Software Engineering Institute

Proceedings of the Fourth International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2010)

Grace A. Lewis
Dennis B. Smith
Kostas Kontogiannis

September 2011

SPECIAL REPORT
CMU/SEI-2011-SR-008
ISBN: 0-9786956-8-2

Research, Technology, and System Solutions Program

http://www.sei.cmu.edu

Carnegie Mellon

This material is based upon work funded and supported by the United States Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

Table of Contents

List of Figures

List of Tables

Acknowledgments

We would like to thank many people for making this workshop a success.

First, we thank the authors and presenters who shared their work with us:

- Andrei Solomon and Marin Litoiu (York University, Canada)
- Hamzeh Zawawy (University of Waterloo, Canada)
- John Mylopoulos (University of Toronto, Canada)
- Serge Mankovskii (CA Technologies, Canada)
- Harry M. Sneed (ANECON GmbH, Austria)
- Andreas Metzger and Eric Schmeiders (University of Duisburg-Essen, Germany)
- Cinzia Cappiello, Elisabetta Di Nitto, and Barbara Pernici (Politecnico di Milano, Italy)
- Raman Kazhamiakin and Marco Pistore (FBK-Irst, Italy)
- Mantis Cheng, Priyanka Gupta, Hausi Müller, Atousa Pahlevan, Norha Villegas, and Qin Zhu (University of Victoria, Canada)

We thank the workshop attendees who contributed to the lively discussion:

- Amir Aryani (RMIT University, Australia)
- Marin Litoiu (York University, Canada)
- Mircea Lungu (University of Bern, Switzerland)
- Mehdi Mirzaaghaei (University of Duisburg-Essen, Germany)
- Hausi A. Müller (University of Victoria, Canada)
- Komondoor V. Raghavan (Indian Institute of Science, India)
- Alexander Serebrenik (Eindhoven University of Technology, Netherlands)
- Harry Sneed (ANECON GmbH, Austria)
- Stephen Thomas (Queens University, Canada)
- Scott Tilley (Florida Institute of Technology, USA)
- Norha Villegas (University of Victoria, Canada)
- Ken Wong (University of Alberta, Canada)
- Carl Worms (Credit Suisse, Switzerland)
- Hamzeh Zawawy (University of Waterloo, Canada)

We thank our program committee, who made sure we had high-quality papers to frame our workshop:

- Massimiliano Di Penta (University of Sannio, Italy)
- Liam O'Brien (National ICT, Australia)
- Mike Smit (University of Alberta, Canada)

- Harry Sneed (ANECON GmbH, Austria)
- Scott Tilley (Florida Institute of Technology, USA)
- Norman Wilde (University of West Florida, USA)

Finally, we thank the International Conference on Software Maintenance (ICSM) 2010 organizers for all their help, especially Radu Marinescu and Marius Minea from the "Politehnica" University of Timişoara.

Abstract

The Fourth International Workshop on Maintenance and Evolution of Service-Oriented Systems (MESOA 2010), organized by members of the Carnegie Mellon Software Engineering Institute's technical staff, was held at the 26th International Conference on Software Maintenance (ICSM 2010) in Timişoara, Romania, on September 17, 2010. The goal for MESOA 2010 was to share current research efforts and discuss emerging technologies in the maintenance and evolution of service-oriented systems. A second goal of the workshop was to identify areas of future work needed to address existing gaps and problems in the taxonomy of research topics in service-oriented architecture (SOA). This report summarizes the workshop and includes the accepted papers that were the basis for the presentations given during the workshop. Topics include using simulation models to evolve business processes, a requirements-driven framework for root cause analysis in SOA environments, SOA integration as an alternative to source migration, proactive adaptation as illustrated by the S-Cube service life cycle, a dynamic framework for quality web-service discovery, a characterization of policies that govern SOAs, and context-driven adaptive monitoring for supporting SOA governance. The report concludes with highlights from the discussions among workshop attendees.

1 Workshop Introduction

The Software Engineering Institute (SEI) started developing a service-oriented architecture (SOA) research agenda in 2007, based on an ideal life cycle for service-oriented systems that emphasizes the relationship between business strategy and SOA strategy. The core of the agenda is a taxonomy of research topics where new, more, or different research is needed to support both short-term and long-term strategic SOA adoption [1].

The following sections present a summary of the SOA research agenda and describe how the Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA) has contributed to the evolution of the topic of maintenance and evolution during the past three years.

1.1 SOA Research Agenda

In an ideal setting for service-orientation adoption, an organization will develop a service strategy that takes into account the organization's business drivers, context, and application domain. In order to accomplish the service strategy, the organization must generate plans to achieve the given goals and objectives. The execution of these plans requires business, engineering, and operations decisions, taking cross-cutting concerns into consideration, such as governance, security, risk management, social and legal issues, and training and education. All of these decisions take place within the service-oriented systems life cycle.

1.1.1 Service-Oriented Systems Life Cycle

The life cycle of service-oriented systems recognizes that SOA adoption requires an iterative approach to systems development that reflects the strong link between business strategy and development strategy. SOA adoption is not "all or nothing"; one of the benefits of SOA adoption is that systems and system components can be deployed gradually. The main differences between SOA adoption and other iterative development frameworks, such as the IBM Rational Unified Process (RUP), are SOA adoption's emphasis on activities to establish and analyze the relationship with business goals at the beginning of the cycle, its emphasis on evaluation at the end of the cycle, and the specification and review of business objectives at the end of the cycle so that the requirements for each iteration follow business objectives.

1.1.2 Taxonomy of SOA Research Topics

The development of a service-oriented system requires business, engineering, and operations decisions as well as cross-cutting decisions. The taxonomy of research areas, shown in Figure 1-1, is divided into topics pertaining to these decisions. Each research area contains a set of research topics where new, different, or additional research is needed to support a strategic approach to service-oriented systems development. Each research topic is analyzed in terms of (1) rationale—why it is an important research issue; (2) current efforts; and (3) challenges and gaps. The goal for the research agenda is therefore to identify topics for industrial or academic research that can fill these challenges and gaps and that make a difference in strategic SOA adoption.

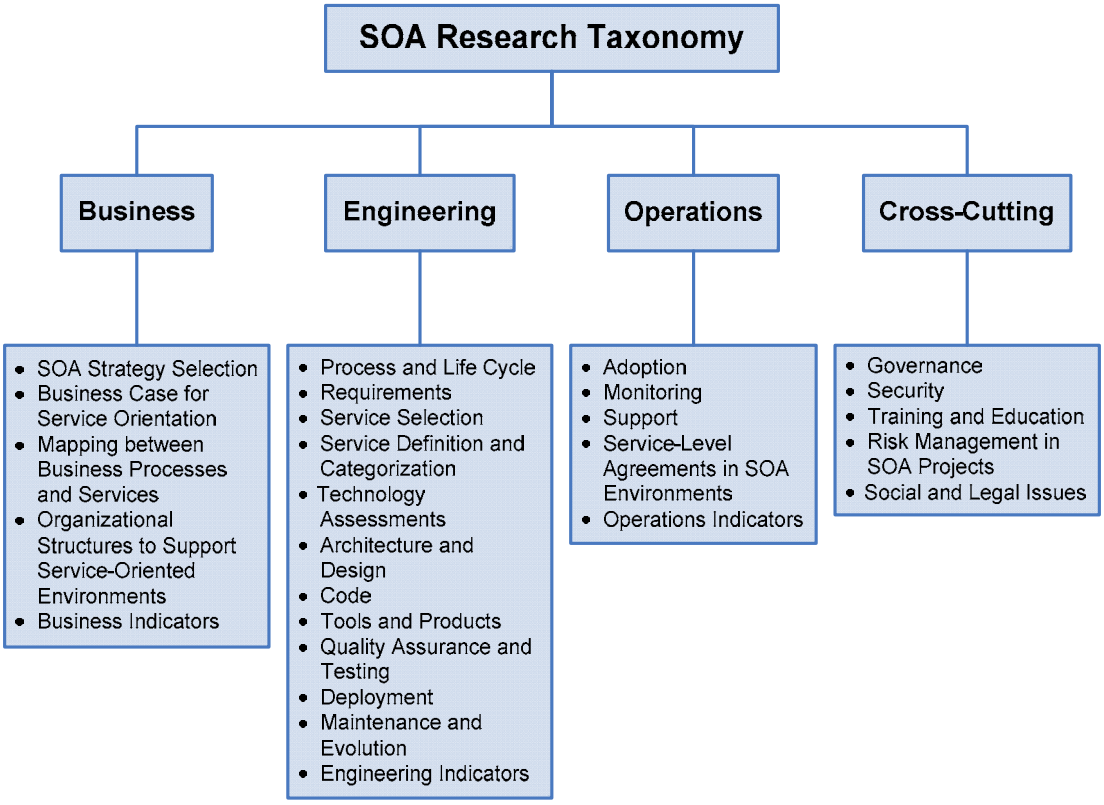


Figure 1-1: SOA Research Taxonomy

1.2 Evolution of the SOA Research Agenda through MESOA

Service-oriented systems are different from traditional systems. These differences include (1) the diversity of service consumers and service providers; (2) shorter software release cycles, because of the ability to rapidly adapt to changing business needs; and (3) the potential to leverage legacy investments with potentially minimal change to existing systems.

Therefore, an important question remains: What does maintenance and evolution look like in this dynamic, heterogeneous, and potentially distributed development and maintenance environment? *Maintenance and Evolution* is a research area under *Engineering*, as shown in Figure 1-1. The area has evolved as service orientation has become better understood and as more service-oriented systems are deployed and now have to be maintained and evolved [1, 2, 3, 4, 5]. The research topics currently under investigation in the *Maintenance and Evolution* research area are presented in Figure 1-2 and are summarized in the following sections.

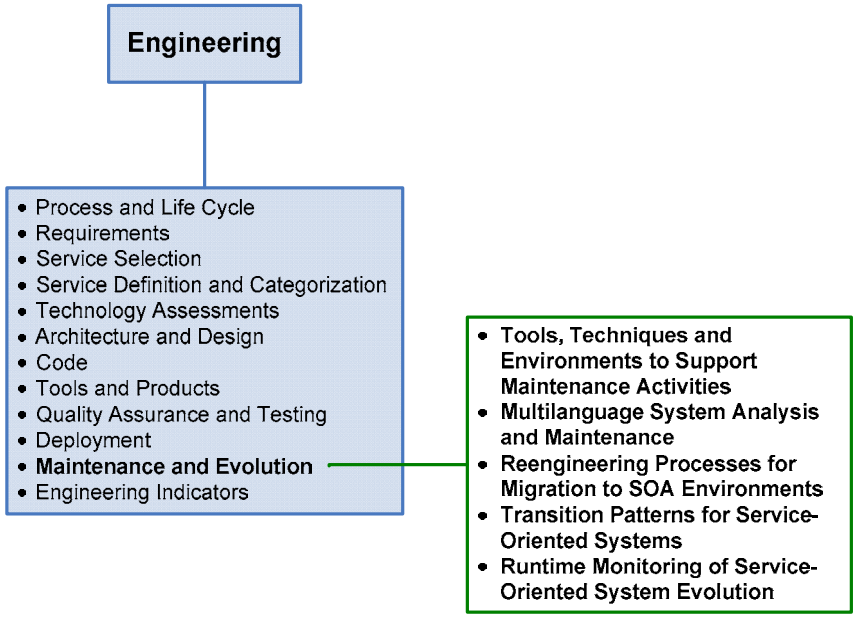


Figure 1-2: Research Topics in Maintenance and Evolution of Service-Oriented Systems

1.2.1 Tools, Techniques, and Environments to Support Maintenance Activities

The development of specialized methods and tools to support the maintenance and evolution of large service-oriented systems is in the early stages. Current efforts seem to indicate that maintenance activities for service-oriented systems are not that different from maintenance activities in traditional systems. However, we are still in the stage where most service-oriented systems are deployed for internal integration and where there is still some control over deployed services.

Current management practices will have to change with the emergence of a market for third-party services and the deployment of more service-oriented systems that cross organizational boundaries. These practices include maintenance processes, change impact analysis, change management and version control, and organizational structures and roles.

During MESOA 2007, Harry Sneed (ANECON GmbH, Austria) contributed a paper on testing titled "Testing a Service-Oriented Architecture: Installing Order in a Chaotic World." This short paper outlines the problems involved with demonstrating correctness in a complex, nondeterministic environment, such as a service-oriented system. The paper emphasizes the interaction between dynamic business processes and static web services and points out that testing a service-oriented system will be an endless task with countless uncertainties. There will never be a fully correct service-oriented system because it will evolve faster than we can demonstrate its correctness. Therefore, testing must be limited to achieving possible goals, such as constraining the effects of errors or validating essential requirements. Test coverage will mean something quite different for the web services than for the processes that use them.

1.2.2 Multilanguage System Analysis and Maintenance

Most research in this area is limited to small projects and a small number of languages, which is a problem for an environment that promotes platform independence.

In the case of third-party service providers, it is rarely possible to obtain direct access to source code. If that is the case, an important area of research is identification of the type of information that service providers should expose to service consumers who wish to do code analysis on invoked services and research on the tools to support the process.

During MESOA 2008, Sneed contributed a paper titled "Static Analysis and Measurement of Web Service Interfaces." This paper states that the critical hinge in an SOA is the web service interface, as defined in the Web Services Description Language (WSDL). This interface must be very high quality, and the paper describes the role of static analysis and interface metrics in helping to ensure that quality. First, the paper defines a set of rules and metrics for assessing web service interfaces. Next, it describes a tool for statically analyzing the interface definitions. Finally, it presents a conclusion about the effectiveness of static analysis of web service interfaces.

1.2.3 Reengineering Processes for Migration to SOA Environments

The ideal reengineering process would implement the SOA-migration "horseshoe" that starts from implemented legacy code, develops legacy software and legacy enterprise models, and then transforms the models into services and components [6]. Current techniques and tools implement portions of the horseshoe, but not the full horseshoe. A necessary research topic is the development of concrete processes that implement the horseshoe and tools (or suites of tools) to support the process.

A significant challenge for reengineering is mining legacy code for services that have business value. This requires research in areas such as

- tools and techniques for analyzing large source code bases to discover code that has business value
- metrics for "wrapability" and business value to determine reusability [7]
- the application of feature-extraction techniques to service identification [7]

There have been multiple MESOA contributions in this area:

- MESOA 2007
  - Lerina Aversano, Massimiliano Di Penta, and Ciro Palumbo (University of Sannio, Italy) contributed a paper titled "Towards Service Identification from Legacy Code: An Integrated Approach." They proposed an architecture and technique based on library schema extraction and feature extraction to identify models from the legacy application, build the service description from the extracted features, and identify the available services or components that match the service description. Since then, the authors have published other work related to the use of business process reengineering (BPR) and workflow analysis to support the maintenance and evolution of service-oriented systems in a project called METAMORPHOS (MEthods and Tools for migrAting software systeMs towards web and service Oriented aRchitectures: exPerimental evaluation, usability, and tecHnOlogy tranSfer) and an empirical comparison of methods to support quality-of-service-aware service selection [8, 9].
  - Oleksandr Panchenko (Hasso Platner Institute, Germany) contributed a paper titled "Factors Impacting Concept Location in Service-Oriented Enterprise Software." The

paper states that software maintenance is a difficult activity. Searching for source code that implements a certain real-world concept is an essential part of maintenance and is called "concept location." The difficulty of concept location in service-oriented software systems is still unknown but is expected to be seriously underestimated. The paper discusses factors that could either complicate or support concept location in a service-oriented enterprise software system. These thoughts are based on experiments and theoretical speculations. Since then, the author has published related work, including a paper on extending service scope while preserving backward compatibility [10].

  - Ned Chapin (InfoSci, USA) contributed a paper titled "Value-Focused Research Directions on SOA Applications." The paper stated that during the past seven years, the definition of SOA has changed, yet formal research on what has been gained or lost by those changes in software design, implementation, and maintenance has been largely neglected. The author also contributed a paper to MESOA 2009 on the maintenance characteristics of service-oriented systems [5].

- MESOA 2008
  - Andrea de Lucia (University of Salerno, Italy) contributed a presentation titled "Migration of Legacy Systems to the Web: Challenges for Migration to SOA." Since then, the author has published related work on METAMORPHOS, evaluating legacy system migration technologies through empirical studies, and a survey on software migration projects in Italian industry [8, 11, 12].
- MESOA 2009
  - Sedigheh Khoshnevis, Pooyan Jamshidi, Reza Teimourzadegan, Ali Nikravesh, Alireza Khoshkbarforushha, and Fereidoun Shams (Shahid Beheshti University, Iran) contributed a paper titled "A Method for Automating Model Evolution of Service-Oriented Systems." The paper states that a service model is the key work product of a service-oriented solution life cycle, in which localizing the effects of business changes could reduce the cost of evolution and maintenance. It proposes an automated approach for service model evolution, named the Automated Service Model Evolution Method (ASMEM), and describes a preliminary implementation of the Automated Service-Oriented Modeling Tool (ASOM-Tool), which is developed to supply automation and traceability features for modeling and maintaining service-oriented solutions [5].

1.2.4 Transition Patterns for Service-Oriented Systems

Models for analyzing the total cost of service-oriented systems are urgently needed; however, systematic work in this area is in the very early stages. In 2009, Grace Lewis reported on preliminary work that she was conducting that addressed total modernization costs, including costs for development, deployment and infrastructure, data migration, and transition. Transition in this context is the incremental migration of a legacy system to a service-oriented environment in which legacy code and new code coexist until the end of the last increment. The goal of this research is to project an optimal number of increments and minimize the throwaway costs of increments.

1.2.5 Runtime Monitoring of Service-Oriented System Evolution

This topic was added to the research agenda over the past year as a result of discussions from MESOA 2008 and 2009.

One benefit of SOA adoption is the agility to create and change systems as the business changes. Because service-oriented systems are implemented as a collection of interacting services and systems, collaboration usually takes the form of workflows that enact specific business processes. However, it is difficult to monitor the satisfaction of requirements, reference architectures, service-level agreements, and business goals as business processes and services change.

To address this problem, service-oriented systems should incorporate self-adaptation and self-management mechanisms. Given the strong links between business strategy and SOA, runtime monitoring is a best practice to verify whether business goals are being met. Current SOA infrastructures provide the capability to define metrics and collect data. The real challenges are to define the most relevant data to measure and monitor and to determine which adaptation strategies to execute.

The MESOA workshop has featured several contributions from Hausi Müller and his research group (University of Victoria, Canada) related to this topic:

- MESOA 2007: "Implications of Autonomic Computing for SOA Maintenance and Evolution"
- MESOA 2008: "Runtime Monitoring of Service-Oriented Systems: Implications for Maintenance and Evolution"
- MESOA 2009: "SOA Governance Optimizes the Business and Evolution of Service-Oriented Systems," with Priyanka Gupta, Ron Desmarais, Alexey Rudkovskiy, Norha Milena Villegas, Qin Zhu (University of Victoria, Canada), and Leho Nigul (IBM Center for Advanced Studies, Canada). This paper surveys leading governance methodologies designed by different organizations and aligns them along three critical pillars of SOA governance—people, policies, and processes. It then examines how these governance pillars optimize the business and evolution of service-oriented systems. At the design time of an SOA initiative, the people pillar clearly dominates the other two, while at runtime, there is a symbiotic relationship among the three SOA governance components. The goals of runtime SOA governance are to control, monitor, and adapt these components and their relationships to optimize business and evolution [5].

References

[1] Lewis, Grace; Kontogiannis, Kostas; and Smith, Dennis. A Research Agenda for Service-Oriented Architecture (SOA): Maintenance and Evolution of Service-Oriented Systems. Pittsburgh: Carnegie Mellon University, CMU/SEI-2010-TN-003, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tn003.cfm

[2] Kontogiannis, Kostas, et al. "The Landscape of Service-Oriented Systems: A Research Perspective." Proceedings of the International Workshop on Systems Development in SOA

Environments, International Conference on Software Engineering (ICSE), Minneapolis, 2007.

[3] Kontogiannis, Kostas; Lewis, Grace; and Smith, Dennis. "A Research Agenda for Service-Oriented Architecture." Proceedings of the Second International Workshop on Systems Development in SOA Environments, Leipzig, Germany. Association for Computing Machinery, 2008.

[4] Kontogiannis, Kostas; Lewis, Grace; and Smith, Dennis. "The Landscape of Service-Oriented Systems: A Research Perspective for Maintenance and Reengineering." Proceedings of the 11th International Workshop on Service-Oriented Architecture Maintenance and Reengineering (SOAM), Amsterdam, the Netherlands. IEEE Computer Society Press, 2007.

[5] Lewis, Grace; Smith, Dennis; Chapin, Ned; and Kontogiannis, Kostas. Proceedings of the 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2009). Pittsburgh: Carnegie Mellon University, CMU/SEI-2010-SR-004, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10sr004.cfm

[6] Winter, A.; and Ziemann, J. "Model-Based Migration to Service-Oriented Architectures." Proceedings of the International Workshop on SOA Maintenance Evolution, Amsterdam, the Netherlands. IEEE Computer Society Press, 2007.

[7] Sneed, H. "Migrating to Web Services: A Research Framework." Proceedings of the International Workshop on SOA Maintenance Evolution (SOAM 2007), 11th European Conference on Software Maintenance and Reengineering (CSMR 2007), Amsterdam, the Netherlands. IEEE Computer Society Press, 2007.

[8] De Lucia, A.; Di Penta, M.; Lanubile, F.; and Torchiano, M. "METAMORPHOS: MEthods and Tools for migrAting software systeMs towards web and service Oriented aRchitectures: exPerimental evaluation, usability, and tecHnOlogy tranSfer." Software Maintenance and Reengineering (CSMR 2009), Kaiserslautern, Germany. IEEE Computer Society Press, 2009, 301–304.

[9] Cavallo, Bice; Di Penta, Massimiliano; and Canfora, Gerardo. "An Empirical Comparison of Methods to Support QoS-Aware Service Selection." Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS '10), Cape Town, South Africa. ACM, 2010, 64–70.

[10] Borovskiy, Vadym; Muller, Jurgen; Panchenko, Oleksandr; and Zeier, Alexander. "Generic Web Services: Extending Service Scope while Preserving Backwards Compatibility." ICSOFT 2009: Proceedings of the 4th International Conference on Software and Data Technologies, vol. 1, Sofia, Bulgaria, 2009.

[11] Colosimo, Massimo; De Lucia, Andrea; Scanniello, Giuseppe; and Tortora, Genoveffa. "Evaluating Legacy System Migration Technologies through Empirical Studies." Inf. Softw. Technol. 51, 2 (February 2009), 433–447.

[12] Torchiano, M.; Di Penta, M.; Ricca, F.; De Lucia, A.; and Lanubile, F. “Software Migration Projects in Italian Industry: Preliminary Results from a State of the Practice Survey.” 23rd IEEE/ACM International Conference on Automated Software Engineering Workshops, Sept. 2008, pp. 35–42.

2 Using Simulation Models to Evolve Business Processes

Andrei Solomon, York University, Canada (asolomon@cse.yorku.ca)
Marin Litoiu, York University, Canada (mlitoiu@yorku.ca)

Abstract

This paper investigates the evolution of business processes triggered by changing operating conditions, company policies, and deterioration of key performance indicators (KPIs). We monitor the execution of business processes, build and tune the performance model of the process under study, and then use it to investigate future changes or improvements.

2.1 Introduction

A business process is a series of coordinated tasks or activities, performed by either people or equipment, that transform one or more inputs into a defined set of results that are of value to a customer. Business process management (BPM) is a discipline whose goal is to systematically organize, manage, analyze, and optimize an organization's business processes. Software tools supporting BPM throughout the business process improvement life cycle are called business process management systems (BPMS).

The BPM life cycle consists of four phases, as shown in Figure 2-1:

- *Model*. In this phase, an "as-is" or "to-be" business process is defined, designed, graphically modeled, and simulated in a BPMS. Graphical standards in this stage (e.g., business process modeling and notation [BPMN], Unified Modeling Language activity diagrams) allow developers, analysts, and business managers to express and understand business processes and their possible flows in a diagrammatic way at a higher level [6].
- *Assemble*. At this stage, the model designed in the previous stage is assembled, developed, configured, and tested on the underlying system infrastructure.
- *Deploy*. The *Deploy* phase is where the modeled business process is deployed on an actual BPMS engine and uses the configuration from the *Assemble* phase.
- *Manage*. After being deployed, business processes are monitored and analyzed with appropriate tools for potential bottlenecks in order to prepare the grounds for the new *Model* phase adaptations in response to changes.

A business process has to achieve certain key performance indicators (KPIs) to satisfy company business objectives. Although the process is designed and deployed to do so, based on the initial assumptions and requirements, changing operating conditions trigger the need for further evolution of the process.

The key to a successful evolution is an understanding of the effect of operating conditions on the qualities of the business process. Tracing the execution of instances and analyzing the data can allow for quantitative evaluation of different change decisions [7, 9]. Simulation is appropriate

when trying to gain insight into future situations and when experimentation is too expensive or even dangerous.

