany, 2003.

[23] Jason I. Hong and James A. Landay. An Infrastructure Approach to Context-Aware Computing. 16:287–303, 2001.

[24] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling Context Information in Pervasive Computing Systems. In *Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002)*, volume 2414 of *Lecture Notes in Computer Science*, pages 169–180, Zurich, Switzerland, 2002. Springer-Verlag.

[25] J.C. Laprie, A. Avižienis, and H. Kopetz. *Dependability: Basic Concepts and Terminology*. Springer-Verlag, New York, 1992.

[26] Claudio Basile, Marc-Olivier Killijian, and David Powell. A Survey of Dependability Issues in Mobile Wireless Networks. Technical report, LAAS CNRS, Toulouse, France, 2003.

[27] Timothy Roscoe, Richard Mortier, Paul Jardetzky, and Steven Hand. InfoSpect: Using a Logic Language for System Health Monitoring in Distributed Systems. In *Proceedings of the 10th ACM SIGOPS European Workshop*, Saint-Emilion, France, 2002.

[28] Ian Foster, Carl Kesselmann, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[29] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy. Dependability in the Web Service Architecture. In Rogério de Lemos, Cristina Gacek, and Alexander Romanovsky, editors, *Proceedings of the Workshop on Architecting Dependable Systems*, Orlando, USA, 2002. available from `http://www.cs.kent.ac.uk/events/conf/2002/wads/proceedingsW12.pdf`.