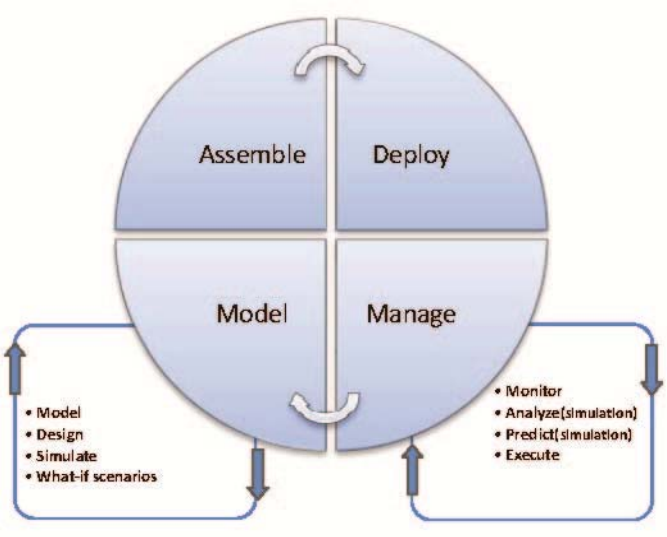


Figure 2-1: Model-Assemble-Deploy-Manage Life Cycle of a Business Process

In this paper, we propose a feedback-based evolution architecture, and we show how to use a runtime-tuned simulation model to assist the evolution of business processes. The simulation model uses an estimator and an online monitor so that it accurately matches the execution of the real process. A simulation model, in sync with the underlying process and integrated with modeling tools, will speed the adaptation through the process life cycle and enable autonomic computing activities [11]. It automates and speeds up the cycle time between the *Manage* and the *Model* life-cycle stages by enabling the delivery of runtime data and statistics to the process modeling environment.

The remainder of the paper is structured as follows. Section 2.2 introduces the key technologies used in our study, BPMN and business process execution language (BPEL); Section 2.3 presents the *model*, *assemble*, *deploy*, and *manage* phases of a business process; Section 2.4 presents the proposed feedback-based evolution architecture; Section 2.5 describes a case study and preliminary results of the new architecture; and Section 2.6 shows a possible automatic optimization strategy using the proposed architecture. Finally, conclusions and further work are presented in Section 2.7.

2.2 Background and Related Work

BPEL for web services is an OASIS standard executable language for business processes [2]. It allows composition of web services and is thus the top-down approach to SOA implementation. Involved web services do not know that they are involved in a composition and that they are a part of a higher business process. BPEL can currently express conditional behavior and loops, declare variables, copy and assign values, and define fault handlers, but it is unsuitable for business process analysts and designers because it forces users to think in terms of low-level BPEL constructs.

As a core enabler of BPM, BPMN provides the graphical notation for specifying business processes in business process diagrams (BPDs). It has gained much acceptance in industry because of the different levels of abstraction it provides, but no tools can execute BPMN directly. Therefore, many of its implementations can be mapped directly to BPEL code, which is supported by several execution platforms.

2.3 BPM Cycle

2.3.1 Modeling the Business Process—Model Phase

Successful reengineering comes with a thorough understanding of the existing process performance under different conditions together with an accurate representation of it. Before implementation, it is important to determine the business items that get transformed during business operations, the resources required to complete each activity, their availability timetables, and their process flow. For instance, Figure 2-2 presents a simple BPD where three web services are used: "Process," "Disburse," and "Reject." An "Application" business item is passed on each connector line, directing the flow from one task to another. Business measures describe instance or aggregated metrics about the process that are important for business success. Instance metrics are a collection of measurements resulting from each execution of the process. Aggregate metrics are calculated across multiple instances using a function of interest (e.g., average, maximum, minimum, sum, or count) to get other useful information about the process.

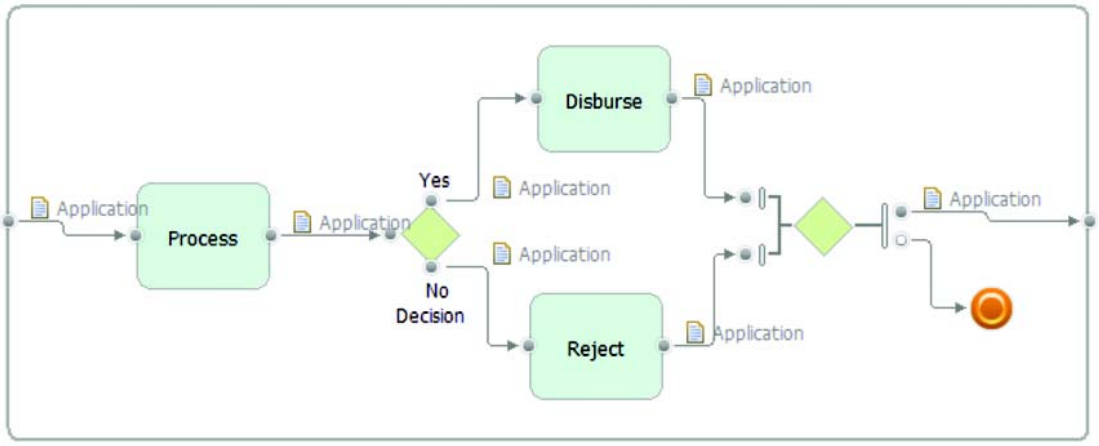


Figure 2-2: BPMN Example

For instance, in Figure 2-3, a business measures summary shows the KPIs, instance metrics, and aggregated metrics defined for the process in Figure 2-2. The KPI definition specifies the method of calculation, given the current instance and aggregated metrics, a predefined set of aggregation functions, and the time period for data collection. For the Average Process Duration KPI in Figure 2-3, calculation involved the average duration of "BPMNExample Processing Time" instance metrics on a rolling period of (last) 30 days. Analysis helps management reduce the current target Average Process Duration KPI to less than 3 seconds in order to improve customer satisfaction.

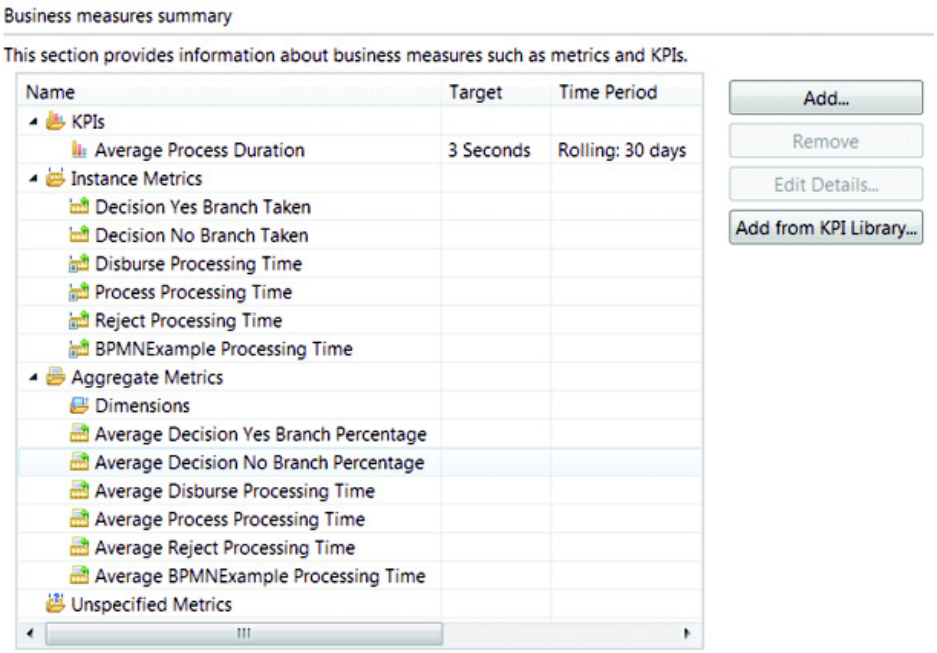


Figure 2-3: Business Measures Summary

2.3.2 Assemble and Deploy Phase

The newly designed process can be imported as BPEL code and assembled as an application by wiring the activities defined in the previous stage to their implementations. Figure 2-4 shows the equivalent representation of Figure 2-2 in BPEL.

Note that BPMN diagrams flow horizontally while BPEL diagrams flow vertically, due to the usual business analyst and IT specialist preferences, respectively.

Once the assembly phase is finished, the execution environment can be configured for security purposes, data sources can be defined, and the executable BPEL model can be deployed on a process server. A business monitor application can also be deployed in order to capture and provide the business metrics in real time.

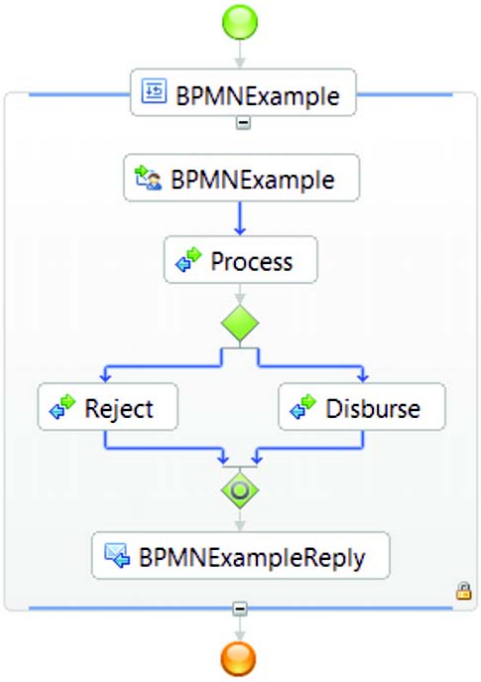


Figure 2-4: BPEL Mapping

2.3.3 Business Process Management—Management Phase

Measurement results taken from the monitor are useful during the *Manage* phase for simulation and for further business process improvement decisions. Recorded activity durations and decision node probabilities can be inserted into the simulation model to produce realistic statistical reports about the performance of the currently deployed process or possible improvements. Management revises old strategy and objectives, and new business objectives are defined to increase revenue and reduce costs. This marks the end of the BPM cycle where a business process has been defined, assembled, deployed, monitored, and simulated for improvement.

Once the process is optimized, it is redeployed, and the BPM cycle is repeated. The main challenges of this approach include the following:

- In order to make the right evolution decisions, a simulation model should emulate the business process precisely. At the highest level, the structure of the business model and that of the simulator can match, but the business process has many implementation details that cannot be accurately captured in the simulation model. For example, the business process KPIs can be affected by queuing delays in the underlying IT infrastructure, which is not captured in the simulation models. If the state space of the simulation model resembles the real system's state space, it can often answer any questions about the system [9]. Therefore, there is a necessity to investigate techniques that build a simulation model that accurately reflect the real processes.
- A business process life cycle iteration is very long. This creates the possibility that the operating conditions captured in instance traces and further in a simulation model are outdated at the time of the decisions. Therefore, it is advantageous to shorten the life cycle and, even more importantly, to automatically implement some evolution decisions [1, 12].

2.4 A Business Process Evolution Architecture

We propose an evolution architecture based on a feedback loop, as shown in Figure 2-5.

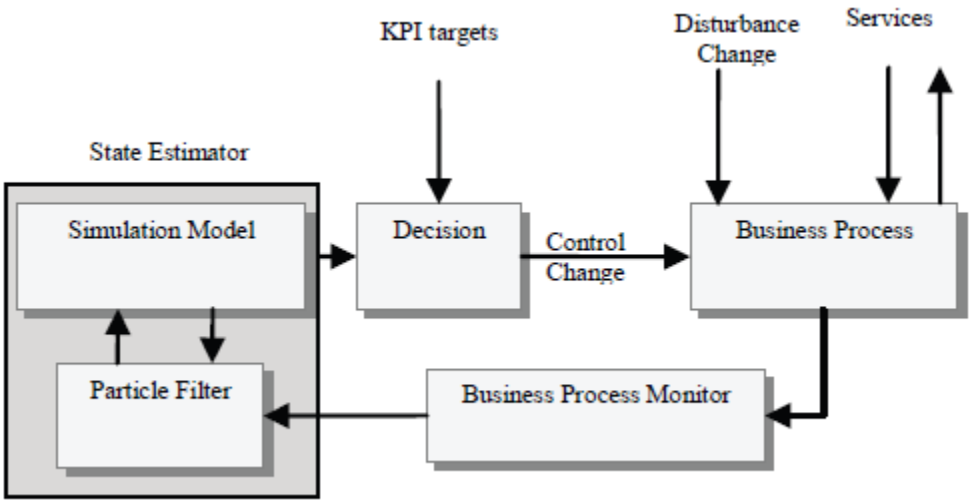


Figure 2-5: Proposed Optimization Loop for a Business Process

A deployed business process makes use of service components, such as web services, that can experience external disturbances (e.g., varying arrival rates). The process and its subcomponents are monitored by the business process monitor and recorded for future analysis. This component also provides real-time access to critical KPIs. The system's goal is to maintain performance of the process close to the reference KPI targets while optimizing the use of resources. The performance, which might vary because of disturbances, is monitored and fed back through a state estimator. The state estimator output reflects an estimated system performance. It iteratively generates simulator inputs (for example, with the aid of a particle filter) until the estimated system performance matches the monitored performance.

Finally, a decision block runs a control algorithm that uses the estimated system performance configuration to decide on a suitable correction to bring the monitored performance closer to the KPI targets using context-specific actuators [3].

The state estimator is the key element in the control loop, because its goal is to find simulator parameter estimates that cannot be measured directly from a set of monitor observations coming sequentially over time. Automatic model calibration, which is needed for accurate simulations that reflect the real system's unpredictable performance drifts, is achieved with the aid of a particle filter and a business process simulator.

With sufficient sample diversity, particle filters proved to be more accurate than conventional filters, such as the extended Kalman filter (EKF) or the unscented Kalman filter (UKF) [4, 5, 8], on nonlinear problem estimation. Raw monitored data contains noise, while end-to-end activity durations include queuing delays and hide other relevant information, such as branching probabilities. Aggregated measurements are obtained by removing the noise from raw monitored data through smoothing. Task service times without queuing delays are obtained from the state

estimator by finding the best input that brings the output of the simulation model closest to the smoothed, monitored, end-to-end process duration.

2.5 A Case Study and Preliminary Results

We used IBM WebSphere Business Modeler Advanced to design, simulate, and analyze a business process under a wide variety of conditions. For monitoring, we used WebSphere Business Monitor on a WebSphere process server. Finally, WebSphere Integration Developer (WID) was used for the assembling (assemble BPM life-cycle phase) and deployment (deploy BPM life-cycle phase) of business processes.

With the aid of the IBM WebSphere Modeler tool, we designed a simple process containing only three tasks and a decision node. In order to observe how the state estimator responds to load variation, each task is automated by a web service and its service time is controlled to follow a sine function. In reality, these service demands do not have the high variation generated in these experiments. However, we consider them here to stress the architecture. Also, in real business processes, many of the business process activities are performed by people. Our implementation has not considered this case.

2.6 Results

Workloads were designed to explore different scenario conditions by capturing arrival rate variation with uniform service time for each task, service time variation with uniform arrival rate, and both inter-arrival time and service time variation. Detailed results about the estimation process were reported by Solomon and colleagues in another paper, which is currently under review [10]. In the next paragraphs, we present a summary.

We consider two cases, illustrated in Figure 2-6 and Figure 2-7, where the first configuration uses the simulator alone and the second configuration uses a state estimator. For the first configuration, the simulator input contains both the queuing time and the service time for each task. The output, produced by the estimator with queuing time removed for each task, becomes the simulator input in the second configuration. Then, we compare the results of the simulation both with and without an estimator in Figure 2-7.

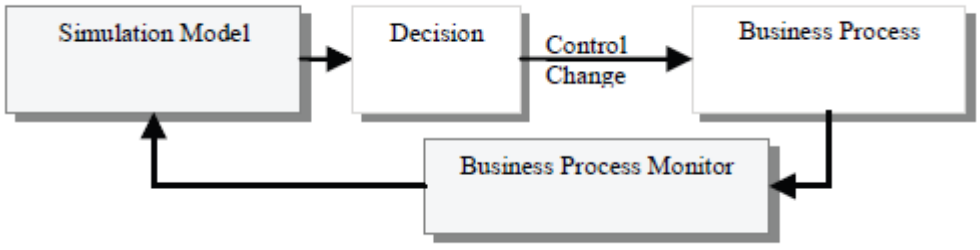


Figure 2-6: Configuration without a Particle Filter

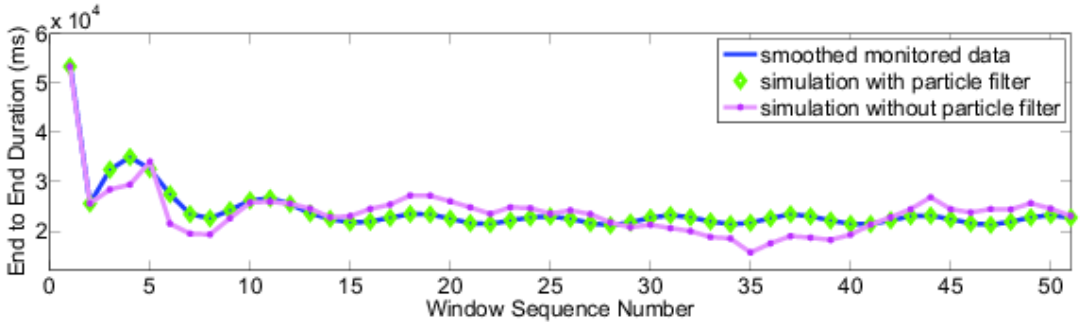
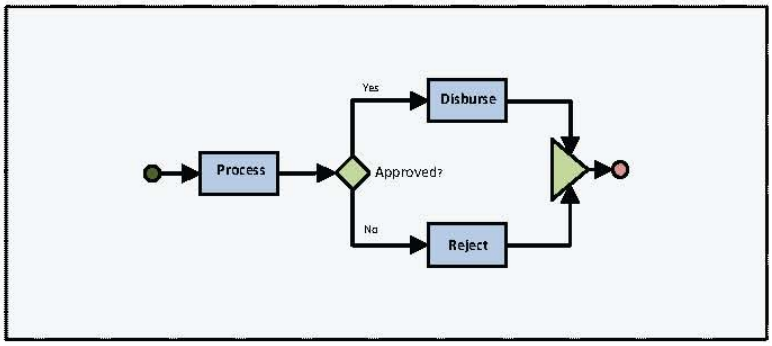


Figure 2-7: State Estimation

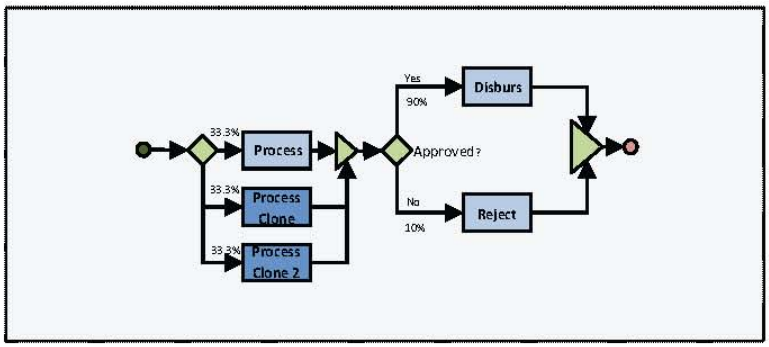
We define the modeling error as the absolute difference between the end-to-end simulated duration and the measured response time averaged over the number of window sequences. For the experiments where no estimation is used, the modeling error is 1,411 ms. On the other hand, the particle filter algorithm runs for a maximum of 10 iterations between any two measurements and returns an optimal solution with an average modeling error less than 0.96 ms [10].

2.7 Using a Simulation Model to Improve the Process

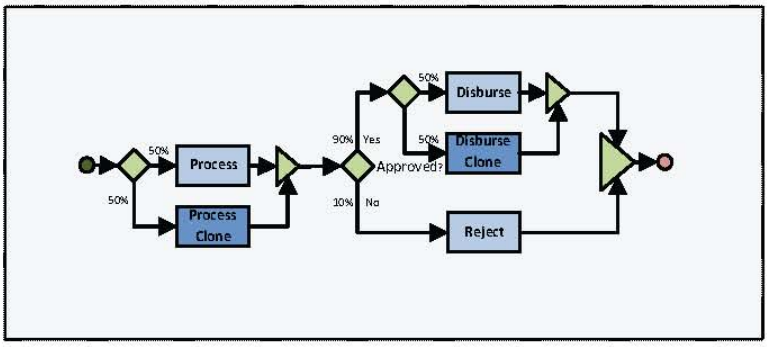
Business process improvement can be achieved through many means. For example, if the end-to-end response time of a business process is too high, the cause may be insufficient resources allocated to the process. The remedy is to add more. In the case of IT resources, those can be added dynamically by the decision block. Part of the decision block goal is finding poorly performing tasks that slow down the overall performance of the currently deployed business process. Figure 2-8(a) presents a business process with three tasks, each one being automated by a web service: “Process,” “Disburse,” and “Reject,” respectively.



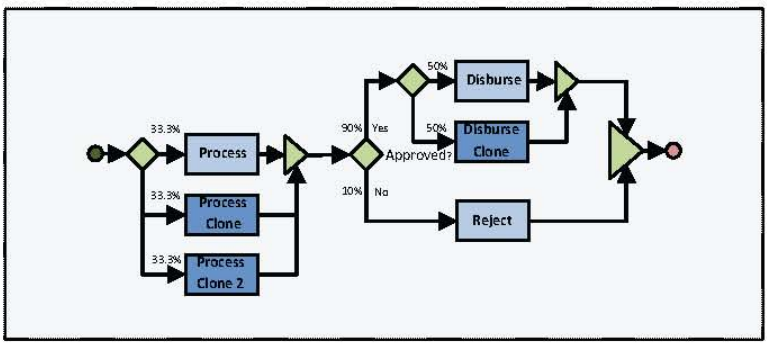
(a) Original configuration.



(b) 2 clones are added to worst performing task.



(c) Distributing clones between worst performers



(d) Configuration (b) and (c) combined.

Figure 2-8: Simulated System Configurations

A business process, together with its web services, can be deployed entirely on a single machine. The simulator can recommend tasks sharing the same resource or tasks that are deployed on different machines. Poorly performing tasks can be found and sorted dynamically by total queuing delay via simulation reports after running different possible inter-arrival times. Once the worst performing web service is known, possible allocation configurations are tried algorithmically by adding clones in parallel to the original task inside the business process (also known as horizontal scaling). Figure 2-8 illustrates how a new decision node is inserted just before the task to be cloned and the branching probability is divided equally among tasks.

Cloned tasks can be dynamically deployed on other machines to take over part of the increasing workload and maintain the service-level agreement (SLA) or KPIs within acceptable limits. For instance, assume that Figure 2-8(a) is the original configuration and activities "Process" and "Disburse" are those with the highest queuing delay, mentioned in descending order. Figure 2-8(b) and Figure 2-8(c) present possible future configurations where two cloned activities are added to the worst performing task "Process" and where they are distributed between the two bottlenecks, respectively. An alternative option would be the combination of configurations (b) and (c) in Figure 2-8(d). These four configurations' simulated performance is represented in Figure 2-9 as a possible solution space for our optimization problem.

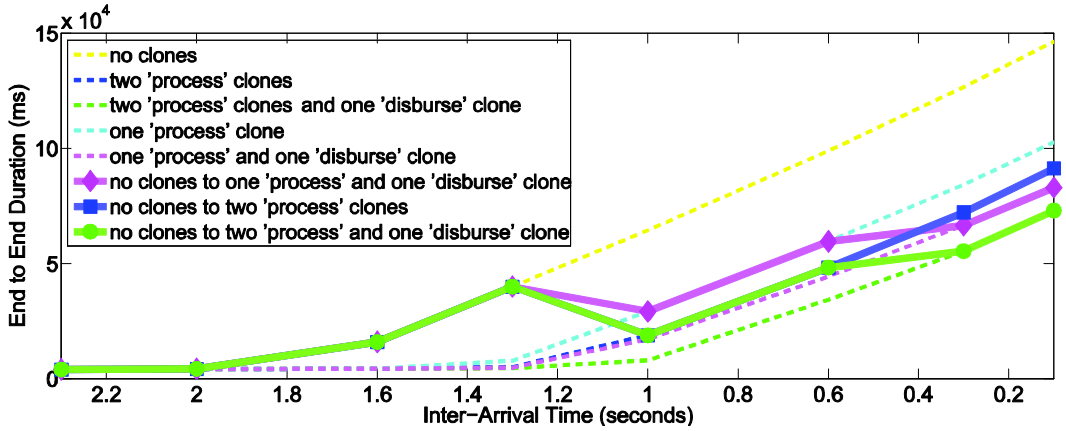


Figure 2-9: Scaling with Different Configurations

The simulator is used to calculate the total end-to-end duration of the business process for all chosen configurations at different inter-arrival times. Figure 2-10 shows a possible application of this optimization approach and the pattern of choosing the best configuration at runtime. The optimal solution shown tries to minimize the area under the SLA threshold line with minimum resources and cost. The second graph in Figure 2-10 reflects the transition between configuration and the number of clones needed to achieve our optimization goal. It is worth mentioning that the simulator can give great insight into the possible effective configurations.

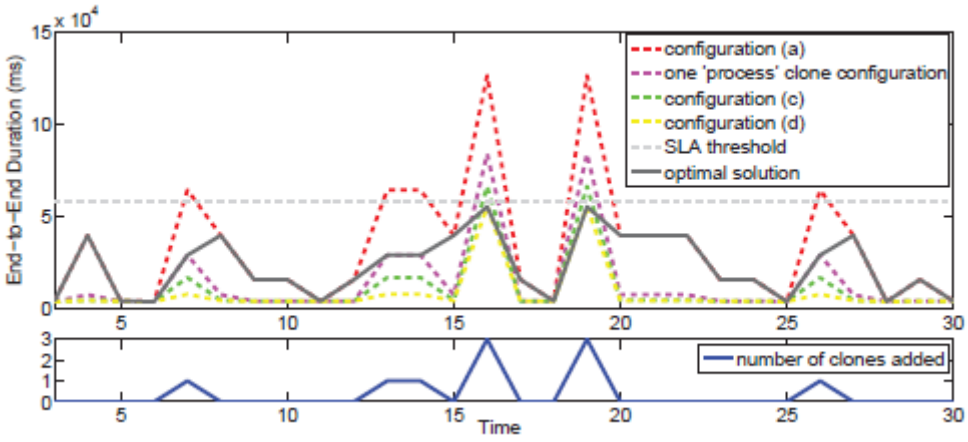


Figure 2-10: BPM Optimization Application

In Figure 2-9, we observe that configuration (c) performs better than configuration (b) by comparing the dotted line "one 'process' and one 'disburse' clone" with the "two 'process' clones" line. Therefore, it is actually better to distribute the clones between worst performers. This solution can be achieved iteratively by finding the current worst performer, cloning it in a new simulation, and then finding the new worst performer to be cloned, until the end-to-end duration of the process arrives under the SLA threshold. In the end, depending on the resource costs and KPIs, we can arrive at the most convenient configurations iteratively by computing the gain-to-resource-cost ratio and by testing them with the simulator in an algorithmic manner.

2.8 Conclusions and Future Work

We proposed a feedback-based evolution architecture for business processes. The architecture can be used for many evolution scenarios, but we envision it having the most impact on business process optimization. The architecture is centered on monitoring the business process and on using estimators to find hidden states in the process. The architecture deals with measurement and modeling errors and builds a simulation model that can be used for process optimization. We validated the architecture on a small business process using industrial products that support the entire life cycle of business processes. We used IBM WebSphere Modeler Advanced, WebSphere Modeler Simulator, WebSphere Integration Developer, and WebSphere Business Monitor.

Further work and future challenges include

- validating the architecture in larger-scale business processes. In general, a business process contains hundreds of activities that can stress both the simulation and the estimation components of our architecture. We know that particle filters have a high computational cost, and it might be necessary to implement other types of estimators.
- considering other types of resources. We modeled IT resources in our sample implementation. However, human resources are harder to model in the sense that their past performance cannot be an accurate indicator of future performance. Also, human resources introduce time granularities (hours, days) that are several orders of magnitude higher than

those of IT resources (milliseconds, seconds, minutes). Considering these different time granularities might prove challenging for both the simulator and the estimator.

- automating the optimization of IT resource allocation. Many of the optimization decisions can be implemented at runtime. There should be no need to go through the whole life cycle and wait for a new version of the process to allocate more IT resources to the process. However, automation has to be considered carefully and must be supported by the underlying infrastructure.
- predicting future states and KPIs. So far, we have looked at the history and current state of the process and therefore of the KPI. Predicting future states and KPIs opens new opportunities for business process optimization.

We would like to apply linear regression and autoregressive model techniques to predict trends from historical data and the correlation between metrics. We would also implement an optimization algorithm that will use the simulation model and current and forecasted KPIs to optimize a business goal by tuning or reallocating resources or by guiding business users with options they could consider.

References

[1] M. Al-Mashari and M. Zairi. BPR implementation process: an analysis of key success and failure factors. Business Process Management Journal, 5(1):87–112, 1999.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, and S. Thatte. Business process execution language for web services specification, version 1.1. IBM, Microsoft, BEA, SAP, and Siebel Systems, 2003.

[3] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzé, and M. Shaw. Engineering self-adaptive systems through feedback loops. Software Engineering for Self-Adaptive Systems, pages 48–70, 2009.

[4] S. J. Julier and J. K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In International Symposium on Aerospace/Defense Sensing, Simulation and Controls, volume 3, page 26. Citeseer, 1997.

[5] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. A new approach for filtering nonlinear systems. In Proceedings of the American Control Conference, volume 3, pages 1628–1632. American Automatic Control Council, Evanston, IL, 1995.

[6] R. K. L. Ko, S. S. G. Lee, and E. W. Lee. Business process management (BPM) standards: a survey. Business Process Management Journal, 15(5):744–791 2009.

[7] R. J. Paul, V. Hlupic, and G. Giaglis. Simulation modeling of business processes. In Proceedings of the 3rd UK Academy of Information Systems Conference, pages 311–320. Citeseer, 1998.

[8] A. Romanenko and J. Castro. The unscented filter as an alternative to the EKF for nonlinear state estimation: a simulation case study. Computers & Chemical Engineering, 28(3):347–355, 2004.

[9] P. J. Sánchez. Fundamentals of simulation modeling. In Proceedings of the 39th Conference on Winter Simulation: 40 years! The best is yet to come, pages 54–62. IEEE Press, 2007.

[10] A. Solomon, M. Litoiu, J. Benayon, and A. Lau. Business process adaptation on a tracked simulation model. In Proceedings of ACM IBM Center for Advanced Studies Conference, pages 184–198. CASCON, 2010.

[11] L. Verner. BPM: the promise and the challenge. Queue, 2(1):91, 2004.

[12] B. Wetzstein, Z. Ma, A. Filipowska, M. Kaczmarek, S. Bhiri, S. Losada, J. M. Lopez-Cobo, and L. Cicurel. Semantic business process management: a lifecycle based requirements analysis. In Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM-2007), volume 251, pages 1613–0073. Citeseer, 2007.

3 Requirements-Driven Framework for Root Cause Analysis in SOA Environments

Hamzeh Zawawy, University of Waterloo, Canada (hzawawy@engmail.uwaterloo.ca)
John Mylopoulos, University of Toronto, Canada (jm@cs.toronto.edu)
Serge Mankovskii, CA Technologies, Canada (Serge.Mankovskii@ca.com)

Abstract

Problem diagnosis in service-oriented systems is a challenging and complex task. Thus, automated support for human operators and administrators who perform root cause analysis is most often required. We propose a requirements-driven root cause analysis framework for multilayered service-oriented systems. The proposed root cause analysis framework uses annotated requirements goal models to represent computer systems of interest. Diagnostic reasoning is achieved by analyzing log data against goal trees. A root cause analysis diagnosis entails those components and services from a goal tree that can be shown as contributing to the failure of the functional or nonfunctional behavior observed. The diagnostic component returns sound diagnoses accounting for observed aberrant behaviors for each monitored system. A case study involving commercial off-the-shelf (COTS) components with multiple machines and layers including a business process layer, a middleware layer, and a services layer, as well as an infrastructure and database layer, was conducted to evaluate the diagnostic capabilities of the proposed framework.

3.1 Introduction

Problem determination in enterprise environments is typically a highly manual process. To deliver customer service at agreed-upon levels, a computer system needs to be monitored to ensure reliable operations and to resolve problems in a timely fashion. A typical enterprise environment contains multiple applications that are hosted on different machines, in different sites or geographies, and that are running under various operating systems. Enterprise applications run in heterogeneous environments and interact with one another to exchange technical and business information. The task of monitoring becomes more complex for distributed systems, such as systems based on service-oriented architecture (SOA).

Most applications and hardware devices generate log files to capture information about their internal events. Vendors typically use their own proprietary log file formats to report on the health of their devices or software applications. Products from the same vendor may use different log file formats, due to acquisitions or simply due to lack of a log format standard in the same organization. Thus, each system may generate log data in its own logical and physical format. In addition, the corresponding logs will have different hierarchies, locations, access rights, and so forth. As a result, specific application or system knowledge is required to monitor and debug each application as it executes over heterogeneous systems. In addition, systems such as web servers, sensor networks, transactional databases, and others generate a large quantity of log data, making manual problem determination a very time-consuming task. To help in problem determination,

automated support for performing root cause analysis is most often welcomed. Root cause analysis (RCA) consists of a set of techniques and processes that attempt to discover faults that are responsible for an observed failure. RCA is typically performed by analyzing log files emanating from various loggers and monitors.

There are two main approaches toward performing RCA. The first is a rule-based approach, in which specific rules denote diagnostic knowledge much like a diagnostic expert system would do. These approaches suffer from a problem known as rule coverage, as it may be very difficult to encode in rules all possible situations by which a complex system may fail. The other approach is model-driven, in which a model of the system is built and diagnostic knowledge takes the form of identifying the combination of component faults that may contribute toward an observed failure. These approaches are more flexible, but it is not always possible to build an accurate model of the system.

In this paper, we opt for a model-driven approach using goal trees that associate specific system behavior, as well as functional and nonfunctional requirements, with specific system components and services. The particular technique is adapted from work presented in Wang et al. [27], in which the normal operation of the system is described in terms of a goal model that denotes system requirements and the diagnostic component uses a Boolean or propositional satisfiability (SAT) solver from the artificial intelligence theory of action and diagnosis. These goal models are either reverse engineered from source code using techniques presented in Yu et al. [28] or are provided by requirements analysts. More specifically, we propose the use of annotated goal trees to model requirements, and we then propose a temporal query language to analyze log data and verify the occurrence or lack of key events associated with the operation of system components being diagnosed. Failure to observe expected patterns of precondition, occurrence, and effect (postcondition) events associated with the expected behavior of a component serves as an indication of a possible failure of this particular component or of the components it depends on.

To showcase and evaluate the RCA framework, we used a collection of COTS software applications simulating a multilayered service-oriented system and injected several different faults pertaining to infrastructure, application, and business process layer components. The rest of the paper is structured as follows. Section 3.2 covers the research baseline of our work. Section 3.3 is a description of the proposed RCA framework. Section 3.4 describes the test environment and scenarios. Section 3.5 represents an experimental evaluation for the framework. Section 3.6 represents the conclusions and future work.

3.2 Related Work

This paper proposes a framework for performing RCA in multilayered service-oriented systems. This work builds on and extends the monitoring framework presented by Wang et al. [27], which was based on an SAT diagnostic engine, where the problem of diagnosis is turned into a problem of monitoring the satisfaction of software requirements that are represented as goal models. Below we discuss related work in the areas of requirement goal modeling, RCA, and log analysis in more detail.

3.2.1 Goal Models

Goal models have been used in requirements engineering to model and analyze stakeholder objectives [25]. Hard goals and soft goals are used to represent functional and nonfunctional requirements, respectively [16]. A goal model is a treelike structure. By using *AND*- and *OR*-decompositions, goals are broken down into subgoals.

Using means-ends links, leaf-level goals are linked to tasks (actions) or design decisions that should be performed to fulfill the leaf goals. The non-leaf nodes of the tree represent subgoals, and the leaf nodes represent tasks. All the nodes of the requirements model are annotated with preconditions and effects. For example, if goal *G* is *AND/OR*- decomposed into subgoals *G1, G2,* on to *Gn*, then all or at least one of the subgoals must be satisfied for *G* to be satisfied. Another type of link that exists among goals is contribution links [8]. Hard goals can be related to one another through contribution links: ++*S*, --*S*, ++*D*, and --*D*. For two goals *G1* and *G2*, the contribution link ++*S* from *G1* to *G2* (respectively --*S*) means that if *G1* is satisfied, then *G2* is satisfied (and respectively denied). However, if *G1* is denied, we cannot deduce denial (or respective satisfaction) of *G2*. The links ++*D* and --*D* have the same meaning to deniability as ++*S* and --*S* have for satisfiability. The formalization of this class of goal models is done by Giorgini et al. [7], who provide sound and complete algorithms for inferring whether a set of root-level goals can be satisfied.

3.2.2 Root Cause Analysis

In the context of software maintenance, RCA represents a class of techniques for the detection and identification of concealed faults that are at the source of a system failure or a user-reported incident. Although most of the literature on RCA has been in the area of low-level communication systems, the advent of SOA has led to more interest in developing and adopting RCA techniques in higher layers, such as business processes and software services layers.

Hanemann proposes a hybrid reasoning approach, in which the root cause of an incident is identified by first searching a set of rules that map symptoms to root causes [11]. If no matching symptom is found, this incident is treated with a case-based approach by manually resolving it the first time it is encountered and then storing it for future reference. The drawback to this approach is that it is hard to identify all the needed rules. Steinder analyzes the symptoms in the system and uses Bayesian belief networks to model end-to-end services in a system and compute the most probable explanation set [23]. Yuan collects trace data and compares it to already collected sets of trace data [29]. History log and trace data are already mapped to symptoms. Based on the result of the comparison, the problem is classified and a potential root cause is identified. Similarly to the work of Steinder and Sethi [23], our proposed approach includes modeling the system to be monitored before using the RCA framework; however, the preparatory effort is substantially less than in the case of rule-based approaches, which require building association relationships [11], [17]. Similar to other approaches relying on raw log data, the diagnosis produced by the framework directly depends on the quality of the collected log data.

Another technology that has been recently used for RCA in software systems relates to the propositional satisfaction of system properties using SAT solvers [27]. More specifically, given a propositional formula *f*, the SAT problem consists of finding values for the variables of the propositional formula that can make an overall propositional formula evaluate to true. The

propositional formula is said to be true if such a truth assignment exists. An SAT solver is a procedure that determines the satisfiability of a propositional formula, identifying the satisfying assignments of variables. Most SAT solvers use a conjunctive normal form (CNF) representation of the Boolean formula *f*. In CNF, the formula is represented as a conjunction of clauses, where each clause is a disjunction of literals and a literal is a variable or its negation. Note that in order for the overall formula *f* to be satisfied, each clause must evaluate to true.

Inherently, the SAT problem is intractable, but recently there have been many improvements to SAT algorithms. The earliest and most prominent SAT algorithm is the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [5], which uses backtracking search. Currently, the fastest SAT solvers are those of Chaff [15], Berkmin [9], and Siege [19]. Wang et al. [27] used SAT4J, an efficient SAT solver that inherited some features from Chaff's [15]. The framework in this paper uses goal models as formalized by Giorgini et al. [7]. Similar to Wang et al. [27], we annotate goal models with preconditions and effects (postconditions). We introduce a new type of annotation to represent the actual occurrence of the goal or task. Unlike the work of Wang et al. [27], in which annotations are mere links between the goal model and the actual source, the annotations used in our work are logical expressions that are described using temporal query language.

3.2.3 Log Analysis

Log analysis (or trace analysis) aims at analyzing computer-generated log data (also called an audit trail). Log data is generated by computer systems to report an error or for audit purposes. The main reasons to do a log analysis are, first, security reasons; second, to show compliance with enterprise regulations or system requirements; and, third, system troubleshooting. In our literature review, we cite Denning [6], who presents a real-time intrusion detection model based on profiling log data and detecting abnormal patterns of system usage. And Shieh et al. present a system to detect anomalies in operational security based on models that represent various patterns caused by the unintended use of programs and data [20].

Andrews proposes a state-machine-based log analysis framework with applications in unit and system testing [1]. Vaarandi presents a data mining algorithm for profiling and pattern recognition in log file data sets [24]. Cho et al. present an intrusion detection system based on hidden Markov models representing the normal operation of a system. The system incorporates a soft-computing-based analysis technique to detect intrusions by noting significant deviations from the model [4]. Jiang et al. discuss log analysis applications for RCA and troubleshooting purposes and conclude that relying on a single failure message is a poor indication of root cause failure but that aggregating messages can increase the accuracy by a factor of three [14]. Wang et al. instrument the source code of monitored applications and use the generated log data to verify the requirements satisfaction as well as for system diagnostics [27]. Similarly to the work presented above, we use log data as evidence to prove or disprove events of interest; however, unlike the work of Wang et al. [27], the proposed framework does not instrument the systems to be monitored but instead relies on raw log data generated by these systems. Unlike the work of Andrews and Zhang [1], Cho [4], and Jiang et al. [14], the proposed framework does not depend on the specific syntax or semantics of the log data but instead follows a best-effort approach based on the available log data. We have proposed the use of latent semantic indexing as a filtering technique to collect log data [30]. In this context, we treat log data and the goal model annotations

queries as documents, and we calculate the similarity among the log entries and the queries using the concept-document matrix.

3.3 Monitoring and Diagnosing in SOA

In this section, we describe the proposed RCA framework. Figure 3-1 illustrates a block diagram of the framework. The aim of this framework is to discover the root cause of a system failure, as this is typically manifested by an initial system alert emitted by one or more components of the distributed service-oriented system. Typically, the framework is intended to be operated by a system administrator who receives the alert and starts the diagnostic process. The framework starts by parsing the input (goal model, log data, and user's points of interest) into memory. It then extracts a subset of the log data and uses it to verify the occurrence of certain events of interest. Based on the analysis of log data, the requirement model of the system being analyzed, and a SAT solver, the framework comes up with a list of potential underlying systems that may have triggered the alert of concern.

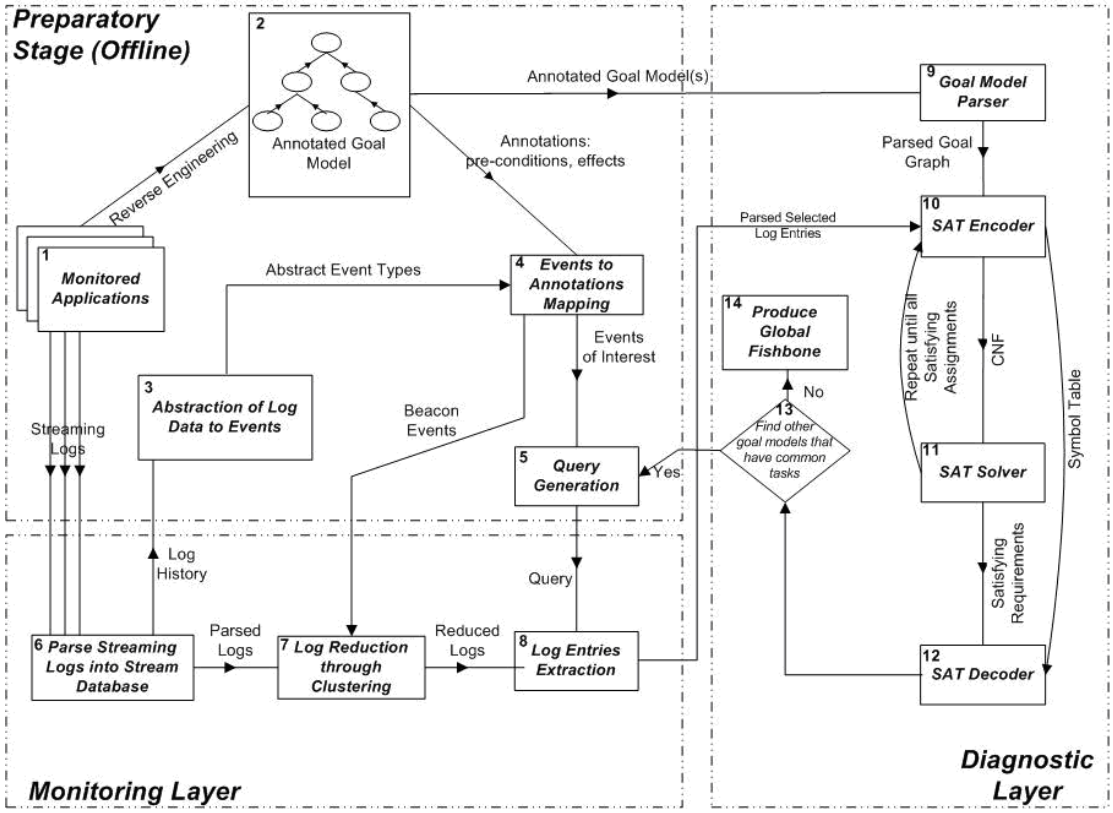


Figure 3-1: Block Diagram for the Proposed RCA Framework

3.3.1 Framework Inputs

As mentioned above, the input to the proposed RCA framework includes the log data and the requirements-related goal model(s) for the monitored applications, which can be either reverse engineered from the monitored applications or modeled by requirements analysts. The goal models that we use to represent the systems of interest have been originally formalized in Giorgini et al. [7]. Goal models for the monitored applications are built in an offline preparatory step

before using the RCA framework. The RCA process is typically started by the system administrator when a system alert is raised. The system administrator could be concerned about this specific system alert and would like to know the possible root causes for this alert. The RCA framework starts by analyzing the system that generated the alert.

The second input to the RCA framework is the log data generated by the monitored applications. Log data is received as sequences of events from the different monitored applications and is stored in a stream database, as we will discuss in Section 3.3.2.

The third input to the RCA framework is the set of domain constraints that represent the system administrator's points of interest, such as specific machine names or IP addresses, user names or IDs, time intervals, and so forth. The domain constraints are expressed using the query language that we describe in Section 3.3.5.

3.3.2 Streaming Log Data Normalization

The stream database management system (SDMS) component is a collection of data management systems that handle streaming data consisting of a real-time, continuous, ordered (explicitly by timestamp or implicitly by arrival time) sequence of logged events. SDMS extends the relational model by using windowing operators, in which the answer for a query is bounded in time by window attributes that are added to the query. Examples of the general purpose SDMS are the Aurora and STREAM systems presented in Golab and Ozsu [8].

In the context of this work, we assume that the logged data is based on a reduced version of common base event [12]. The reduced data set contains timestamp, context information (e.g., host name), process ID, event type, event result (successful or failed), and a general description (see Table 3-1 for more details). The choice of the fields to be included in this reduced set was made by examining log data coming from different logging frameworks and selecting a representative set of these fields on a reduced set.

Table 3-1: LogData Database Table: A Unified and Reduced Set of Log Data

Column Name	*Corresponding CBE Field*	*Description*
TIMESTAMP	CREATION_TIME	Date/time of log entry creation based on the local clock.
EVENT_NATURE	EXTENDED_DATA_ELEMENT_EVENT_NATURE	Name that is assigned to the abstract event type
DESCRIPTION	N/A	General description of the event
PROCESS_ID	SOURCE_COMPONENT_ID_PROCESS_ID	ID of the process that generated the log message
SITUATION	SITUATION_START_SITUATION_SUCCESS_DISPOSITION	Enumeration of: SUCCESSFUL OR FAILED
SITUATION_QUALIFIER	SITUATION_STARTSITUATION_SITUATION_QUALIFIER	Enumeration of: COMPLETED OR INCOMPLETE

3.3.3 Log Data Abstraction

This section is an offline preparatory step that corresponds to Step 3 in Figure 3-1. Generated log data for each component in a distributed system usually conforms to the same log format schema, but each log entry can contain dynamic sections that are specific to the particular transaction or session. They may refer to a particular transaction or session ID or contain common contextual data. Log abstraction algorithms parameterize dynamic contents of the log entries. By doing so, log lines generated by the same execution events will look the same. This enables us to associate an abstract log type with each execution event type. In the proposed RCA framework, we use the Simple Logfile Clustering Tool (SLCT) to abstract the different event types from all the log data generated by the different applications. SLCT has a C-based implementation available for academic research [21].

The resulting set of abstract log types can be described using query language by assigning a query expression to each event or log abstract type. As mentioned earlier, log entries contain static and dynamic parts. The static part is common for the different instances that belong to the same type of events, while the dynamic part is specific to each log instance. To represent the different event types, we use the concept of watch rule expression, which is part of BEA's WebLogic Diagnostic Framework event query language [2].

The Backus-Naur Form of the proposed query expression is described below.

```
queryExpression := searchClause | boolean queryExpression
searchClause := columnField comparitor identifier | searchClause
columnField := TIMESTAMP | CONTEXT | EVENT_NATURE | DESCRIPTION | PROCESS_ID
    | SITUATION | SITUATION QUALIFIER
comparitor := symbolComparitor | namedComparitor
comparitorSymbol := > | < | >= | <= | <>|=
namedComparitor:= LIKE | MATCHES | IN | LATER | BEFORE
boolean := and | or | not
identifier := String | Integer | Timestamp | % | ? | identifier
```

An example of the query representation for the abstract log or event type is the *REQUEST* event type that starts the *Apply_For_Loan* business process:

```
Loan REQUEST: (CONTEXT LIKE 'Machine2'; 'SOAPUI'; %) AND (DESCRIPTION LIKE
'%Apply_For_Loan%') AND (EVENT_NATURE = 'POST')
```

Consequently, a mapping is established among constructs in the proposed query language and a subset of the standard object constraints language (OCL) [18]. OCL 2.2 is a declarative text language for describing rules and constraints that apply and align with UML 2.2 and MOF 2.0 models, and it is a key component of queries/view/transformation, the Object Management Group (OMG) standard recommendation for model transformation.

First, the *SELECT* statement containing the *queryExpression* in the proposed query language maps to the *ExpressionInOCL* in the OCL specification. The *queryExpression* maps to *FeatureCallExp*, which is a subtype of the *OCLExpression* in the OCL specification. The *columnField* and *comparitor* clauses in our proposed query language map respectively to the *TypeExp* and *LiteralExp*, which are subtypes of *OCLExpression*. The *comparitor* clause contains time-based operands that allow comparison and handling of the timestamp fields, which are clearly essential when handling log data.

This mapping is useful in supporting the applicability and showing the limitations of the proposed query language. On the other hand, the added value of our proposed query language is in being a lightweight query language with temporal extension for queries, which is not available in the standard OCL. In this context, the choice of OCL as a formalism to represent the annotations on the monitored goal model is motivated by the fact that OCL is designed as a query language to support MOF modeling, which is the modeling framework used to represent our domain models. In addition, OCL is an OMG standard with existing engine implementations, such as the DresdenOCL toolkit [3].

3.3.4 Abstract Events to Goal Model Annotations Mapping

This step consists of mapping preconditions, effects, and occurrences of goals and tasks in the goal model to the abstract event types (generated in the log abstraction step described in Section 3.3.3). The mapping (Step 4 in Figure 3-1) is done manually by an expert user. The expert user assigns an event type to each goal or task's occurrence/precondition/effect in the goal model. As a result of this step, we select and extract from the log database a subset of log data that verifies or denies the precondition, effect, and occurrence of each goal or task. An example of the mapping based on the use case that we present in Section 3.5 is shown by assigning the Loan Request query sample in Section 3.3.3 to the precondition event of goal *g1*: *Apply for Loan*. This mapping operation assigns a query representation to the events of the goal model using the query language that we have described in Section 3.3.3. This query representation will be extended in the next step as part of the automatic query generation and will be eventually used to extract actual log data from the log database that in turn will be used for root cause diagnosis and analysis.

3.3.5 Automatic Query Augmentation

To automate the process of log data extraction, we propose an algorithm to generate queries that will be used to extract the log data (Step 5 in Figure 3-1). The extracted log data is used in a subsequent step to verify the occurrences of tasks in the goal model, as well as their preconditions and effects (Step 7 in Figure 3-1). The algorithm works by augmenting the queries that describe the annotations on the goal model with the domain constraints supplied by the user. For example, the REPLY abstract event type is represented with the following query expression:

```
CONTEXT LIKE 'Machine1; 'ApplyForLoan';%)
AND (DESCRIPTION LIKE '%Apply_For_Loan%')
AND (EVENT NATURE = 'REPLY')
```

On the other hand, the user's points of interest represent a set of domain constraints that are typically a time constraint (from time T1 to time T2), machine names, application and component names, process or user IDs, and so on. These domain constraints can be represented using a similar query expression:

```
T.TIMESTAMP > T1 AND T.TIMESTAMP < T2
AND (T.CONTEXT = Server1)
AND (T.DESCRIPTION LIKE DESC)
AND (T.PROCESS = P1 OR T.PROCESS = P2)
```

The final augmented query is produced by the union of the two partial queries. The query generation algorithm takes as input a set of user-defined domain constraints and a set of queries representing goal model annotations (preconditions, effects and occurrences events) and generates a set of queries that will retrieve log data that match the query. The algorithm below illustrates this process in more detail.

Algorithm: Query Augmentation

Inputs:

- Domain Constraints Query: *Constraints(T1, T2, [Contexts],[DescStrings], [Processes])*
- Set of annotations / event types of interest (preconditions, effects, occurrences): *Annotations [Event]*

```
Output:
Query to extract all log entries of interest:
FinalQuery = { Time1, Time2, [Contexts], [Events], [DescStrings],
[Processes],
Situation],[Situation Qualifiers] }
GENERATE FINAL QUERY (Constraints, Annotations)
// Create a query for each annotation by adding constraints information to
it.
// First, set the time frame for the new query to start based on the user
// constraints, but ends after a whole business process session is
completed.
FinalQuery.Time1 = Constraints.T1
// Let S be the time interval the complete session needs to complete top
goal
If Constraints.T2 < (Constraints.T1 + sessionTime)
FinalQuery.Time2 = Constraints.T1 + sessionTime
else
FinalQuery.Time2 = Constraints.T1
// For each event type of interest, assign all the different values that the
user
// has specified
For each event E from Annotations[n]
Create a new query expression queryN
queryN.Context = Constraints.Context[]
queryN.Desc = Constraints.Desc[]
queryN.Process = Constraints.Process[]
```

3.3.6 Goal Model Parser

The RCA framework includes a goal model parser (Step 8 in Figure 3-1) that parses a serialized copy of the goal model of interest and uses the Eclipse Modeling Framework to load it into memory as an EMF model [13]. As an offline preparatory step, the goal model is described as an EMF-based model that uses the Eclipse Modeling Framework. Following the implementation of the goal models using the EMF, the Eclipse Modeling Framework is used to automatically generate a set of Java application programming interfaces (APIs) to parse or serialize the EMF-based models. This set of Java APIs is used to programmatically parse the goal model into memory as an EMF model. The generated set of APIs can also be used to update and serialize the EMF-based goal model using the XML metadata information (XMI) format, which is an XML-based format. This set of Java APIs represents the core of our goal model parser.

3.3.7 SAT Encoder, Solver, and Decoder

The SAT encoder is represented by Step 9 in Figure 3-1. The SAT encoder transforms the parsed goal model and log data offline into a propositional formula in CNF. The SAT solver (Step 10 in Figure 3-1) uses the CNF generated by the encoder and tries to find a set of variables in the propositional formula that makes it evaluate to TRUE. This combination represents a diagnosis. As a result, there could be no single diagnosis (system running correctly), or one or more single diagnoses. A diagnosis specifies if any goal or task is denied or not. If a denial is found, it is traced back to the lowest level task to identify the problematic components [26], [27]. If one task in the generated diagnosis represents a separate system, we can invoke the RCA framework again

by loading the corresponding goal model and repeating the whole analysis. By repeatedly invoking the RCA framework, we can find all possible diagnoses.

The next section describes the use case scenario and test environment that we have used in our experiments.

3.4 SOA/BPM Test Environment

To illustrate the application of the proposed RCA framework in distributed environments, we have assembled a service-oriented test application that is based on COTS software. The business process contains a financial business process application and a supporting web service. *Apply_For_Loan* is the business process containing loan application business logic. The business process, which is exposed as a web service, is invoked when it receives a Simple Object Access Protocol (SOAP) request containing the loan application information. It evaluates the loan request by checking the applicant's credit rating. If the applicant's credit rating is good, his or her loan application is accepted and a positive reply is sent back to the applicant. Otherwise, a loan rejection is sent back to the applicant.

If an error in processing occurs, a SOAP fault is sent back to the requesting application. The credit rating evaluation is done via a separate web service business application (*Credit_Rating*). Once the credit evaluation is done, the *Credit_Rating* service sends a SOAP reply back to the calling application that contains the credit rating and the ID of the applicant. If an error occurs during the evaluation, a SOAP fault is sent back to the requesting application. During the credit rating evaluation, the *Credit_Rating* application queries a database table and stores or retrieves the applicant's details. Figure 3-2 shows the requirements goal model of the loan application use case test environment systems and services.

The *Apply_For_Loan* business process is deployed into the IBM WebSphere process server. The *Credit_Rating* application is a business service exposed through the enterprise service bus (ESB) implemented using the IBM WebSphere message broker. The database table is hosted by the IBM DB2 database management system. The front-end application is simulated using *soapUI*, which is a web service invocation tool used by developers for testing purposes [22].

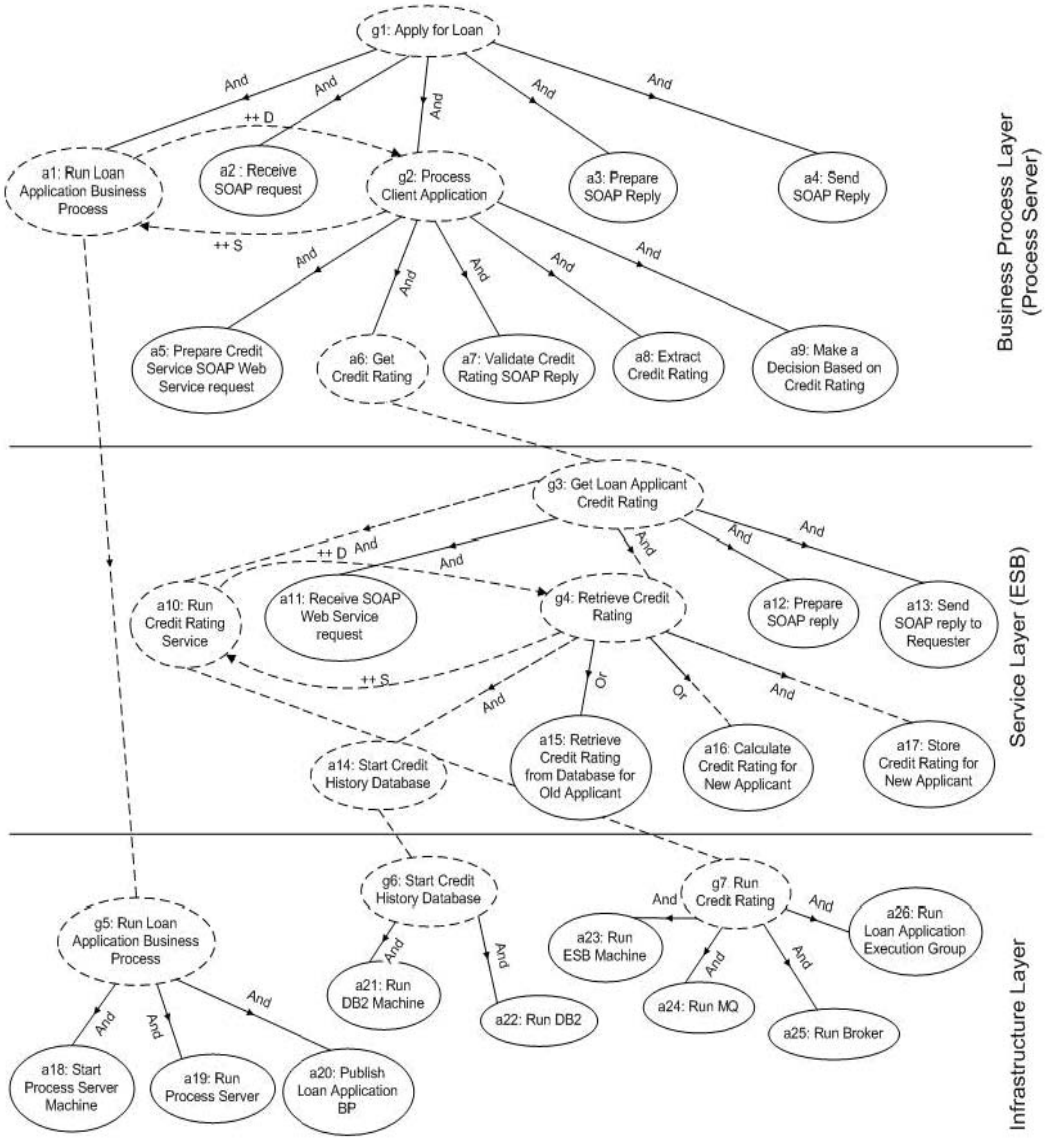


Figure 3-2: Layered Goal Model for the Systems Used in the Test Environment

3.4.1 Multilayered Distributed Environment

The use case test environment is a three-tiered environment containing a front-end application (*soapUI*) representing the presentation layer, a middle tier containing the business logic and middleware components, and a third tier containing the back-end database (IBM DB2). The middle tier consists of three abstraction layers:

- the business process layer: represented by business processes and the process server (IBM WebSphere business process/integration developer, or WID)
- the service layer: represented by the credit web service and the ESB
- the infrastructure layer: represented by the HTTP and MQ communication protocols (IBM WebSphere MQ)

To have a distributed environment in our use case, we installed the front end (requesting application) in a separate environment with an independent clock and IP address. The product versions used in this scenario are as follows: WID (6.2.0.2), WMB (6.1.0), MQ (6.0), and DB2 (8.2). The host operating systems are Windows XP Professional with Service Pack 2 [20, 23].

3.4.2 Logging Systems

The test environment includes four logging systems that emit or store the log data or events in the distributed environment. These logging systems are

1. Windows Event Viewer: a component of Microsoft Windows that acts as a centralized log service for running local applications. Events emanating from applications running as services such as the IBM Message Broker, DB2, and MQ can be accessed and viewed using the Windows Event Viewer.
2. front-end application (*soapUI*): an open source web service testing tool for SOA. The *soapUI* application logs its events in a local log file.
3. Middleware Service (Credit_Rating): the Credit_Rating application generates its own set of events and stores these events in a log file in the local file system, independently from events generated by the hosting ESB application.
4. WebSphere Process Server and Business Processes: the process server allows each business process to generate events based on a customized policy.

Figure 3-3 illustrates the architecture of the test environment, including the applications and their logging components.

The next section is a step-by-step description of the experimental evaluation of the framework, as well as an experimental analysis of the impact of event generation on performance.

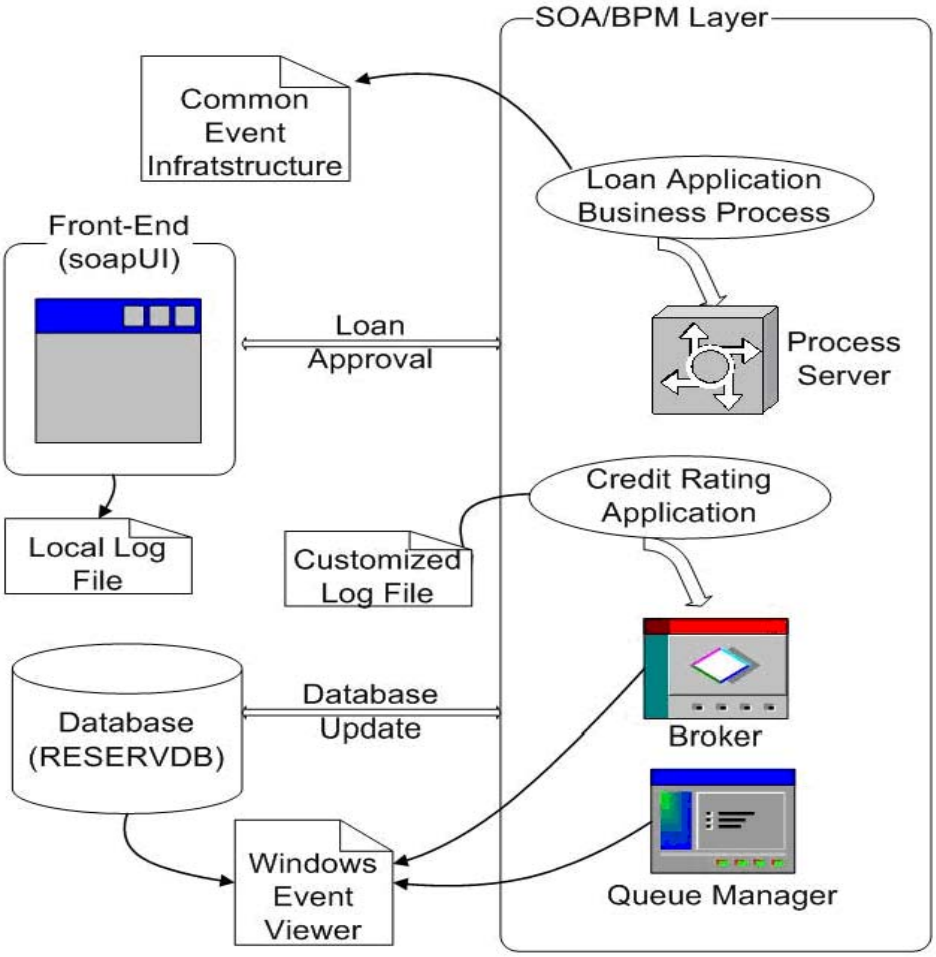


Figure 3-3: Test Environment Layout

3.5 The Framework in Action

Using the test environment described in Section 3.4, we ran a set of experiments to describe the framework in action. This is achieved by the injection of common types of software faults that occur in distributed systems into the monitored environment and then applying the RCA framework to discover the root causes.

Below, we describe the experimental evaluation of the RCA framework using the test setup described earlier. This test scenario consists of injecting a set of faults into the loan application test environment, leading to failures in the *Apply_For_Loan* business process execution. A quantitative study in trends of software faults and failures by Hamill et al. suggests that most failures are caused by faults in requirements, coding, and data [10]. Requirements and coding faults each contribute to a third of the total number of failures.

We injected three types of faults. The first injected fault is a missing requirement that was supposed to enforce an early check on the loan applicant to make sure that he or she is a client of the bank before processing the application. The missing requirement leads to a failure when the

system fails to locate the applicant's information in the database. The second injected error is a coding error in which the database table name is incorrectly spelled. The third injected error is a data error in which the applicant ID is alphanumeric and the system expects a numeric ID field. Due to space reasons, we describe only the coding fault injection case in detail.

The starting point of analysis in the RCA framework is to investigate the system that generated the raised alert. The injected error code (misspelled database table) caused the failure in processing loans. Exception handling was automatically applied to the execution of the business process. The analysis requires parsing of the goal model of the loan business process into memory as well as querying the related log data that captured the events of interest.

The next step is to obtain log data that can show the occurrence of the events that correspond to the goals' and tasks' preconditions, occurrences, and effects. Log data is extracted from the stream database and parsed into memory. The log data extraction is based on the generated queries. The queries are first developed by using the log abstraction techniques described in Section 3.3.3. The outcome of the log abstraction step is a collection of abstract events that represents all types of events that occur in this distributed environment. An example query that represents a loan business process instance completion is shown below:

```
CONTEXT = '10.1.10.137; LoanProject; ApplyForLoan'
AND EVENT_NATURE = 'EXIT'
AND SITUATION_QUALIFIER Like '%COMPLETED'
```

A mapping between events of the tasks or goals of the goal model and the abstract events generated earlier is manually created in advance of any RCA analysis. Each event in the goal model is represented by one or more abstract events. Next, the queries representing the abstract event types are augmented with user points of interest (such as time, user name, process ID, etc.). An example of an augmented query is shown below:

```
TIMESTAMP = 2009-08-11T21:21:22.000Z
AND CONTEXT = 10.1.10.137 ; LoanProject ; ApplyForLoan
AND EVENT_NATURE = EXIT
AND SITUATION = SUCCESSFUL
AND SITUATION_QUALIFIER Like %COMPLETED
```

The augmented query is then applied to the database to extract log data. If the query application returns no log data, we consider that the corresponding event did not occur. On the other hand, if the query application does return log data, we consider it evidence that the event of interest did occur. The log data is then transformed into logical statements that show whether these events have or have not occurred during the time frame that the system administrators have specified in their points of interest.

The interpreted log data generated by the *Apply_For_Loan* business process is shown below:

```
WS1_Sub(1); BP_Idle(2); occ(3, 2); WSReq_Sub(3); occ(4, 3); BP_St(3);
BPLoan_St(4); ClntInf_Vld(4); occ(6, 4); Prpre_CR_Rq(5); occ(7, 5); !
Rcvd_CR_Rpl(6);
! Valid_CR(7); ! CR_Avail(8); ! Decision_Done(9); occ(11, 9); !
Reply_Gen(10); occ(12, 10); ! Reply_Sub(11);
```

The diagnosis produced by the SAT solver shows that the top goal has not been satisfied. The diagnostics for the business process model test run results in the following diagnosis:

```
Diagnosis 1: [fd(7), fd(11), fd(12)]
```

This diagnosis shows that tasks a_6, a_{11}, and a_{12} have been denied. Tasks a_{11} and a_{12} represent the business process tasks of *Prepare_SOAP_Reply* and *Make_a_Decision_Based_on_CR*. Task a_6 (*Get_Credit_Rating*) is implemented by the *Credit_Rating* service at the service layer. We reapply the RCA framework using the goal models and log data of the credit rating service. The goal model of the *Credit_Rating* service is parsed, and the corresponding log file data is shown below:

```
ClntInf_Vld(1); CR_Idle(2); occ(3, 2); CR_St(3); CR_Rq_Sub(3); occ(4, 3);
Rcvd_CR_Valid(4); CR_DB_Idle(4); occ(6, 4); CR_DB_Av(5); ID_avail(5);
occ(7,5); ! CR_Ret(6); ! CR_Avail(6); ! occ(9, 6); ! DB_Update(7); !_CR
Avail(7); ID_Avail(7); occ(10, 7); ! CR_Rpl_Ready(8); occ(11, 8); !
Sub_CR_Rpl(9);
! Rcvd_CR_Rpl(9);
```

The corresponding diagnosis produced by the monitoring framework is shown below:

```
Diagnosis 1: [fd(7), fd(10), fd(11)]
```

The complete set of diagnoses for tasks that span across the three layers of abstraction (business process, service, and infrastructure layers) includes:

- a_3 (Prepare_SOAP_Reply)
- a_4 (Make_a_Decision_Based_on_Credit_Rating)
- a_6 (Get_Credit_Rating)
- a_{11} (Prepare_SOAP_Reply)
- a_{12} (Submit_SOAP_Reply)
- a_{15} (Retrieve_Credit_Rating)

In fact, we know that a_{15} is the real root cause for this problem, which verifies the soundness of our framework. Another point of interest to indicate during the evaluation of the proposed framework is the number of log entries that are generated during each transaction. For one particular loan business process execution, the following set of events is generated by the monitored systems:

- 11 CBE events generated by the business process *Apply_For_Loan*
- 37 log entries in the *soapUI* log generated by the *soapUI* toolkit for each web service invocation for the business process *Apply_For_Loan*
- 3 log entries in the credit rating service application deployed on the ESB runtime

The total number of events generated during a certain time frame is a factor of the number of events generated per process instance multiplied by the total number of executions. In a typical large enterprise environment, the number of events can range from thousands to millions per day.

Finally, in our experimental approach, we have manually executed the steps of log abstraction, query generation, and mapping from log types to events of interest. In addition, we have not used

an actual stream database but manually parsed the log data to the log format expected by our diagnosis framework. The diagnosis part of our framework was done using a Java implementation by Wang et al. [27].

3.6 Conclusions and Future Work

This paper presented a framework for RCA in SOA environments. The framework builds on the work proposed by Wang et al. [27]. First, given a collection of alerts, hypotheses are generated by examining goal trees related to the functional or nonfunctional requirement that is observed to have failed. The hypotheses are generated by considering possible paths in the goal tree that support the observed failure of the root goal node of the tree. Second, a temporal query language is proposed that facilitates the generation of queries that are used to extract log data and verifies the occurrence of events associated with the component being diagnosed. Apart from proposing an extension and offering algorithms for each element of the extension, we also present experimental results that demonstrate the feasibility of the proposal. Initial results show that the proposed framework is scalable when applied to SOA systems. However, more experimentation is required to identify the performance of the framework when more complex systems, multiple failures, or incomplete data are involved.

The empirical evaluation in Section 3.5 discusses the handling of three types of faults: requirements, coding, and data. In fact, the RCA framework can be applied to perform root cause defect analysis. In this respect, the RCA framework can be applied as part of regression testing to detect any introduced bugs. For new software versions, the corresponding goal models can be updated with the new requirements that were added as part of the new release, and the RCA framework can be applied on the new system, thus contributing to the software maintenance and evolution life cycle.

The process of collecting log data and using it for RCA as proposed in this paper can be automated and thus can be readily used to detect system errors in a near real-time fashion. Prior to applying the framework, there is manual preparatory work of developing the goal models and assigning queries to capture the conditions for the individual goals or tasks to occur. In our test scenarios, we have prepared these queries, collected the log data, and later injected errors as described in Section 3.5. Our knowledge of the log data and the type of errors that we inject causes a bias in the query formulation. However, this bias is accepted as part of the preparatory process of the framework. In fact, to successfully deploy the proposed framework in a typical enterprise environment, we expect the query formulation process to go through an iterative process of training in which typical errors are injected, the corresponding collected data is inspected, and queries are updated until satisfactory results are achieved.

More specifically, in a typical distributed environment, the source of failures can often be attributed to a collection of independent faults with varying degrees of impact rather than to a single fault. One way to handle this issue is to assign weights to the generated hypotheses and consider them independently for analysis and evaluation. An example in an SOA environment is when the average response time of a business process increases, which could be caused by multiple business processes using common services and data sources. The degradation of the performance of one process can be caused by a combination of reasons, such as database response time degradation or increased demand on another process that is putting pressure on shared

services. Future work in this area may include the extension of the diagnostic framework to consider the identification of cases in which multiple faults are involved. The other type of future extension to this work is to handle the case in which limited or no events are available to verify the occurrence or deniability of tasks. For this, we could consider probabilistic reasoning instead of the deterministic reasoning.

References

[1] J. H. Andrews and Y. Zhang. Broad-Spectrum Studies of Log File Analysis. In ICSE '00: Proceedings of the 22nd International Conference on Software Engineering, pp. 105–114, New York, NY, USA, 2000. ACM.

[2] BEA. Wldf Query Language. http://download.oracle.com/docs/cd/E12840_01/wls/docs103/wldf_configuring/appendix_query.html (2009).

[3] M. Brauer. Dresden ocl2 Toolkit. http://dresden-ocl.sourceforge.net (2010).

[4] S.-B. Cho. Incorporating Soft Computing Techniques into a Probabilistic Intrusion Detection System. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 32(2):154–160, 2002.

[5] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. Communications of the ACM, 5(7):394–397, 1962.

[6] D. E. Denning. An Intrusion-Detection Model. IEEE Transactions on Software Engineering, 13(2):222–232, 1987.

[7] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with Goal Models, pp. 167–181. Springer, 2002.

[8] L. Golab and M. T. Ozsu. Issues in Data Stream Management. SIGMOD Record, 32(2):5–14, 2003.

[9] E. Goldberg and Y. Novikov. Berkmin: A Fast and Robust SAT-Solver. Discrete Applied Mathematics, 155(12):1549–1561, 2007.

[10] M. Hamill and K. Goseva-Popstojanova. Common Trends in Software Fault and Failure Data. IEEE Transactions on Software Engineering, 35(4):484–496, 2009.

[11] A. Hanemann. A Hybrid Rule-Based/Case-Based Reasoning Approach for Service Fault Diagnosis. In AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications, pp. 734–740, Washington, DC, USA, 2006. IEEE Computer Society.

[12] IBM. Common Base Event. http://www.ibm.com/developerworks/library/specification/ws-cbe (2009).

[13] IBM. Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf (2009).

[14] W. Jiang, C. Hu, S. Pasupathy, A. Kanevsky, Z. Li, and Y. Zhou. Understanding Customer Problem Troubleshooting from Storage System Logs. In FAST '09: Proceedings of the 7th Conference on File and Storage Technologies, pp. 43–56, Berkeley, CA, USA, 2009. USENIX Association.

[15] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver, pp. 530–535, 2001.

[16] J. Mylopoulos, L. Chung, and B. Nixon. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. IEEE Transactions on Software Engineering, 18(6):483–497, 1992.

[17] S. Nadi, R. Holt, I. Davis, and S. Mankovskii. Draca: Decision Support for Root Cause Analysis and Change Impact Analysis for CMBDS. University of Waterloo, CA Labs, Canada, 2009.

[18] OMG. Object Constraint Language, version 2.0. http://www.omg.org/spec/OCL/2.0 (2006).

[19] L. Ryan. Efficient Algorithms for Clause-Learning SAT Solvers. M. Eng. Thesis, Simon Fraser University, 2004.

[20] S.-P. Shieh and V. D. Gligor. On a Pattern-Oriented Model for Intrusion Detection. IEEE Transactions on Knowledge and Data Engineering, 9(4):661–667, 1997.

[21] SLCT. Simple Log Clustering Tool (SLCT). http://ristov.users.sourceforge.net/slct (2009).

[22] soapUI. soapUI 2.0.2. http://www.soapui.org (2009).

[23] M. Steinder and A. S. Sethi. Probabilistic Fault Diagnosis in Communication Systems through Incremental Hypothesis Updating. Computer Networks, 45(4):537–562, 2004.

[24] R. Vaarandi. A Data Clustering Algorithm for Mining Patterns from Event Logs. In IEEE IPOM03 Proceedings, pp. 119–126, 2003. IEEE Computer Society.

[25] A. Van Lamsweerde, R. Darimont, and P. Massonet. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. In RE '95: Proceedings of the Second IEEE International Symposium on Requirements Engineering, p. 194, Washington, DC, USA, 1995. IEEE Computer Society.

[26] Y. Wang, S. A. McIlraith, Y. Yu, and J. Mylopoulos. An Automated Approach to Monitoring and Diagnosing Requirements. In ASE '07: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, pp. 293–302, New York, NY, USA, 2007. ACM.

[27] Y. Wang, S. A. McIlraith, Y. Yu, and J. Mylopoulos. Monitoring and Diagnosing Software Requirements. Automated Software Engineering, 16(1):3–35, 2009.

[28] Y. Yu, Y. Wang, J. Mylopoulos, S. Liaskos, A. Lapouchnian, and J. C. S. do Prado Leite. Reverse Engineering Goal Models from Legacy Code. In Proceedings of the 13th IEEE International Conference on Requirements Engineering, pp. 363–372, La Sorbonne, France, 2005. IEEE Computer Society.

[29] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. Automated Known Problem Diagnosis with Event Traces. In EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, pp. 375–388, New York, NY, USA, 2006. ACM.

[30] H. Zawawy, K. Kontogiannis, and J. Mylopoulos. Log Filtering and Interpretation for Root Cause Analysis. In ICSM '10: Proceedings of the 26th IEEE International Conference on Software Maintenance, Timişoara, Romania, 2010. IEEE Computer Society.

4 SOA Integration as an Alternative to Source Migration

Harry M. Sneed, ANECON GmbH, Austria (harry.sneed@t-online.de)

Abstract

In the past, it was common to migrate existing applications implemented in different languages into a common programming language in order to integrate them. In a service-oriented architecture (SOA), this is no longer necessary. Diverse components can be integrated into a common whole without changing the language. Only their communication interfaces and their data types have to be altered. This paper describes how this can be done with a minimum of cost and time. It then presents a case where this approach has been applied successfully.

4.1 The Migration Approach

In the past, it has been common for users to migrate their existing applications in order to integrate them [1]. If the legacy systems happened to be implemented in Assembler, COBOL, PL/I, or some other fourth-generation language, and the newer systems were implemented in Java, then the legacy systems would be first converted to Java and then integrated with the new Java systems. This has been a common approach to moving from a mainframe computer to distributed Unix or Linux computers [2]. The main rationale for proceeding in this way was not so much the incompatibility of the code, but the incompatibility of the data. Different languages have different data types that do not map to each other, which prohibits the easy exchange of data between distributed components [3].

In addition, the components implemented in different languages also have different communication mechanisms. COBOL and PL/I running on an IBM mainframe will normally be using either the customer information control system (CICS) or the information management system (IMS) to communicate with the user. In the case of CICS, the user programs are subprograms running under the CICS monitor, which invokes them and passes them the data from the user interfaces. In the case of IMS, it is exactly the opposite. The user program invokes the IMS routines to provide services like turning over the contents of the next user panel or sending a message. Fourth-generation languages like Natural and CSP have their own communication framework and their own commands for passing data back and forth to the user. It is not possible to integrate such totally different architectures into a common service architecture. This is a flagrant example of architectural mismatch [4].

To overcome these barriers to integration, the preferred approach of the past has been to migrate the legacy systems into a common language, such as Java, with a common framework, such as J2EE. This entails a lot of rework because not only do the procedural commands have to be translated but also the data types must be translated. On top of that, the architectural framework has to be exchanged. As a result, migration projects turn out to be very costly and carry a high risk. In addition, the results of a conversion are seldom better than what went into the conversion and are sometimes worse—at least from the point of view of the new language. For this reason,

users are often tempted to go ahead and re-implement the existing systems, but that opens them up to even higher risks and costs that cannot be controlled. In summary, migration is a dangerous and costly undertaking with questionable results [5].

If the old code is acceptable and there are still enough programmers around to maintain it, there must be a better way to integrate the old and the new applications in a common framework running on a common platform. This integration would achieve what the managers are striving for, namely, to unify their environment and to save the costs of sustaining so many diverse platforms. The answer lies in unifying the legacy data and getting rid of the legacy communication framework.

4.2 The Goals of IT Management

One of the least understood subjects in academic research of software migration is that of IT management motivation. What is it that motivates managers to change languages or to move to another platform? Academics would like to think that the management decisions are quality driven, but that is seldom the case. The main motivation is, as borne out by the investigation of more than 20 large-scale migration projects in middle Europe, to escape the dependency on a particular vendor [6].

The reasons for waiting to change vendors are varied:

- The vendor may no longer support the platform on which the user applications are running. In this case, the user has no choice but to leave it.
- The vendor may charge exuberant license fees that the user no longer wants to pay. This is often the case when the user is using some proprietary software, such as VisualAge, Natural/Adabas, or IMS. The vendor uses its monopoly over the software upon which the user is dependent to blackmail him or her, which is a common practice in the IT world.
- When companies with different vendors merge, they may also want to merge their IT systems and have only one vendor. To achieve this goal, they have to migrate their diverse IT systems on to a common platform.
- The user may have a dispute with his current vendor and wants to get away from the dependency on his software.
- In recent years, there has been a trend to get away from proprietary software and to migrate to an open environment in which software products can be easily exchanged. The main reason for migrating to Java is not because Java is such a great language and it is object oriented, but that Java will run in any environment. That appeals to the IT managers who are keen on being independent of vendors. They believe that, this way, they are not trapped into a particular environment.

Regardless of the reason, the decision to change the environment is often accompanied by a change of languages. If the user has implemented his or her systems in a fourth-generation language or in the Assembler language of a particular vendor, he or she has no other choice but to change languages if his or her goal is to escape the dependency on that vendor. A similar situation exists with PL/I, but there are now alternative PL/I compilers for PC and Unix machines. There are compilers for COBOL and FORTRAN on the PC as well as on Unix—but they cost money.

C++ and Java appear to be the only free universal languages that will run on any platform. There are also scripting languages such as PHP, Python, and Perl, but they are not usually used for business applications. Assuming that this is true, then a user intent on being independent of all vendors should convert his languages to either C++ or Java. If he or she is willing to accept a certain degree of dependency, then he or she might remain in COBOL or PL/I. C# is a convenient language, but it locks users into a proprietary .NET environment [7].

These are the major reasons for migration, although they are more of a political nature. The quality of the software is only a side issue. Therefore, no great importance is attached to the quality of the migration results. The fact is that many IT managers do not have the slightest notion of what software quality is. Their main goal is that the migrated software should be functionally equivalent to the original software and that its performance is not significantly worse. As pointed out above, the primary goal is to escape from a proprietary environment. The responsible technicians believe that the quality can always be improved once they are in the new environment. Unfortunately, they seldom get around to changing it, so the poor quality of the original code remains forever.

4.3 The Integration Approach

Today, it is no longer necessary to convert systems into a common language in order to integrate them. The only prerequisite is that there is a compiler for them on the server they are running on. If the user has different languages such as Java, COBOL, and PL/I systems, he or she might have a separate server for each language. The servers are connected via the enterprise bus as shown in Figure 4-1. The languages will be split into components, and each component can have its own web service interface [8].

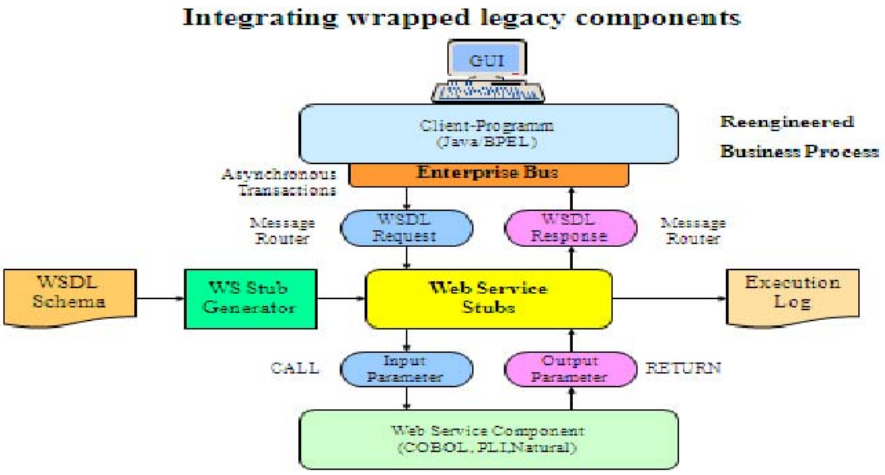


Figure 4-1: Building on Wrapped Components

The client business processes, implemented in business process execution language (BPEL) or some similar business process language, will be running on the client workstations. They will dispatch their requests to a router that knows what web service to invoke on what server. It does not matter what language the service is implemented in [9]. The problem comes up only when web services want to exchange data with one another or when they share a common database, as

the data types must be compatible. The COBOL programs will be using binary and packed data types. The PL/I programs will be using pointers, binary data, packed data, or even floating point data. Such exotic data types are unknown to Java as well as to XML. Thus, they must be cast every time a web request is received, a response is sent, or a database record is stored or retrieved. This makes up for a significant amount of processing time, causing a performance loss—which is something that IT managers notice immediately. The measurement of wrapped services has shown that processing time increases by a factor of 2.5 as compared to the original unwrapped procedures. Over 50 percent of this runtime increase is taken up by the data type conversion [10].

4.3.1 Unifying the Data for Ease of Integration

For this reason, we suggest that the data in the wrapped services be converted only once, when the code is wrapped. The code has to be processed anyway in order to modify the component interfaces. At the same time, all of the special data types can be converted to ASCII character format within the program. The idea is to convert the data types before wrapping the code. Thus, an additional step is added to the wrapping process, as shown in Figure 4-2.

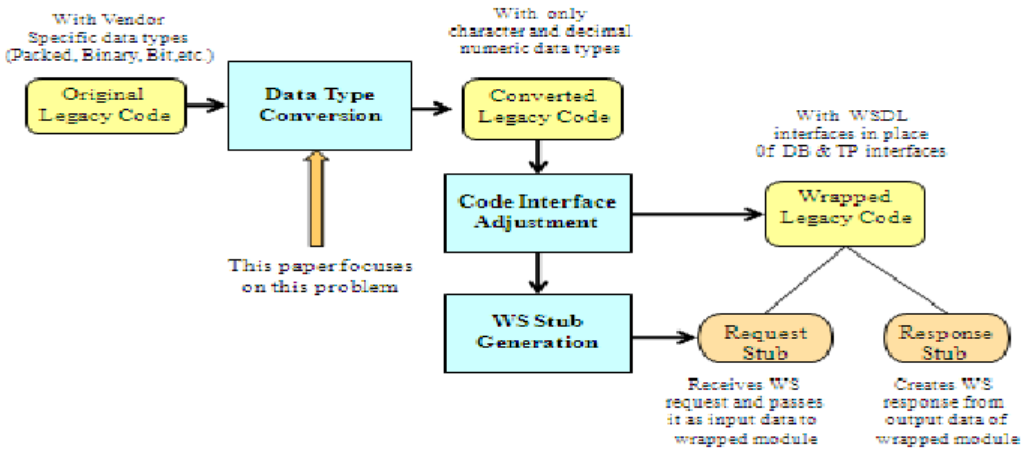


Figure 4-2: The Wrapping Process

In COBOL, the following transformations can be made:

- packed decimal is converted to zoned decimal
- floating decimal is converted to zoned decimal
- binary types are converted to zoned decimal
- bit is converted to character
- pointer is converted to integer long
- index is converted to integer short

In PL/I, the transformations are similar. In fourth-generation languages, there may be other data types to deal with, such as the descriptor in Natural. In any case, the result should be a program that processes data only in ASCII character format. Because the type change will cause the fields to be longer, redefined fields and structures have to be adjusted accordingly. If the redefined data structure becomes longer, the redefining data has to be padded. If the redefined data structure

becomes shorter, the redefined data has to be padded. The matching of overlaid data fields is one of the greatest sources of error involved here, so it has to be validated [11].

This type of one-time data conversion is not easy to program and test, but it is well worth the effort. By converting all data to ASCII character format, the processing of web service requests and the creation of web service responses become much easier. They also become at least two times quicker. The same applies to the storage and retrieval of database records. All in all, the web service components become more unified.

4.3.2 Converting the Database

Having converted the data types in the wrapped components, it will then be necessary to convert the various databases into a common unified database. This database should know only character-type fields. Even the dates should be in character format, which will avoid any date conversion problems.

The conversion of data types in the source code is done statically by changing the source code. The conversion of data types in the databases has to be done dynamically by a data conversion utility. The old data tables or records are read in, the data physically converted to the new types, and the new tables or records written out. The basis for the database conversion is a table of data types that is created when the program source is adjusted. The result of this conversion will be databases that have only character-type attributes [12].

With this, we now have a unified set of systems using a unified database. Even though the web services are in different languages and their data are contained in different database systems, they will be able to easily exchange data with one another and with the processes that use them because all of their data is in the same format. This is a very important issue when it comes to system integration within a service-oriented architecture (SOA).

4.3.3 Freeing the Legacy Programs from Their Proprietary Communication Harness

The freeing of legacy code from a proprietary communication harness has been discussed in previous papers on migration [13]. Typical examples of such a harness are CICS and IMS from IBM and user-defined data type (UDT) from Siemens. Unisys and Bull also had their own unique frameworks. These so-called telecommunication monitors link the application code to the remote user workstations and control the execution of the online transactions. The control and input/output operations are built into the code in the form of EXEC macros, such as EXEC-CICS or EXEC-IMS. A preprocessor converts these macros into calls to the transaction processing (TP) monitor.

When wrapping such a program, it is necessary to comment out these EXEC macros and to replace them with calls to the wrapper. There are some macros that need only to be removed. Others, such as SEND and RECEIVE, have to be replaced. The RECEIVE macros—or in IMS the GET NEXT UNIQUE map—should be replaced by a call to get the next service request. The SEND macros—or the INSERT map in IMS—should be replaced by a call to send the current service response. The called modules are standard stubs linked to the target program. They not only parse the incoming Simple Object Access Protocol (SOAP) message to extract the data but also convert the XML data types into corresponding COBOL or PLI data types, which may be machine dependent—for instance, bit strings, binary numbers, and packed decimal numbers. In

the opposite direction, the stubs marshal the data into a SOAP response, converting the local data types back into the standard XML data types. This machine-dependent data conversion takes up two-thirds of the stub code.

By unifying the data types in the user programs, this time-consuming runtime data conversion can be left out. The stubs need only extract the request data and marshal the response data, as shown in Figure 4-3 [14].

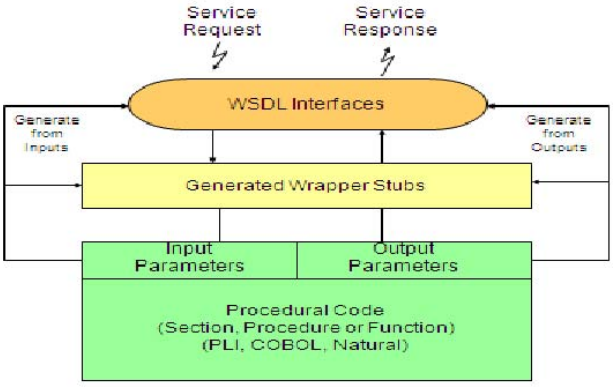


Figure 4-3: Wrapper Stubs as Connectors from WSDL to the Legacy Code

The author has developed wrapping tools that will make these source transformations for both CICS and IMS, as well as for Siemens' UDT programs. Recently, he has also made one for Honeywell/Bull. Their new TP monitors—for example, IBM's WebSphere—can substitute the 3270 maps with Web Services Description Language (WSDL) interfaces. They are interested in keeping the user dependence on them, so for users who want to escape from the dependency on a particular vendor, this is no solution. They must alter their source code to rid it of vendor-specific features.

4.4 A Case Study in Source-Level Wrapping

The case study described here involves a PLI/IMS program from a Swiss bank. The program is generating a background job. It receives its input data from an input file. It accesses an IMS job database, and it creates the JCL for the job. It then confirms that the job is generated by printing out a protocol.

In the first sample, it is possible to see how the data is converted to ASCII character format. The computational and packed fields are commented out and replaced by display numeric fields. There are no longer any specific IBM mainframe data types. The results of the conversion are included in Appendix A.

In a second step, a WSDL schema is generated from the converted data structure. Here only the character and picture data types remain. All data is in ASCII character format, so there is no need for any conversion. The converted program will work with the data from the web request just as it is. The same applies to the web requests. (See Appendix B.)

The wrapper uses the WSDL schema generated from the PLI/IMS source to produce two stubs, one for transferring the data from the web request to the storage of the PLI program and one for transferring the data from the PLI storage to the web response. With these source code modifications, the sample PLI program will now run in any environment that has a PLI compiler without having to convert the data at runtime.

Thus, with the aid of the tool CodeWrap, both COBOL and PL/I programs can be converted over to web services that will execute on any UNIX, LINUX, or Windows platform. Source-level system integration has been achieved, giving IT managers who are keen on being vendor independent the possibility of obtaining that goal without the high cost of a language conversion.

4.5 Summary and Future Work

In this paper, the integration of existing legacy systems into a SOA is suggested as an alternative to system migration. There is no need to convert existing languages, with all the cost and risk involved, when those languages can be ported to an SOA platform just as they are. It is true that the original quality of the code remains, but no one of importance is interested in that anyway. The goal is to reuse existing programs as web services with a minimum of change. The prerequisites are to free the code from the legacy communication harness and to convert their data to a common, exchangeable data format. This can be achieved with some minor code modifications. The business logic as expressed in the legacy language is left as it is [15].

The focus of future work is to collect more data on the feasibility of this approach and to test it in real projects. Current mainframe users must be convinced that this is an alternative for them. The only way to achieve that is through empirical evidence.

References

[1] Seacord, R., Plakosh, D., and Lewis, G. Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices, Addison-Wesley, Boston, 2003.

[2] Allen, P. “Practical Strategies for Migrating to Distributed Components,” Cutter IT Journal, Vol. 11, No. 12, Dec. 1998, p. 22.

[3] Henrard, J., Thiran, P., and Hamut, J. “Strategies for Data Reengineering,” Proc. of 9th WCRE, IEEE Computer Society Press, Richmond, Oct. 2002, p. 211.

[4] Garlan, D., Allan, R., and Ockerbloom, J. “Architectural Mismatch—Why Reuse Is so Hard,” IEEE Software Magazine, Nov. 1995, p. 26.

[5] Sneed, H. “Risks Involved in Reengineering Projects,” Proc. of WCRE-1999, IEEE Computer Society Press, Atlanta, Oct. 1999, p. 204.

[6] Sneed, H., Wolf, E., and Heilmann, H. Software Migration in Practice, dpunkt Verlag, Heidelberg, 2010, p. 240.

[7] Terekhov, A. and Verhoef, C. “The Realities of Language Conversions,” IEEE Software, Dec. 2000, p. 111.

[8] Sneed, H. “Encapsulation of Legacy Software: A Technique for Reusing Software Components,” Annals of Software Engineering, Vol. 9, Baltzer A.G., Amsterdam, 2000, p. 113.

[9] Sneed, H. “Integrating Legacy Software into a Service-Oriented Architecture,” Proc. of 10th European Conference on Software Maintenance and Reengineering, IEEE Computer Society Press, Bari, Mar. 2006, p. 3.

[10] Sneed, H. “Certification of Web Services,” Proc. of 2nd Workshop on Evaluation of Service-Oriented Architectures, Shaker Verlag, Karlsruhe, Nov. 2007, p. 1.

[11] Ceccato, M., Dean, T., Tonella, P., and Marchignoli, D. “Migrating Legacy Data Structures Based on Variable Overlay to JAVA,” Journal of Software Maintenance and Evolution, Vol. 22, No. 3, Apr. 2010, p. 211–237.

[12] Hamut, J.-L., Thiran, P., and Houben, G. “Database Wrapper Development—Towards Automatic Generation,” Proc. of 9th European Conference on Software Maintenance Reengineering, IEEE Computer Society Press, Manchester, Mar. 2005, p. 207.

[13] Sneed, H. “Program Interface Reengineering for Wrapping,” Proc. of 4th WCRE, IEEE Computer Society Press, Amsterdam, Oct. 1997, p. 201.

[14] Sneed, H. “Experience in Extracting Web Services from Legacy Code,” Proc. of 8th Workshop on Web Service Evolution (WSE), IEEE Computer Society Press, Beijing, Sep. 2008, p. 72.

[15] Sneed, H. “Migrating to Web Services—A Research Framework,” Proc. of SOAM Workshop, CSMR-2007, Amsterdam, March 2007, p. 3.

Appendix A: PL/I Data Type Conversion

Original PLI Data Types

```
DCL SUCHKEY         CHAR(04)        INIT('') ;
DCL LINECOUNTER     BIN FIXED(15,0) INIT('') ;
DCL COUNTER         DEC FIXED(15,0) INIT('') ;
DCL TAWERT_Z        PIC 'Z.ZZZ.ZZZ.ZZ9V,99' ;
DCL TAWERT_NEU      CHAR(16) ;
DCL BSTKNOM         FLOAT(16)   INIT(0) ;
DCL BNOMSFR         FLOAT(16)   INIT(0) ;
DCL H1              FLOAT(16)   INIT(0) ;
DCL H2              FLOAT(16)   INIT(0) ;
DCL H3              FLOAT(16)   INIT(0) ;
DCL STEUER_VALUE    PIC '(1)9'  INIT('') ;
DCL LOOP_ERROR_COUNT      BIN FIXED (15,0) INIT(0) ;
DCL AKTUELLE_AKTSTUFE     CHAR(1) INIT('') ;
/* OPERATOR FUER DLI-ZUGRIFFE           */
DCL GLOBAL_OP       CHAR(2) INIT(' =') ;
/* ZAEHLERFELDER FUER LISTEN_OUTPUT     */
DCL VALOREN_I       DEC FIXED (5) INIT(0) ;
DCL VALOREN_O       DEC FIXED (5) INIT(0) ;
DCL BESTAND_I       DEC FIXED (5) INIT(0) ;
DCL BESTAND_O       DEC FIXED (5) INIT(0) ;
DCL TABELLEN_I      DEC FIXED (5) INIT(0) ;
DCL TABELLEN_O      DEC FIXED (5) INIT(0) ;
DCL MARCHZINSEN             DEC FIXED (11,2) ;
DCL MARCHZ_MELDUNG          BIT (1)         INIT('0'B) ;
DCL PARM_FAELL              CHAR    (10)    INIT('') ;
DCL PARM_SICHERST           PIC     '(8)9'  INIT('') ;
DCL PARM_LZPERB             CHAR    (10)    INIT('') ;
DCL PARM_LZPERV             CHAR    (10)    INIT('') ;
DCL PARM_EVERF              CHAR    (10)    INIT('') ;
DCL PARM_STKNOM             DEC     (13,2)          ;
DCL PARM_ZSATZ              DEC     (7,5)           ;
DCL PARM_WAEK               PIC     '999V9999'      ;
DCL PARM_WAEE               PIC     '999'           ;
DCL PARM_USANZ              CHAR    (1)     INIT('') ;
DCL INDEX_GEFUNDEN          BIT (1)         INIT('0'B) ;
DCL I_COUNTER               BIN FIXED (15)  INIT(0) ;
DCL AUSWAHL_PTR PTR ;
DCL CONVERSIONS_FELD_1      CHAR(11) INIT('') ;
DCL CONVERSIONS_FELD_2      CHAR(11) INIT('') ;
DCL CONVERSIONS_FELD_3      CHAR(12) INIT('') ;
DCL KEY_FELD                CHAR(33) INIT('') ;
```

Converted PLI Data Types

```
DCL SUCHKEY         CHAR(04)        INIT('') ;
DCL LINECOUNTER     PIC '9(04)'     INIT('')   ;         /*CONVERTED*/
DCL COUNTER         PIC '9(15)'     INIT('') ;           /*CONVERTED*/
DCL TAWERT_Z        PIC 'Z.ZZZ.ZZZ.ZZ9V,99' ;
DCL TAWERT_NEU      CHAR(16) ;
DCL BSTKNOM         PIC '9(16)' INIT(0) ;                /*CONVERTED*/
DCL BNOMSFRS        PIC '9(16)' INIT(0) ;                /*CONVERTED*/
DCL HFLOAT_1        PIC '9(16)' INIT(0) ;                /*CONVERTED*/
DCL HFLOAT_2        PIC '9(16)' INIT(0) ;                /*CONVERTED*/
DCL HFLOAT_3        PIC '9(16)' INIT(0) ;                /*CONVERTED*/
DCL STEUER_VALUE    PIC '(1)9'  INIT('') ;
DCL LOOP_ERROR_COUNT      PIC '9(04)'     INIT(0)   ;    /*CONVERTED*/
DCL AKTUELLE_AKTSTUFE     CHAR(1) INIT('') ;
/* OPERATOR FUER DLI-ZUGRIFFE           */
DCL GLOBAL_OPERATOR CHAR(2) INIT(' =') ;
/* ZAEHLERFELDER FUER LISTEN_OUTPUT     */
DCL VALOREN_I       PIC '9(05)'   INIT(0) ;              /*CONVERTED*/
DCL VALOREN_O       PIC '9(05)'   INIT(0) ;              /*CONVERTED*/
DCL BESTAND_I       PIC '9(05)'   INIT(0) ;              /*CONVERTED*/
DCL BESTAND_O       PIC '9(05)'   INIT(0) ;              /*CONVERTED*/
DCL TABELLEN_I      PIC '9(05)'   INIT(0) ;              /*CONVERTED*/
DCL TABELLEN_O      PIC '9(05)'   INIT(0) ;              /*CONVERTED*/
DCL MARCHZINSEN             PIC '9(11).9(2)' ;           /*CONVERTED*/
DCL MARCHZ_MELDUNG          CHAR(1)         INIT('0') ;;
DCL PARM_FAELL              CHAR    (10)    INIT('') ;
DCL PARM_SICHERST           PIC     '(8)9'  INIT('') ;
DCL PARM_LZPERB             CHAR    (10)    INIT('') ;
DCL PARM_LZPERV             CHAR    (10)    INIT('') ;
DCL PARM_EVERF              CHAR    (10)    INIT('') ;
DCL PARM_STKNOM             PIC '9(13).9(2)'         ;  /*CONVERTED*/
DCL PARM_ZSATZ              PIC '9(07).9(5)'         ;  /*CONVERTED*/
DCL PARM_WAEK               PIC     '999V9999'       ;
DCL PARM_WAEE               PIC     '999'            ;
DCL PARM_USANZ              CHAR    (1)     INIT('') ;
DCL INDEX_GEFUNDEN          CHAR(1)         INIT('0') ;;
DCL I_COUNTER               PIC '9(04)'     INIT(0)   ; /*CONVERTED*/
DCL AUSWAHL_PTR PTR ;
DCL CONVERSIONS_FELD_1      CHAR(11) INIT('') ;
DCL CONVERSIONS_FELD_2      CHAR(11) INIT('') ;
DCL CONVERSIONS_FELD_3      CHAR(12) INIT('') ;
DCL KEY_FELD                CHAR(33) INIT('') ;
```

Appendix B: WSDL Interface Conversion

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org.wsdl/soap/"
    name = "XXXXXXXX" targetNamespace ="PLIServices">
    <!--This schema was generated from prog:WrapOut\P2715PLI.pli by the Sneed
        Tool GENSCHEMA on date:100811 -->
    <types>
        <schema name = "P2715PLI"
            xmlns="http://www.w3.org/2001/WSDLSchema"
            xmlns:"soap11-enc=http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            targetNamespace="PLIServices">
    <XSDPLI:complexType type = "#file" name = "P2715IN"
              content = "eltOnly" model = "closed">
              <XSDPLI:element type = "#char" name = "SUCHKEY"
                 content = "TextOnly" model = "closed" level = "02"
                 occurs = "1" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0000" lng = "0004" pic = "X(4)" usage = "DISPLAY"/>
              <XSDPLI:element type = "#char" name = "LINECOUNTER"
                 content = "TextOnly" model = "closed" level = "02"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0004" lng = "0004" pic = "9(4)" usage = "DISPLAY"/>
              <XSDPLI:element type = "#char" name = "COUNTER"
                 content = "TextOnly" model = "closed" level = "02"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0008" lng = "0015" pic = "9(15)" usage = "DISPLAY"/>
              <XSDPLI:element type = "#char" name = "TAWERI_Z"
                 content = "TextOnly" model = "closed" level = "02"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0023" lng = "0016" pic = "Z.ZZZ.ZZZ.ZZ9V,99"
                 usage = "DISPLAY"/>
              <XSDPLI:element type = "#char" name = "TAWERI_NEU"
                 content = "TextOnly" model = "closed" level = "02"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0039" lng = "0016" pic = "X(16)" usage = "DISPLAY"/>
            </XSDPLI:complexType>
            <element name = "P2715PLI" type = "XSDPLI:PLIPROC"/>
        </schema>
    </types>
</definitions>

<!--WSDL/SOAP Interface Definition made by the Sneed Tool GENWSDL -->
<message name = "INPUT.P2715PLI" type = "XSDPLI:PLIPROC">
    <part name = "response" element="P2715PLI"/>
</message>
<portType name = "INPUT_Interface">
    <operation name = "P2715PLI">
        <input message = "XSDPLI:P2715PLI"/>
        <output message = "XSDPLI:P2715PLI"/>
        <fault name = "WSExceptionHandler"
            message = "XSDPLI:WSExceptionHandler"/>
    </operation>
```

```
</portType>
<binding name = “INPUT_Interface_Binding” type = “XSDPLI:INPUT”>
   <soap:binding transport=“http://schemas.xmlsoap.org/soap/http“
      style=“document”/>
   <operation name = “P2715PLI”>
      <soap:operation soapAction = “ ”/>
      <input>
         <soap:body use = “literal”>
      </input>
      <output>
         <soap:body use = “literal”>
      </output>
   </operation>
</binding>
```

5 Toward Proactive Adaptation: A Journey along the S-Cube Service Life Cycle

Andreas Metzger, University of Duisburg-Essen, Germany (andreas.metzger@paluno.uni-due.de)
Eric Schmieders, University of Duisburg-Essen, Germany (eric.schmeiders@paluno.uni-due.de)
Cinzia Cappiello, Politecnico di Milano, Italy (cappiell@elet.polimi.it)
Elisabetta Di Nitto, Politecnico di Milano, Italy (dinitto@elet.polimi.it)
Raman Kazhamiakin, FBK-Irst, Italy (raman@fbk.eu)
Barbara Pernici, Politecnico di Milano, Italy (pernici@elet.polimi.it)
Marco Pistore, FBK-Irst, Italy (pistore@fbk.eu)

Abstract

Service-oriented applications are deployed in highly dynamic and distributed settings. Therefore, such applications are often equipped with adaptation capabilities to react to critical issues during their operation, such as failures, unexpected changes of third-party services, or context changes. In this paper, we discuss the shortcomings of current solutions for adaptive service-oriented applications. To address these shortcomings, we introduce techniques that can be utilized to build and evolve proactive applications. These techniques have been developed in S-Cube, the European Network of Excellence on Software Services and Systems. Proactive adaptation capabilities are considered to be particularly promising, as they can prevent costly compensation and repair activities. Using these techniques in an integrated way is described along the phases of the service life cycle. We use a running example to illustrate the shortcomings of current solutions for self-adaptation and to demonstrate the benefits of the S-Cube techniques.

5.1 Introduction

Service orientation is increasingly adopted as a paradigm for building highly dynamic, distributed, and adaptive software systems, called service-oriented (or service-based) systems. This paradigm implies a fundamental change to how software is developed, deployed, and maintained: a service-based system cannot be specified and realized completely in advance (i.e., during design time) due to the incomplete knowledge about the interacting parties (e.g., third-party service providers) as well as the system's context and communication infrastructure [1, 2]. Thus, as compared to traditional software engineering, many more decisions need to be made during the operation of the service-oriented system (i.e., after it has been deployed). For instance, those systems will need to react to failures of their constituent services (e.g., if a service provider fails to adhere to its contract) to ensure that they maintain their expected functionality and quality.

In such a dynamic setting, evolution and adaptation methods and tools become key in enabling those systems to respond to changing conditions. In accordance with the terminology defined by the S-Cube Network of Excellence, this paper differentiates between evolution and adaptation as follows: Evolution is considered to be the modification of the system's requirements, specification, models, and so forth during design time (also known as maintenance). In contrast, adaptation is considered the modification of a specific instance of a service-based system during operation [3]. In the current paper, we focus on adaptation due to malfunctioning of the system.

While the proposed techniques could also support general adaptation due to context changes, this is not discussed in the present paper.

5.1.1 Problem Statement and Related Work

Adaptive systems automatically and dynamically adapt to changing conditions. The aim of adaptation (also called self-adaptation) is to reduce the need for human intervention as much as possible. While the behavior of a nonadaptive system is controlled by only user input, adaptive systems consider additional information about the application and its context (e.g., failures of constituent services or different network connectivity). Thus, in order to realize self-adaptive behavior, methods and tools that implement control loops are established to collect details from the application and its context (e.g., by exploiting monitoring mechanisms) and decide and act accordingly [4].

So far, the majority of the work on adaptation has been centered around reactive adaptation capabilities based on monitoring [5]. This means that adaptation is performed *after* a deviation or critical change has occurred. Such a reactive adaptation based on monitoring, however, has at least the following two important shortcomings [6, 7, 8]:

- It can take time before problems in a service-based system lead to monitoring events that ultimately trigger the required adaptation. One key trigger for an adaptation is when the service-based system deviates from its requirements (such as expected response time). If only those requirements are monitored, the monitoring events might arrive so late that an adaptation of the service-based application (SBA) is not possible anymore [9]. For instance, the system could have already terminated in an inconsistent state or taken more time than the expected response time.
- Reactive adaptation can become very costly, especially when compensation or rollback actions need to be performed. As an example, when using stateful (or conversational) services, the state of the failed service might need to be transferred to an alternative service [10].

Of course, one can monitor the individual services of an SBA and trigger an adaptation as soon as the service has failed, that is, violated its contract [11]. However, when using those techniques, it remains unclear whether the failure of this service could lead to a violation of the SBA's requirements. This means that there may be situations in which the SBA is adapted, although it would not have been necessary because the requirements might still have been met. Consider the following simple example: Although a service might have shown a slower response time as (contractually) expected, prior service invocations (along the workflow) might have been fast enough to compensate for the slower response of that service. Such unnecessary (or "false-positive") adaptations have the following shortcomings [6]:

- Unnecessary adaptations can lead to additional costs and effort that could be avoided. For instance, additional activities, such as service-level agreement (SLA) negotiation for the alternative services, might have to be performed or the adaptation may lead to a more costly operation of the SBA, for example, if a seemingly unreliable but cheap service is replaced by a more costly one.

- Unnecessary adaptations could be faulty (e.g., if the new service has bugs), which could lead to severe problems.

In summary, one key problem that needs to be solved to enable proactive adaptation is to determine whether the SBA might deviate from its requirements during its future operation.

5.1.2 Paper Contributions

This paper describes techniques developed in the S-Cube[1] Network of Excellence to determine deviations from requirements based on monitored failures. Previous publications have discussed proactive adaptation techniques that are mainly in isolation and confined to individual phases of the service life cycle [6, 7, 8, 12, 13]. A first, more integrated view on adaptation has been presented by Bucchiarone et al. [14]. However, the focus lay on reactive adaptation and on the design time activities that are needed to build adaptive service-based systems. In contrast, in this paper, we demonstrate how the techniques for determining proactive adaptation play together along the various life-cycle phases and how they can be jointly applied in a meaningful way. As a basis for our discussions, we employ the S-Cube service life-cycle model [14, 15, 16, 17]. In contrast to more traditional life-cycle models, this model considers the specifics of service-based systems, particularly those concerning evolution and adaptation.

The remainder of the paper is structured as follows. In Section 5.2, the S-Cube service life-cycle model is introduced. In Section 5.4, the S-Cube techniques that jointly allow building proactive service-based systems are discussed, differentiating between activities that are done during design time and activities that are done during the operation phase (runtime). This discussion is illustrated by an example from the e-Government domain, which is introduced in Section 5.3.

5.2 The S-Cube Service Life-Cycle Model

The life-cycle models for SBAs that have been presented in the literature (examples include SLDC, RUP for SOA, SOMA, and SOAD) are mainly focused on the phases that precede the release of software. Even in those cases in which they focus on the operation phases, they usually do not consider the possibility of SBAs adapting dynamically to new situations, contexts, requirement needs, service faults, and so forth [14, 18]. Specifically, the following aspects have not yet been considered in these life-cycle models:

- *Requirements elicitation and design for adaptation.* The requirements engineering phase includes the elicitation and documentation of the system's functional and quality requirements. In the dynamic setting of SBAs, not only the requirements toward the actual application logic need to be analyzed, designed, and developed, but the context in which the system is executed also needs to be understood [1]. Context changes can necessitate the adaptation of the SBA, for instance, if the SLA of a third-party service is violated. During design, the capabilities to observe, modify, and change the SBA at runtime need to be devised.

[1] http://www.s-cube-network.eu

- *Extended operation phase.* The operation phase is not only responsible for executing and monitoring the application. It also requires identifying the need for an adaptation of the system as well as the where and how to enact such an adaptation [1].
- *Continuous quality assurance.* Quality assurance has an impact on all aspects of the life cycle. Therefore, the quality characteristics that are to be assessed and ensured must be identified, starting from the requirements analysis phase. Due to the open nature and the dynamic contexts in which SBAs operate, quality properties that have a lifelong validity need to be "continuously" asserted [19]. For instance, in the case of third-party services, there is no guarantee that a service implementation eventually fulfills the contract promised (e.g., stipulated by an SLA), or it is usually not possible to model and thus assess the behavior of the underlying distributed infrastructure (such as the Internet) during design time.

The service life-cycle model envisioned by the S-Cube network aims at incorporating these aspects. The S-Cube service life-cycle model relies on two development and adaptation loops, which can be executed in an incremental and iterative fashion [14, 15, 16, 17]:

- The development and evolution loop (right side of Figure 5-1) addresses the classic development and deployment life-cycle phases, including requirements, design, construction, operations, and management.
- The operation and adaptation loop (left side of Figure 5-1) extends the classic life cycle by explicitly defining phases for addressing changes and adaptations during the operation of SBAs (see Section 5.2.2).

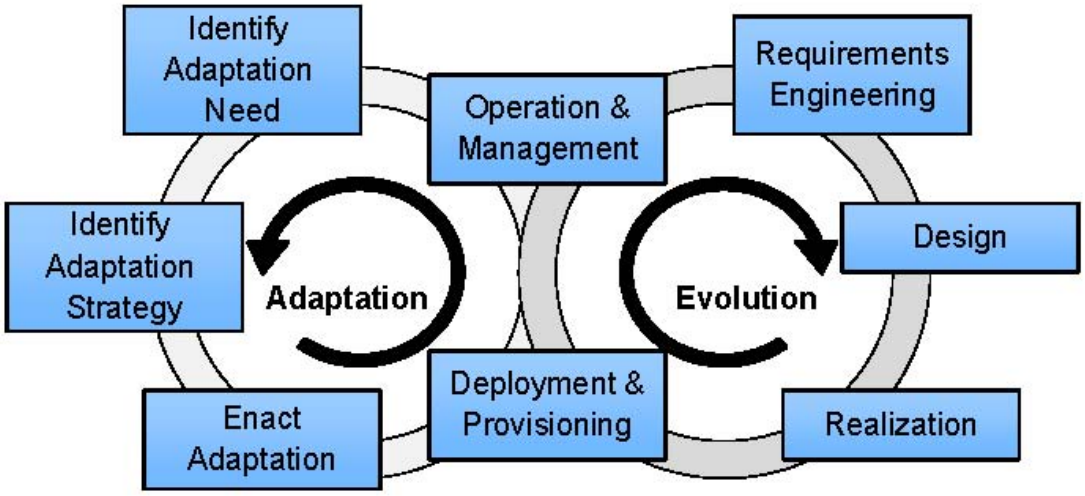


Figure 5-1: The S-Cube Service Life Cycle

5.2.1 Development and Evolution Cycle

Requirements Engineering. In the requirements engineering phase, the functional and quality requirements for the SBA are elicited and documented. The specifics of SBAs make the requirements engineering phase particularly relevant. This is related to the highly dynamic nature

of SBAs and to the need to guarantee the continuous adaptability and evolvability of these applications. Indeed, in a context where the application is in continuous evolution and is characterized by very blurred boundaries, the study of those requirements that exist a priori in the organizational and business setting, and that are hence largely independent from the solution, becomes very important.

Design. During the design phase, the activities and the control flow of the application are specified. In the service-oriented case, this usually means that a workflow is specified using languages such as BPEL. Along with the definition of the workflow, candidate services are identified that can provide the functionality and quality to fulfill the requirements of the SBA. This means that those services that cover, at least partially, the expected functionality and quality of service are identified. This is supported by service matchmaking techniques, such as the ones presented by Comuzzi and Pernici [20]. A further task in this phase is to define adaptation strategies and mechanisms that enable the application to react to adaptation needs [14].

Construction. After the design phase, the construction of the system is initiated. It is important to take into account that SBAs are created by the integration and coordination of services from different providers. In a specific way, this means that for establishing the desired end-to-end quality of those SBAs, contracts between the service providers and the service consumers on quality aspects of services have to be established. This typically requires some form of SLA negotiation and agreement. For each service, the best quality-of-service level within the available budget is negotiated with the providers of the candidate services that have been identified in the previous phase [20].

Deployment and Provisioning. The deployment and provisioning phase comprises all the activities needed to make the SBA available to its users. It should be noted that an SBA can itself be offered as a service.

5.2.2 Operation and Adaptation Cycle

Operation and Management. This phase specifies all the activities needed for operating and managing an SBA. The literature also uses the term "governance" to describe all activities that govern the correct execution of SBAs (and their constituent services) by ensuring that they provide the expected functionality and level of quality during operation. In this setting, the identification of problems in the SBA (e.g., failures of constituent services) and of changes in its context play a fundamental role. This identification is obtained by means of monitoring mechanisms, and more generally by exploiting techniques for runtime quality assurance, such as online testing or runtime verification. When used together, these mechanisms and techniques are able to detect failures or critical conditions.

Identify Adaptation Need. Some failures or critical conditions become triggers for the SBA to leave normal operation and enter the adaptation or evolution cycle. The adaptation cycle is responsible for deciding whether the SBA needs to be adapted in order to maintain its expected functionality and quality (i.e., to meet its requirements). This is an important decision, as it might well be that despite the failure of a service, the end-to-end quality of the SBA is not affected, and hence there is no need to react to that situation. Such decisions may be made automatically, or they may require human intervention (that of the end user, system integrator, or application manager). Moreover, such decisions may be made in a reactive way (when the problem has

already occurred) or in a proactive way (where a potential future problem could be avoided). It should be noted that the decision could also be that there should be an evolution of the system rather than an adaptation, thereby entering the development and evolution cycle.

Identify Adaptation Strategy. After the adaptation needs are understood, the corresponding adaptation strategies are identified and selected. Possible types of adaptation strategies include service substitution, SLA renegotiation, SBA reconfiguration, or service recomposition. It could also be that several adaptation strategies are able to satisfy a specific adaptation need. The selection of the strategy and its instantiation (e.g., which service to use as a substitute or which reconfiguration to perform) may be automatic if either the SBA or the execution platform decides the action to perform, or the selection can be done by a human operator. Specifically, the questions of what to adapt and how to adapt must be answered.

Enact Adaptation. After the choice of the adaptation strategy, the adaptation mechanisms are used to enact the adaptation. For example, service substitution, reconfiguration, or recomposition may be done using automated service discovery and dynamic binding mechanisms, while recomposition may be achieved using existing automated service composition techniques. Depending on the situation, such an adaptation can be done manually (e.g., by a human operator), semi-automatically, or fully automatically.

5.3 Application Scenario

In this section, an example workflow is introduced in order to illustrate the problems as well as the solution that will be presented in Section 5.4. The workflow specifies an e-Government SBA that allows citizens to pay parking tickets online, thereby saving effort and cost [21].

Workflow

The workflow as well as the service composition of the e-Government application are depicted as an extended activity diagram in Figure 5-2. The gray boxes denote concrete services that can be composed into an e-Government application. In the example, each service is provided by a third party, whether it is an external organization or a different unit of the governmental organization. Solid connections between workflow actions and services denote the bindings established at deployment time. Dashed connections denote possible alternative services (from a different provider). In addition, the diagram is annotated with information about the negotiated response times (which could be stipulated by means of SLAs).

Let us assume that the overall workflow is expected to have a response time of at most 1,250 ms. This quality requirement can be satisfied by the bound services, provided that they meet their negotiated maximum response times (as, altogether, the maximum response times along the longest path add up to 1,200 ms).

In the following subsections, we use this example to illustrate the shortcomings of reactive adaptation, which have been introduced in Section 5.1.1. We assume that the *ePay* service of the example workflow fails during runtime; that is, it takes longer than the negotiated maximum response time.

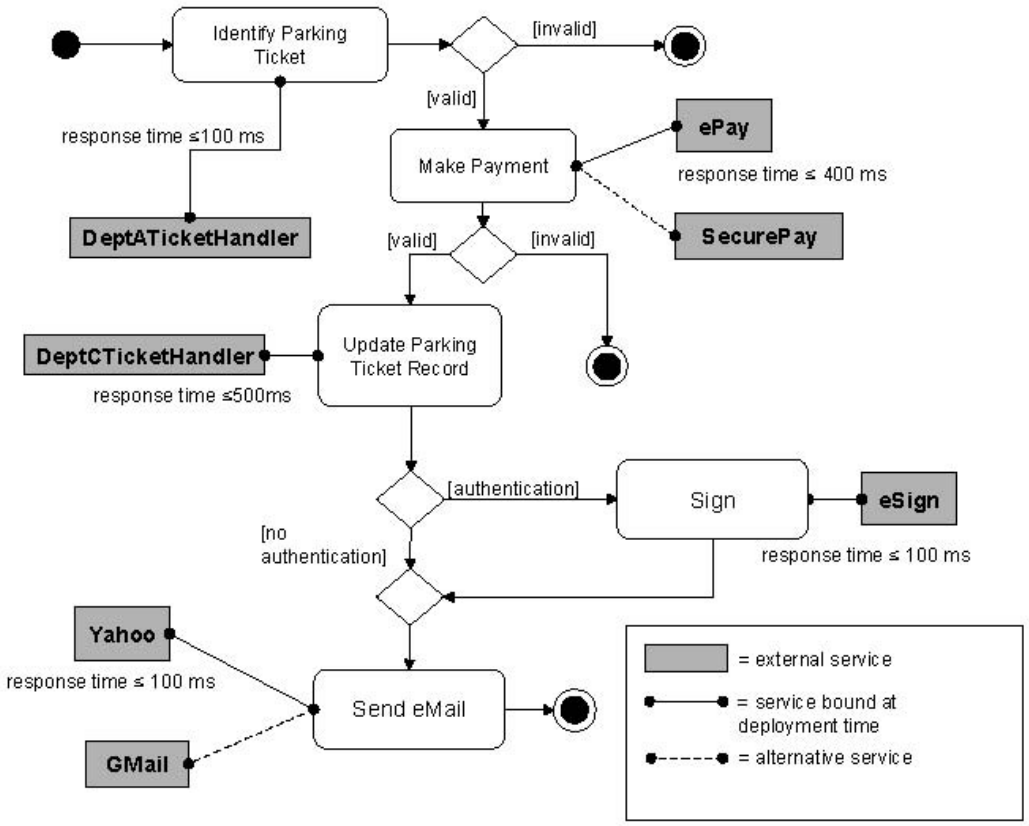


Figure 5-2: Workflow of an e-Government Application

Scenario A: Requirements Monitoring

As mentioned in Section 5.1.1, there are approaches that are restricted to monitoring of requirements. In this case, monitoring events might arrive so late that an adaptation of the SBA is no longer possible. In our example, the *ePay* service invoked by *Make Payment* might take 650 ms to respond instead of the negotiated maximum response time of 400 ms, as shown in Figure 5-3.

Due to the fact that only the requirement (maximum response time of 1,250 ms) is monitored, this failure is not registered until after *Sign* has been invoked. As a consequence, the mechanism was not able to prevent the deviation from the requirements, even though the failure occurred much earlier, as shown by the Δ in Figure 5-3.

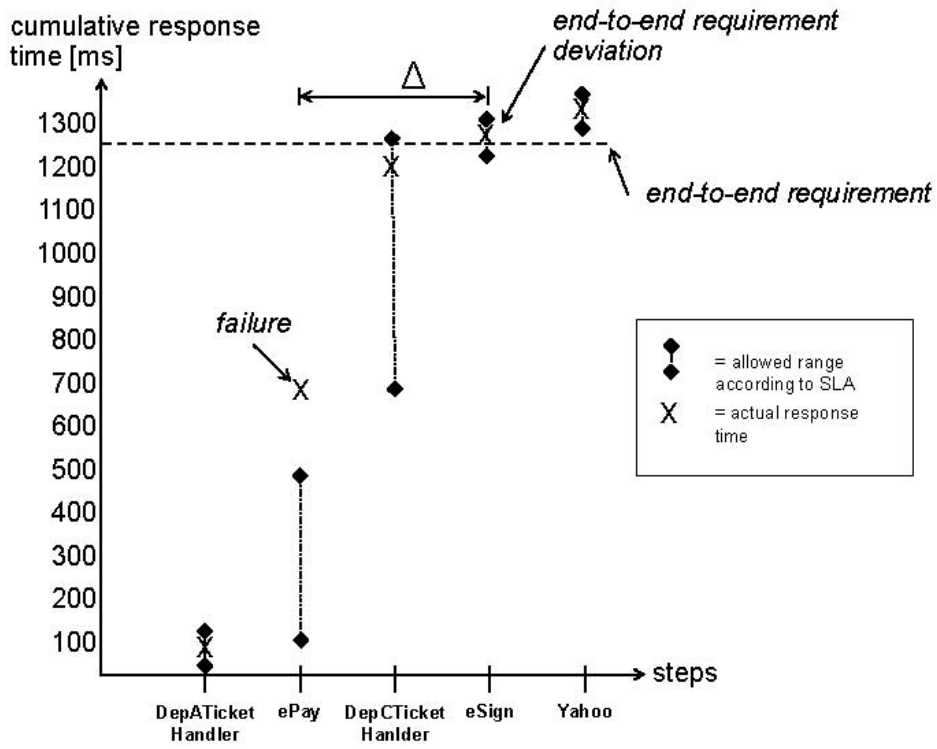


Figure 5-3: Scenario A: Requirements Monitoring

Scenario B: Service Monitoring

As presented in Section 5.1.1, approaches that monitor individual services exist. However, in such settings, it remains unclear whether the failure of a single service leads to a violation of the SBA's requirements. In the example, let us assume that instead of 400 ms, the *ePay* service invocation takes 450 ms, as shown in Figure 5-4.

This failure is observed by means of monitoring and leads to an adaptation of the SBA. However, as is obvious in Figure 5-4, the overall response time would still have matched the required response time, even if no adaptation had been performed. Thus, in this case, an adaptation was triggered although it was not necessary.

In the next section, we will present techniques that enable a more proactive approach to addressing the above shortcomings.

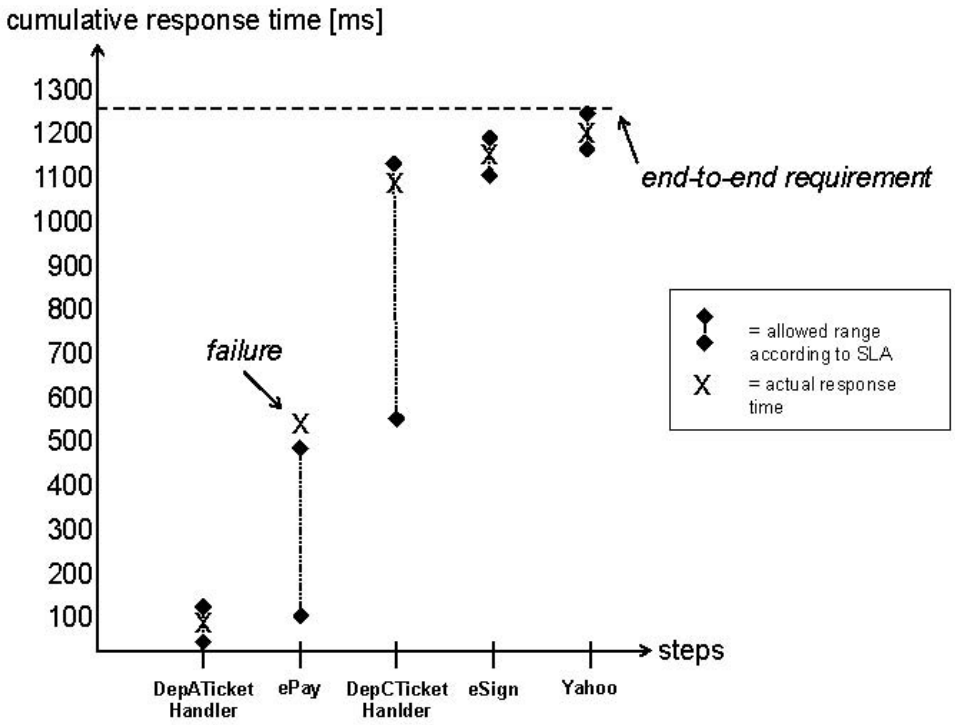


Figure 5-4: Scenario B: Service Monitoring

5.4 Toward Proactive Adaptation across the Life Cycle

This section describes techniques developed in S-Cube for enabling proactive adaptation. The description is organized along the phases of the life-cycle model from Section 5.2. In order to illustrate the techniques, we refer to the example SBA and scenarios presented in Section 5.3.

As explained in Section 5.1.1, adaptive SBAs automatically and dynamically adapt to changing conditions and changes in service functionality and quality. To enable such an automatic adaptation, the relevant artifacts as well as the properties of the SBAs and their context need to be formalized to make them amenable to automated checks and decisions. In the remainder of this section, we introduce concrete formalization approaches as well as techniques that build on this formalization.

Requirements Engineering

To automatically assess whether the application deviates from its requirements during operation and thus trigger an adaptation, functional and nonfunctional requirements need to be collected and formally expressed. We propose to formalize the requirements in the requirements engineering phase, as this also facilitates an early validation of the requirements (e.g., by means of formal consistency checks), and hence reduces the risk of expensive corrections in later phases [22].

S-Cube has developed various approaches to formalize requirements that depend on the actual SBA type. For instance, ALBERT (assertion language for BPEL process interactions) is a specification language based on temporal logics [12]. ALBERT is used to encode functional and

quality attributes. In addition, the S-Cube quality meta-model (QMM) has been defined, which provides a set of key concepts for expressing quality requirements and constraints [23]. To express the requirements for monitoring, an integrated monitoring framework and the corresponding specification language have been provided within the scope of S-Cube [24, 25]. The framework integrates the capabilities of two monitoring platforms: Dynamo and ASTRO [9, 11]. On one hand, the language enables the specification of complex point-wise properties over service composition execution (e.g., pre- and post-conditions on service calls), taking into account current and historical values of the process variables, complex constraints, and event-external properties. On the other hand, simple events and point-wise properties may be aggregated into complex behavioral expressions, also taking into account temporal and statistical information necessary for capturing nonfunctional requirements. While the latter capability is very close to the approach used by ALBERT, the notation allows for expressing properties over classes of processes rather than over single instances. This capability may be very important in order to trigger the "evolution" of the workflow when the problem applies to the whole SBA model rather than to a single SBA instance.

Example: In our example from Section 5.3, we need to formalize the required response time r_{perf} of the e-Government application. r_{perf} is an element of the given set of requirements R_{eGov} of the e-Government application:

$$r_{perf} \in R_{eGov}$$

r_{perf} demands the response time of the e-Government application to be 1,250 ms at most. Due to the capability of ALBERT to express the dependencies of monitoring data along an executed path, we use this language to specify the requirement (r_{perf}) as follows:

$$r_{perf} := onEvent(start, \text{"Identify Parking Ticket"}) \rightarrow Within\ (onEvent(end, \text{"Send eMail"}), 1250)$$

The *onEvent* operator evaluates to true if the activity specified in the second argument performs the state change denoted in the first argument. The *Within* operator evaluates to true if its first argument evaluates to true within the number of milliseconds specified in its second argument.

Design

Similar to requirements, the workflow of the SBA needs to be formalized to support automated checks. Following the same reasoning as in the requirements engineering phase, we suggest formalizing the workflow during the design phase in order to reduce the risk of later corrections. The checks can be performed using model checking techniques [8]. In S-Cube, the use of the BOGOR model checker has been proposed [12] to assess whether the specified SBA satisfies the requirements [12]. We formalize the workflow using the input language of the model checker, in this case, the BOGOR Input Representation (BIR).

Example: In order to use the BOGOR model checker, we specify the e-Government workflow by using BIR [12]. The resulting specification S_{eGov} can be directly executed and analyzed by BOGOR. This specification is included as an appendix.

Realization

During the realization phase, the quality levels (also called service-level objectives) that have been negotiated and agreed upon with the service providers are formalized, as explained in Section 5.2. We treat those quality levels as assumptions (*A*) about the SBA's context [8]. Due to the lack of control over third-party services, those quality levels could be violated during the operation of the SBA, as explained in Section 5.1.

To formalize *A*, we can use one of the quality formalization approaches, just like in the requirements engineering phase. For checking the violation of the assumptions during the operation of the SBA, monitoring mechanisms are implemented that collect the relevant data [12, 24, 25]. This is equivalent to collecting the monitoring data in the reactive case of adaptation presented in Section 5.3.

Example: According to their SLAs, as shown in Figure 5-2, ALBERT is used to formalize the five assumed response times. The set of assumptions A_{eGov} for the parking ticket SBA is defined as

$$A_{eGov} := \{a_{DeptATicketHandler}, a_{ePay}, a_{DeptCTicketHandler}, a_{eSign}, a_{Yahoo}\}$$

The assumption a_{ePay}, related to the *ePay* service invocation, is formalized as follows:

$$a_{ePay} := onEvent(start, \text{“Make Payment”}) \rightarrow Within(onEvent(end, \text{”Make Payment”}), 400)$$

Deployment

Before deploying the SBA, it is checked whether the workflow specification (*S*), under the given assumptions (*A*), satisfies the requirements (*R*):

$$S, A \vDash R$$

This check ensures that the initial composition—both the workflow and the services—satisfies the requirements. If this is not the case, the phases of the evolution loop described in the life cycle in Section 5.2.1 are executed again in order to redesign the application, for example, to bind faster services. If the SBA is successfully verified against the requirements, then the SBA is deployed.

Example: In our example, S_{eGov} and A_{eGov} satisfy R_{eGov}. As a consequence, the SBA is deployed.

Operation and Management

This phase comprises the execution and the monitoring of the individual services of the deployed SBA.

Monitoring is supported by monitoring frameworks such as Dynamo [25]. At runtime, the monitoring framework continuously assesses whether the monitoring data *M* satisfies the formalized assumptions *A* about the services:

$$M \vDash A$$

If a violation occurs, the SBA enters the adaptation loop, as explained in Section 5.2.2. The relevant activities are described below.

Example: After finishing the SBA deployment, the e-Government application is executed. Let us assume that the first activity, which invokes the *DeptATicketHandler* service, lasts 90 ms. The measured response time of the *DeptATicketHandler* call is stored as monitored data $m_{DeptATicketHandler}$. $m_{DeptATicketHandler}$ satisfies the assumption that the service responds within 100 ms ($a_{DeptATicketHandler}$). In the following step, *ePay* is invoked. Let us assume that the invocation of *ePay* is slower than expected. This is the same situation as described in Scenarios A and B in Section 5.3. Instead of the expected 400 ms, the *ePay* invocation takes 450 ms (Scenario B). Hence, the monitoring data of the second service invocation m_{ePay} does not confirm the corresponding assumption a_{ePay}:

$$m_{ePay} \not\models a_{ePay}$$

Due to this violation, the phases of the adaptation loop are entered.

Identify Adaptation Needs

In this phase, it is checked whether the requirements are still satisfied although the assumptions have been violated [8]. For example, it might be the case that a slower response time of one service is compensated by a faster response time of a previous service; consequently, no adaptation is required. When the check is performed, there are usually services that have not been invoked yet. All services have been invoked only after the workflow is finished. This means that there is no monitoring data available for the services that have not yet been invoked. For these services, we continue to use their assumptions in the checks; that is, we use a subset $A' \in A$. Next, it is checked whether the workflow specification *S*, the monitored data *M*, and the assumptions in A' satisfy the given requirements *R*.

$$S, M, A' \models R$$

If *R* is satisfied, then the workflow execution is continued. If *R* is not satisfied, then the SBA must be adapted.

Example: To illustrate that the presented S-Cube approach is adequate to address the shortcomings from Scenarios A and B described in Section 5.3, we compare the S-Cube approach to the requirements monitoring approach presented in Scenario A and the sequence monitoring approach presented in Scenario B. It is checked whether there is a deviation from the requirements, as this could indicate that an adaptation is necessary. This check also covers cases with larger delays, for example, 500 ms in Scenario A.

The approach presented in Scenario A does not observe failures at the moment they occur, as depicted by Δ in Figure 5-3. The S-Cube approach does not have this shortcoming. The continuous monitoring of the service behavior observes failures as soon as a problem occurs.

Based upon such observation, the SBA requirements are immediately checked. This provides the system with the opportunity to adapt itself to prevent the predicted requirements deviation from occurring. Of course, the ability of the system to proactively adapt depends on the time available for such actions. Typically, if a failure of a service is observed earlier in the workflow, then there is more time to adapt the remainder of the workflow accordingly.

In order to determine such requirements violations, model checking techniques are used. The workflow specification (S_1) and the monitoring data ($m_{DeptATicketHandler}$ and m_{ePay}), together with the assumptions of the outstanding service invocations ($a_{DeptCTicketHandler}$, a_{eSign}, and a_{Yahoo}), are checked against the requirement r_{perf}. The expected overall runtime is 1,450 ms, which exceeds the 1,250 ms demanded in r_{perf}. Hence, the requirement r_{perf} is not satisfied. This result is considered an identified adaptation need.

Subsequent to this check, the adaptation can be performed proactively before the requirement is actually violated (i.e., before the system in operation deviates from its expected requirements).

The approach presented in Scenario B is not able to determine whether a failure of a single service leads to a violation of the SBA requirements. Each time a service fails, the SBA adapts immediately. The S-Cube approach presented in this paper allows adapting only in cases when critical failures occur, thereby avoiding unnecessary adaptations. The same check as described above assesses that the expected overall runtime does not exceed 1,250 ms. The requirement r_{perf} is still satisfied, and thus no adaptation trigger is needed. An unnecessary adaptation is prevented, which would have been performed in Scenario B.

Decide on Adaptation/Identify Adaptation Strategy

When the need for adapting an SBA is detected, the next step is to identify and apply an appropriate adaptation strategy from the ones that are available for the considered applications. Depending on the application, the adaptation strategies range from service re-execution, to replacement of a single service or of a process fragment, to renegotiation of quality properties, to changes in underlying infrastructure, and so forth. Note that the adaptation strategies should be designed with the application because some of them require the adoption of specific infrastructure or the implementation of additional components.

Typically, the adaptation strategy is associated with a specific critical situation or a problem at design time. This association may be done either implicitly or explicitly. In the former case, the mechanisms for choosing one action or another are hard-coded in decision mechanisms. A typical scenario is the replacement of a service that violates the SLA or an SBA requirement with a service with more appropriate and suitable characteristics. Based on the selection criteria (e.g., optimization of a quality function, adherence to application constraints), the appropriate decision mechanism may choose one service over another. In the scope of the S-Cube project, several approaches follow this vision. For example, replacement policies may be used to realize such a decision mechanism and define the association between various types of changes (service failure, changes in service properties and models, appearance of new services, and changes in the context and requirements) and the service selection [26]. The decision on the adaptation strategy can be based on the quality factors of the SBA that should be improved [27]. Those factors are identified through the analysis of the dependency tree that captures the relation between simple quality factors and SBA requirements. At design time, the adaptation action is assigned to the quality factors that it influences either positively or negatively. The selection of the adaptation strategy is based on the need to improve quality factors that are critical for the requirement while trying to minimize the negative effect on the other factors. In our scenario, the requirement would need to improve the performance of the last service, and a service replacement would be proposed such that the new service has better performance, while being less costly compared to alternatives.

The definition of the adaptation strategy may be also explicitly assigned to the critical situation. Baresi et al. present an adaptation strategy represented in WS-ReL, a notation for specifying and integrating recovery actions in service composition [28]. Therefore, the adaptation is defined as a rule, in which on the left-hand side a critical situation is defined (as a formal requirement to be monitored) and on the right-hand side a set of actions is applied. Possible actions include re-executing a service invocation, replacing a service or a provider (partner link), ignoring the failure or halting the execution, executing an extra process fragment, or rolling back to a safe point. Simple actions may be joined into a complex strategy by defining a control flow over actions, like "try action A else try action B and action C." These rules are evaluated and applied by the underlying adaptation engine.

Adaptation can also be based on the causes of failures. This is particularly helpful when invoked services are stateful and when their invocation modifies the state of the service, such as in transactional services. For processes involving transactional services, if a diagnosis mechanism is available, the adaptation strategy can depend on the cause of the failure and its implications on the processes [29]. This might imply an adaptation strategy that involves one or more services in the process that must be dynamically generated.

Enact Adaptation

To enact adaptation actions, the SBA or its execution platform should be appropriately instrumented. A typical approach for realizing adaptation mechanisms for SBAs implemented as executable (BPEL) processes is to instrument the process execution engine. Such instrumentation is done through aspect-oriented programming techniques, as the adaptation activities are treated as a cross-cutting concern. Using this approach, the join points allow for injecting the adaptation logic in order to intercept and adjust process execution logic. In particular, Baresi et al. propose a supervision manager component that is attached to the ActiveBPEL process engine and performs the necessary supervision activities: monitoring of critical situations, evaluation of adaptation rules, and calls to the process engine infrastructure to realize the specific strategy [28]. Similarly, Karatoyanova and Leymann propose a mechanism in which aspect-oriented techniques are adopted in order to dynamically bind services into service compositions that are realized as BPEL orchestrations [30].

5.5 Conclusions and Perspectives

This paper has introduced novel techniques developed in S-Cube (the European Network of Excellence on Software Services and Systems) for equipping SBAs with proactive adaptation facilities. These techniques are able to avoid costly compensation and repair activities, as well as unnecessary adaptations, which are deemed to be key shortcomings of current solutions for adaptive service-oriented applications.

The techniques have been introduced along the key phases of the S-Cube service life cycle. This paper has demonstrated when and how these techniques should be applied when developing, evolving, and adapting SBAs.

We are confident that these techniques will become especially relevant in the setting of the Internet of Services, in which applications will increasingly be composed from third-party services that are not under the control of the service consumer. This implies that applications and

their constituent services need to be continuously checked during their operation so that they can be adapted dynamically or evolved in order to respond to failures or unexpected changes of third-party services.

With S-Cube, we are currently striving to push the envelope toward proactive adaptation even further. In addition to determining the need for adapting the SBA based on actual failures of the application's constituent services, we investigate the applicability of online testing to predict the quality of those services [6, 7]. Combined with the approaches introduced in this paper, this means that critical problems could be observed even earlier, thus enabling a broader range of adaptation and evolution strategies. For instance, in our running example, we can react to the violation of the response time of a constituent service only by ensuring that the remainder of the workflow executes faster. However, if the quality prediction techniques forecast a violation of the expected response time of a specific service, this very service can be replaced before it is invoked in the context of the SBA.

Another challenging problem that will be addressed is how to make the techniques robust against other kinds of false positives. Currently, our techniques define the assumptions about a service execution to be the upper limits of the quality properties as stated in the SLAs. As a consequence, it might happen that the proactive techniques predict a performance requirements violation based on a failure of a service despite the fact that the remaining service invocations of the workflow might have been executed much faster than stated in the SLAs and thus compensated for this failure. As a result, we will investigate how past monitoring data can be used to better define the assumptions that can be made about the quality properties of a service.

Acknowledgments

We thank the anonymous reviewers for their helpful and constructive comments. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). For further information, please visit http://www.s-cube-network.eu/.

References

[1] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, "A journey to highly dynamic, self-adaptive service-based applications," Automated Software Engineering, vol. 15, no. 3-4, 2008, pp. 313–341.

[2] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and opportunities," IT Professional, vol. 8, no. 2, 2006, pp. 10–17.

[3] A. Metzger and K. Pohl, "Towards the next generation of service-based systems: The S-Cube research framework," in CAiSE 2009, ser. LNCS, J. P. van Eck, J. Gordijin, and R. Wieringa, Eds. Berlin: Springer, 2009, pp. 11–16.

[4] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," ACM Transactions on Autonomous and Adaptive Systems, vol. 4, no. 2, 2009.

[5] S. Benbernou, "State of the art report, gap analysis of knowledge on principles, techniques and methodologies for monitoring and adaptation of SBAs," S-Cube Consortium,

Deliverable PO-JRA-1.2.1, July 2008 [Online]. Available: http://www.s-cube-network.eu/results/

[6] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka, "Towards proactive adaptation with confidence augmenting service monitoring with online testing," in Proceedings of the ICSE 2010 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '10), Cape Town, South Africa, 2–8 May 2010.

[7] J. Hielscher, R. Kazhamiakin, A. Metzger, and M. Pistore, "A framework for proactive self-adaptation of service-based applications based on online testing," in ServiceWave 2008, ser. LNCS, no. 5377. Springer, 10–13 December 2008.

[8] A. Gehlert, A. Bucchiarone, R. Kazhamiakin, A. Metzger, M. Pistore, and K. Pohl, "Exploiting assumption-based verification for the adaptation of service-based applications," in SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing. New York: ACM, 2010, pp. 2430–2437.

[9] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-time monitoring of instances and classes of web service compositions," in IEEE International Conference on Web Services (ICWS 2006), 2006, pp. 63–71.

[10] D. Dranidis, E. Ramollari, and D. Kourtesis, "Run-time verification of behavioural conformance for conversational web services," in Seventh IEEE European Conference on Web Services (ECOWS 2009), 9–11 November 2009, Eindhoven, the Netherlands, R. Eshuis, P. W. P. J. Grefen, and G. A. Papadopoulos, Eds., 2009, pp. 139–147.

[11] C. Ghezzi and S. Guinea, "Run-time monitoring in service-oriented architectures," in Test and Analysis of Web Services, L. Baresi and E. D. Nitto, Eds. Berlin: Springer, 2007, pp. 237–264.

[12] D. Bianculli, C. Ghezzi, P. Spoletini, L. Baresi, and S. Guinea, A Guided Tour through SAVVY-WS: A Methodology for Specifying and Validating Web Service Compositions. Advances in Software Engineering, ser. Lecture Notes in Computer Science, vol. 5316. Springer-Verlag, 2008, pp. 131–160.

[13] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime prediction of service level agreement violations for composite services," in 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing, co-located with ICSOC 2009, 2009.

[14] A. Bucchiarone, C. Cappiello, E. D. Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore, "Design for adaptation of service-based applications: Main issues and requirements," in Fifth International Workshop on Engineering Service-Oriented Applications: Supporting Software Service Development Lifecycles (WESOA), 2009.

[15] A. Gehlert, M. Pistore, P. Plebani, and L. Versienti, "First version of integration framework," S-Cube Consortium, Deliverable CD-IA-3.1.3, December 2009 [Online]. Available: http://www.s-cube-network.eu/results/

[16] A. Bucchiarone, R. Kazhamiakin, C. Cappiello, E. Di Nitto, and V. Mazza, "A context-driven adaptation process for service-based applications," in PESOS 2010: 2nd International Workshop on Principles of Engineering Service-Oriented Systems. Cape Town, South Africa, 1–2 May 2010, pp. 50–56.

[17] B. Pernici, Methodologies for Design of Service-Based Systems. Berlin: Springer, 2010, ch. 17.

[18] E. Nitto, "State of the art report on software engineering design knowledge and survey of HCI and contextual knowledge," Deliverable PO-JRA-1.1.1, 2008 [Online]. Available: http://www.s-cube-network.eu/results/

[19] D. Bianculli, C. Ghezzi, and C. Pautasso, "Embedding continuous lifelong verification in service life cycles," in Proceedings of Principles of Engineering Service Oriented Systems (PESOS 2009), co-located with ICSE 2009, Vancouver, Canada. Washington, DC: IEEE Computer Society Press, May 2009.

[20] M. Comuzzi and B. Pernici, "A framework for QoS-based web service contracting," ACM Transactions on the Web, vol. 3, no. 3, 2009.

[21] E. D. Nitto, V. Mazza, and A. Mocci, "Collection of industrial best practices, scenarios and business cases," S-Cube Consortium, Deliverable CD-IA-2.2.2, 2009 [Online]. Available: http://www.s-cube-network.eu/results/

[22] F. L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, and H. Wössner, The Munich Project CIP: Volume I: The wide spectrum language CIP-L. London: Springer, 1985.

[23] A. Gehlert and A. Metzger, "Quality reference model for SBA," Deliverable CD-JRA-1.3.2, 2008 [Online]. Available: http://www.s-cube-network.eu/results/

[24] L. Baresi, S. Guinea, R. Kazhamiakin, and M. Pistore, "An integrated approach for the run-time monitoring of BPEL orchestrations," in Service-Wave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet. Berlin: Springer, 2008, pp. 1–12.

[25] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, "Dynamo + astro: An integrated approach for BPEL monitoring," in IEEE International Conference on Web Services, 2009, pp. 230–237.

[26] K. Mahbub and A. Zisman, "Replacement policies for service-based systems," in 2nd Workshop on Monitoring, Adaptation and Beyond (MONA+), co-located with ICSOC 2009, 2009.

[27] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann, "Adaptation of service-based applications based on process quality factor analysis," in Proc. of 2nd Intl. Workshop on Monitoring, Adaptation and Beyond (MONA+), co-located with ICSOC/Service-Wave '09, 2009.

[28] L. Baresi, S. Guinea, and L. Pasquale, “Integrated and composable supervision of BPEL processes,” in International Conference on Service-Oriented Computing (ICSOC), 2008, pp. 614–619.

[29] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, “Exception handling for repair in service-based processes,” IEEE Trans. Software Eng., vol. 36, no. 2, 2010, pp. 198–215.

[30] D. Karastoyanova and F. Leymann, “BPEL’n’aspects: Adapting service orchestration logic,” in International Conference on Web Services (ICWS), 2009, pp. 222–229.

Appendix: Workflow Specification S_{eGov}

```
system ParkingTicketSpec {
   Action identifyParkingTicket;
   Action makePayment;
   Action updateParkingTicketRecord;
   Action sign;
   Action sendEMail;

   int sumMaxResponseTime := 0;

   record Action {
      string serviceName;
      int maxResponseTime;
      boolean serviceInvoked;
   }

   active thread MAIN () {
      init();
      checkWorkflow();
      checkRequirements();
   }

   function init() {
      identifyParkingTicket := createAction("DeptATicketHandler", 100);
      makePayment := createAction("ePay", 400);
      updateParkingTicketRecord := createAction("DeptCTicketHandler", 500);
      sign := createAction("eSign", 100);
      sendEMail := createAction("Yahoo", 100);
   }

   function checkWorkflow() {
      executeAction(identifyParkingTicket);
      choose
         do skip;
         do
            atomic
               executeAction(makePayment);
               choose
                  do skip;
                  do
                     atomic
                        executeAction(updateParkingTicketRecord);
                        choose
                           do executeAction(sign);
                           do skip;
                        end
                        executeAction(sendEMail);
                     end
               end
            end
      end
   }
```

```
    function checkRequirements() {
        assert sumMaxResponseTime <= 1250;
    }

    function executeAction(Action action) {
        sumMaxResponseTime := sumMaxResponseTime + action.maxResponseTime;
        action.serviceInvoked := true;
    }

    function createAction(string serviceName, int maxResponseTime)
    returns Action {
        Action action;
        action := new Action;
        action.serviceName := serviceName;
        action.maxResponseTime := maxResponseTime;
        action.serviceInvoked := false ;
        return action;
    }
}
```

6 A Dynamic Framework for Quality Web Service Discovery

Atousa Pahlevan, University of Victoria, Canada (pahlevan@cs.uvic.ca)
Hausi A. Müller, University of Victoria, Canada (hausi@cs.uvic.ca)
Mantis Cheng, University of Victoria, Canada (mcheng@cs.uvic.ca)

Abstract

Support for dynamic attributes is increasingly important for facilitating high-quality web service discovery for service-oriented architectures. Previously, we proposed the static discovery dynamic selection (SDDS) architecture to overcome limitations of existing web service discovery methods. Our service discovery algorithms consider dynamic attributes to reduce the number of viable and acceptable services and thereby improve the consumer experience. In this paper, we propose a semantic model based on finite state automata to validate interactions among the SDDS components that deal with dynamic attributes. Our analysis using the automata model revealed a flaw in the SDDS communication synchronization. This flaw was corrected by introducing a web-based synchronization component to enforce valid communication patterns.

6.1 Introduction

A service-oriented architecture (SOA) is an architecture based on a collection of services that communicate with each other. SOAs are built around loosely integrated service endpoints, enabling interoperability among heterogeneous systems. As the number of static web services increased, the concept of a discovery mechanism was introduced so that applications could more effectively identify the services they need. One of the key phases in web service discovery is service selection, which is the process of finding a service that is pertinent to a user's request based on dynamic context or attributes [1]. Service providers register their services in a Universal Description, Discovery, and Integration (UDDI) registry that can then be accessed by the public. A broker helps requesters and service providers find each other. The requesters ask the broker about the services they require. When the broker returns the results, the requesters use the findings to bind themselves to a particular service.

There are several drawbacks to existing web service discovery mechanisms. First, these methods assume that the world is static and therefore do not support dynamic attributes. Kim and Rosu show that approximately 16 percent of registered services are not accessible, due to unavailability or long response time [2]. In addition, based on our observations of dealing with public registries, about 34 percent of registered services are invalid due to syntax errors in web service descriptions written in Web Services Description Language (WSDL) [3]. One of the challenging issues in SOA is ensuring quality of service (QoS) by enabling service selection in dynamic environments [4, 5]. Considerable research has been done to support dynamic attributes in SOA. Expanding UDDI to register dynamic attributes is one of the common solutions. A second solution is using an expanded broker to store and manipulate dynamic attributes. A hybrid method combines both approaches.

These enhanced methods, however, do not involve factors that could affect dynamic attributes and consequently service selection. To resolve ambiguities in service selection in SOA, we need to discover and select services by considering the situation of the user and domain information of the service. This information is referred to as service and user context and is domain-specific information [1, 6].

To improve the quality of dynamic web service discovery, we propose an enhanced model to support dynamic attributes based on SOA principles. We outline an architecture framework called static discovery dynamic selection (SDDS) to evaluate the dynamic attributes concerning both context and domain information during discovery [7]. The architecture of SDDS defines individual components that collectively satisfy a flexible and intelligent service selection for realizing SOA. Our main research contributions are as follows:

- A mechanism to collect service information in an active manner.
- A sensing technique to acquire the required information from the UDDI service discovery mechanism on demand.
- A robust resource management approach to collaborate with partners concerning static and dynamic attributes.
- An advanced SOA model to register, collect, store, and measure the dynamic attributes.
- An enriched context-sensitive method to consider the service and user context for effective service selection.
- An intelligent domain-sensitive method to realize the important attributes of each domain.

One of the main challenges to enabling dynamic service discovery is to develop techniques and models to handle the novel aspects of the web services paradigm (i.e., updated static and dynamic attributes). This challenge leads to a variety of research questions:

- What is the best way to model web services using static and dynamic attributes?
- How does one query them to support dynamic attributes?
- How can data management be incorporated into current web service standards?
- How can resource management be used to collaborate between partners to guarantee data liveness and validity?

To address these questions, more attention needs to be paid to the methods used to identify qualified services. In particular, it is worthwhile to understand the key elements in the design process and to identify the activities that support the required properties. For example, orchestration between new components and existing service discovery components is critical to satisfying properties, such as data validity and liveness in shared databases.

In this paper, we focus on analyzing our web-based architecture framework using finite state automata [7, 8, 9]. To discover whether our proposed architecture satisfies data validity and liveness in registries, we modeled individual SDDS components as finite state automata. We then composed a critical set of these components and analyzed the interactions among them to validate the correctness of the architecture. We found that the current design of SDDS could not guarantee the validity of registered services due to several invalid transitions in the associated automaton.

This paper discusses this validation process and proposes a solution to address this issue by using a web-based synchronization component that allows us to eliminate the flawed transitions.

The rest of this paper is organized as follows. The next section discusses related work and outlines existing techniques. Sections 6.3 and 6.4 introduce our proposed framework. The detailed analysis and validation of SDDS using finite state automata are discussed in Section 6.5. Finally, Section 6.6 draws some conclusions.

6.2 Related Work

Several frameworks have been developed in the context-aware area to handle dynamic attributes [10, 11, 12]. However, most of them are not web-based and are implemented using remote method invocation (RMI) or common object request broker architecture (CORBA) technologies. A typical example is the PACE middleware, which features management components to assist context-aware applications [10]. Implementation of the PACE middleware does not address scalability or performance, making it unsuitable to support web service discovery. SOCAM is an OWL-based context-aware middleware mechanism implemented on top of RMI [11]. Another project is CAMUS, which targets intelligent robots [12]. Although its infrastructure covers many kinds of contexts, it remains limited to closed environments that have very different requirements than service-oriented systems.

For web service discovery, we need scalable and flexible infrastructure. Some web-based frameworks concentrate on context; however, they are mostly targeted to mobile users. The aim of the Akogrimo project is to enable mobile users to access services on the grid [13]. Anyserver is another platform that covers a broader definition of context, not limited to time and location [14]. The aforementioned works are either partially implemented in web services or limited to single-organization environments. Given these limitations, it is unclear to what extent they are useful in the context of web services. In contrast to these methods, our framework relies on technologies designed for large-scale, loosely coupled, web service-based environments, and it benefits from run-time context acquisition [15].

6.3 Static Discovery Dynamic Selection Conceptual Architecture

The overall system architecture of SDDS includes two major processes, as depicted in Figure 6-1. Service discovery is a standard service search-and-discovery mechanism, compliant with UDDI. This process suffers from several limitations in finding qualified services. To improve the service discovery process, dynamic selection (DS) is introduced to select the convenient services from discovered ones. Here, the required information about services is provided to DS by sensors from UDDI. DS contains two main components: the user dynamic manager and service dynamic manager. Both facilitate static and dynamic data collection, analysis, planning, and storage. These components collaborate to select the appropriate services for a given request.

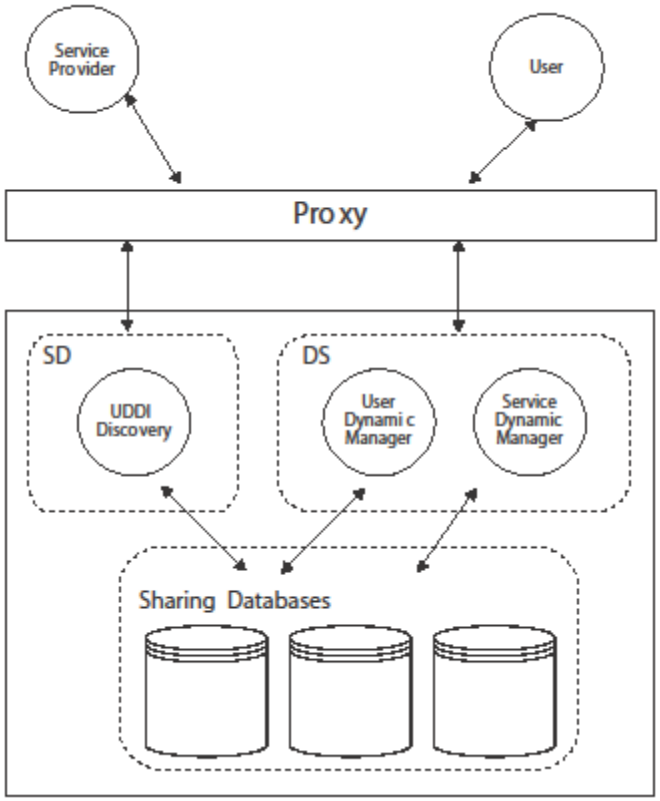


Figure 6-1: SDDS Conceptual Architecture

6.4 SDDS: The Proposed Architecture Model

Our SDDS architecture and experimental setup for enhanced service discovery is based on SOA and is depicted in Figure 6-2 [7, 16]. It supports static and dynamic attributes with the following components.

- Service consumer: A consumer of the services provided by others who makes a request for a specific service.
- Service provider: A provider of services to others who explicitly registers a service with a web service registry.
- Proxy: A web service broker to deal with passing and parsing messages between components. It is used for publishing and querying relevant dynamic attributes. The proxy forwards standard service descriptions to the standard UDDI service discovery and returns the unique identifier of the published service. Additional dynamic attributes are stored with the dynamic repository by the service dynamic manager. The proxy also accepts both standard UDDI queries and dynamic queries.
- Standard UDDI repository: A repository in which to store the standard service descriptions, which are static values.

- Dynamic repository: A database in which to store high-grade services that are checked for dynamic attributes. To identify the high-grade services, several quality aspects are taken into account.
- Ontology repository: The database of semantic knowledge in the ontology of both the external domain and context. The ontology database stores and provides knowledge in machine-readable form for context and domain processing, in addition to information on how the processing is actually performed.
- Web service profile: The history of the selected web services.
- UDDI service discovery: The standard UDDI service discovery and selection mechanism, based on static attributes.
- Service dynamic manager: A manager to address the nonfunctional attributes of selected services in order to identify service properties best. It checks the quality of the services in the standard UDDI repository and stores the high-grade services in the dynamic repository. To determine the high-grade services, aspects such as user context, service context, domain knowledge, and user feedback are taken into account.
- User dynamic manager: Deals with the dynamic values on the user side. It extracts the dynamic attributes from queries and stores and retrieves a user and web service profile on demand.

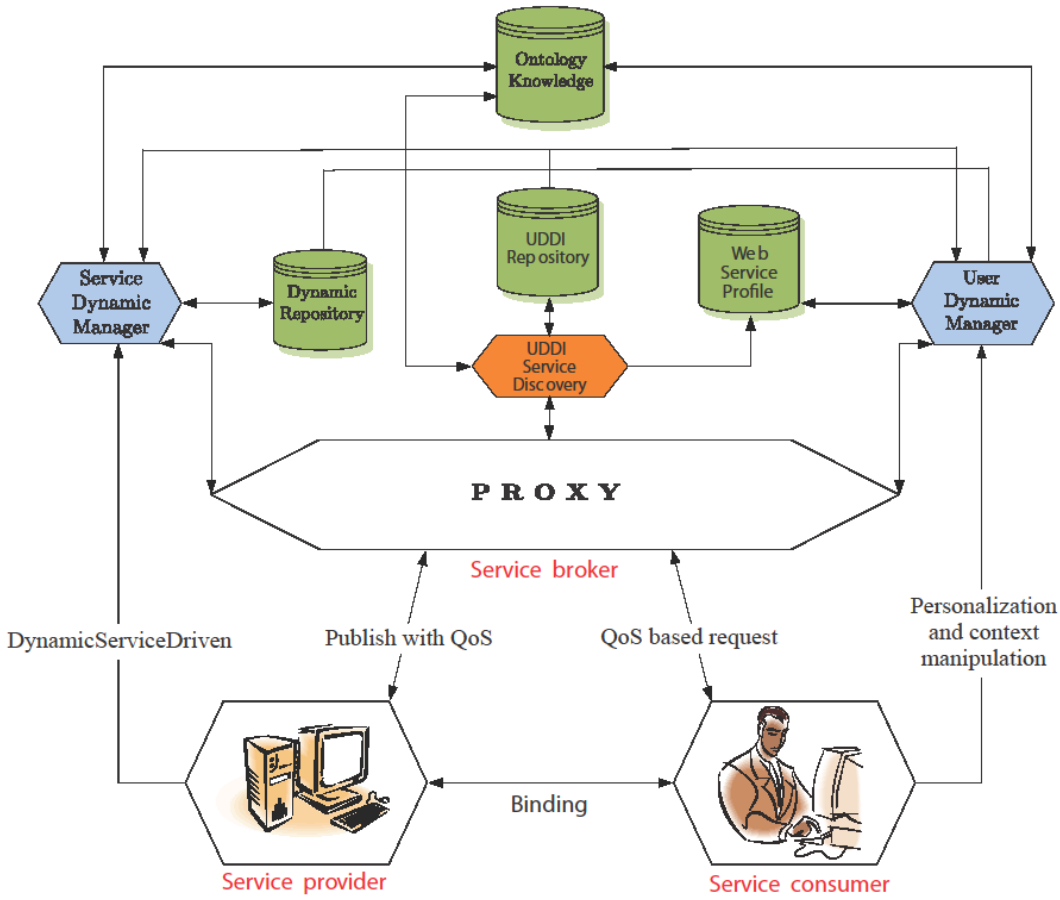


Figure 6-2: SDDS Architecture

Two kinds of dynamic attributes can be specified in SDDS: service and user dynamic attributes. We assume that each service is equipped with a mechanism to capture its dynamic values from relevant sources and provide the gathered information to the SDDS. Our architecture for high-quality web service discovery comprises two phases: static discovery (SD) and dynamic selection (DS). On one hand, the SD process resembles an index method that enables service discovery within a collection of services classified by their static attributes. On the other hand, the DS process is initiated for service retrieval based on dynamic attributes. This process consists of the following steps.

1. SD: SD handles standard service publishing and querying. It discovers the appropriate services by employing a scalable structured index. It employs a classification technique to group similar services together based on service type and to maintain those services in a tree structure for efficient service discovery [17]. The service type is identified based on the static attributes of the web. As a result, service discovery costs are reduced due to a more efficient search with an indexing structure. The candidates resulting from this step are forwarded to the next step for further processing and filtering.
2. DS: DS handles the evaluation of dynamic attributes of web services and filters out services that are not relevant to the user based on dynamic values gathered from different sources. It

is initiated by SD to select the highest quality service by evaluating the dynamic attributes of discovered services.

3. Context manipulation: This enables the use of context to provide relevant services to the user when service behavior can be affected by context values. For instance, an expensive service with higher accessibility would not be a good candidate if cost is an issue for a service consumer at the time of discovery. Context information is always available and up-to-date; however, the service provider and service requesters are unaware of the existence of a mechanism that can use that information.
4. Domain handling: For most domains, there is no quality criteria model for web services. Determining the QoS requires domain-specific knowledge that can be given by experts to represent the priorities and preferences of the specific domain. These quality criteria can be defined by a quality ontology and are stored in ontology knowledge databases to enable the automated web service selection process. However, based on each application, application-specific metrics are specified and assigned a value based on the quality properties of services. Those values are stored and updated as default weight vectors in the dynamic repository. The default weight vectors contain threshold values for quality parameters used in the domains that are time critical or privacy critical. Then, in the selection process, the services that have a value less than the threshold are eliminated. Hence, it creates a trust-based QoS-aware service discovery model by eliminating services that are inconsistent with the default weight vector in the same service type, and it subsequently creates a web service discovery mechanism with implicit QoS filtering.
5. Personalization: SDDS provides personalization by allowing the user dynamic manager to keep track of selections made by users and their web service history. Thus, the next time a user requests a service of the same type without the dynamic values, DS is able to look at the history of the user's previous selections and web service history to make a selection based on that data.
6. Request management: The importance of a dynamic attribute for a specific service may differ for different users. For example, for one user, service reliability might be critical, while to another, a faster service response time might be more important. SDDS provides request management to allow users to specify different preferences for dynamic attributes.

6.5 Design of SDDS

Our SDDS engineering design is mainly component based—that is, it combines existing methods with new software to provide new functionalities. We propose a constrained finite state automata formalism to model the interaction among SDDS components. We capture the I/O behavior of each component with an automaton. Input actions are used to model the methods that can be called, while the output actions are used to model the method calls between components. Using automata, we formalize the behavior of the SDDS architecture by modeling its critical constituents: service dynamic manager, user dynamic manager, and standard UDDI service discovery. We use an automata-based language to capture the order in which the methods of the components are called and the order in which the components call external methods. A composite automaton is constructed from the product of the component automata. When the component automata are composed, the result may contain invalid transitions, such as transitions that jeopardize data integrity in databases. To guarantee atomicity, durability, and consistency in

databases, we attempt to identify and avoid such invalid transitions. This formalism for modeling the behavioral aspects of components can be used both during design and for validating the system [18, 19].

In the following sections, we introduce fundamental concepts and then discuss SDDS component automata and their composite automaton.

6.5.1 Defining Finite State Automata for SDDS

Automata are primarily semantic models that can be used to validate the operational constraints of a system (e.g., being deadlock free). The use of finite state automata was inspired in engineering disciplines to allow for robust mathematical analyses. In an automata model, the architecture is represented as sets of states and transitions. Automata states denote the possible configurations of the system; automata transitions demonstrate the possible actions and their effects on those states. First, we introduce the terminology used in subsequent sections [8, 20].

Definition 1

An SDDS automaton is a tuple $SDDS = (Q, \Sigma, \Delta, q_0, M, V)$, where

- Q represents a finite set of states;
- Σ denotes a set of actions;
- $N \subseteq \Sigma$;
- g is a set of constraints for transition;
- $\Delta : Q \times N \times g \rightarrow Q$ and is referred to as the transition relation for SDDS;
- q_0 denotes a set of initial states where the automaton starts $q_0 \subseteq Q$;
- M presents a set of global memory cell for states;
- V for each $q \in Q$ the value function $V_q : M \rightarrow Value$ is defined when the automaton is in state q. The set V_{q_0} includes the initial value functions of V_q, each of which gives the initial values for the memory cells of its corresponding state $q \in Q$.

For simplicity, we explain the transition $q \xrightarrow{N,g} p$, instead of the more complete $(q, N, g, p) \in \Delta$, where N is the action set and g is the guard of the transition. For every transition $q \xrightarrow{N,g} p$, we assume that $N \neq \emptyset$; that is, automata transitions can fire only if an action occurs.

Let $q_0 \rightsquigarrow^* q_n$ be a finite path as follows:

$$\left[V_{q_0}, q_0 \xrightarrow{N_1,g_1} q_1 \xrightarrow{N_2,g_2} q_2 \cdots q_{j-1} \xrightarrow{N_j,g_j} q_j, q_o \in Q_I\right]$$

With every transition $q_{j-1} \xrightarrow{N_j,g_j} q_j$ in $q_0 \rightsquigarrow^* q_n$, we associate a descriptor $\delta: (In, M_j) \mapsto (O, M_{j+1})$ $j \in [1 \cdots N]$, where *In* and *O* are input and output values, respectively.

6.5.2 Composing SDDS Automata

To compose two SDDS automata, we use the cross-product of two automata as follows.

Definition 2

Let $A_1 = (Q_1, \Sigma_1, \Delta_1, Q_{I_1}, M_1, V_1)$ and $A_2 = (Q_2, \Sigma_2, \Delta_2, Q_{I_2}, M_2, V_2)$ be two SDDS automata. Then the composition of these two automata is:

$$A_1 \bowtie A_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \Delta, \mathrm{Q_{I_1}} \times \mathrm{Q_{I_2}}, \mathrm{M_1} \cup \mathrm{M_2}, \mathrm{V_1} \cup \mathrm{V_2}),$$

where δ is defined by the following rules:

$$\frac{q_1 \xrightarrow{N_1, g_1}_1 p_1, q_2 \xrightarrow{N_2, g_2}_2 p_2, N_1 \cap \Sigma_2 = N_2 \cap \Sigma_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle}$$

This rule denotes the creation of a new transition whenever both automata A_1 and A_2 want to change their states simultaneously. The new transition gets $N_1 \cap N_2$ as its actions if $g_1 \wedge g_2$ is true.

$$\frac{q_1 \xrightarrow{N, g}_1 p_1, N_1 \cap \Sigma_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N, g} \langle p_1, q_2 \rangle}$$

This rule denotes the creation of a new transition whenever automaton A_1 wants to move from state q_1 to p_1 and automaton A_2 remains in state q_2 and is in idle mode.

$$\frac{q_2 \xrightarrow{N, g}_2 p_2, N_2 \cap \Sigma_1 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N, g} \langle q_1, p_2 \rangle}$$

Finally, this rule denotes the creation of a new transition whenever automaton A_2 wants to move from state q_2 to p_2 and automaton A_1 remains in state q_1 and is in idle mode.

6.5.3 User Dynamic Manager

In this design, for a given request, the user dynamic manager collects the required user-side dynamic attributes either from the requester (explicitly) or from the requester profile (implicitly). The goal of this is to expand a query to increase the chance of selecting more relevant web services for a user's request. Figure 6-3 depicts the SDDS user dynamic manager as an automaton. $Q = \{x; y; z\}$ is the set of states, and $\Sigma = \{Add, Find, Expand\}$ is the set of actions.

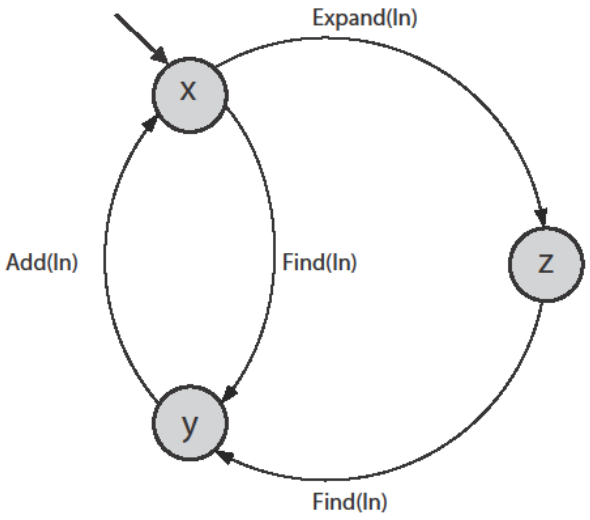


Figure 6-3: User Dynamic Manager Automaton

$Q_I = \{x\}$ is the initial state. The memory cells for the state are

$$M = \begin{Bmatrix} (P: \{p_1, p_2, \cdots, p_i\}; \\ D: \{d_1, d_2, \cdots, d_j\}; \\ S: \{s_1, s_2, \cdots, s_l\}) \end{Bmatrix},$$

where P is a set of web service profiles, D is a set of services in the dynamic repository, S is a set of services in static repository, and $D \subseteq S$. Each p_i and d_i has a locked status. The value function is defined as

$$V_x = \{P = \emptyset; D = \emptyset; S = UDDIBrowser()\}.$$

The transition relations Δ are as follows:

$$\left\{ \begin{array}{cc} & y \xrightarrow{Add(In)} x, \\ & \xrightarrow{Find(In),\left(\Lambda_{k=1}^{j} d_k.lock=false\right)} y, \\ x & x \xrightarrow{Expand(In)} z, \\ & z \xrightarrow{Find(In),\left(\Lambda_{k=1}^{j} d_k.lock=false\right)} y \end{array} \right\}.$$

The descriptor $\delta: (In, M) \longmapsto (O, V(M))$ is

$$\delta(In, M) \mapsto \begin{cases} \left(null, (P \cup \{f | \forall w \in D \wedge SAT(In, w), f = REL(In, w)\}; D; S)\right) if\ \delta = Add, \\ \left(\{w |\ w \in D \wedge SAT(In, w)\}, (P; D; S)\right) if\ \delta = Find, \\ \left(\{In = EXPAND(In)\}, (P; D; S)\right) if\ \delta = Expand. \end{cases}$$

We know that the *SAT(In; w)* function returns a true value if *w*, input web service *In*, is satisfied by requester parameters. *SAT( )* is defined as $SAT: (In, w) \mapsto \{true, false\}$ with $w \in S$. The *REL(In; w)* function returns a record to insert into the web service profile for those web services

that are relevant to input *In*. The *EXPAND(In)* function returns an input request that is expanded by the set of profile parameters *P*.

6.5.4 UDDI Discovery

UDDI discovery uses a standard UDDI web service registry to store and discover services. Modeling of UDDI discovery in automata is illustrated in Figure 6-4, where *Q = {1; 2; 3}* are the states, and the set of actions is $\Sigma = \{UFind, UAdd, Acknowledge\}$.

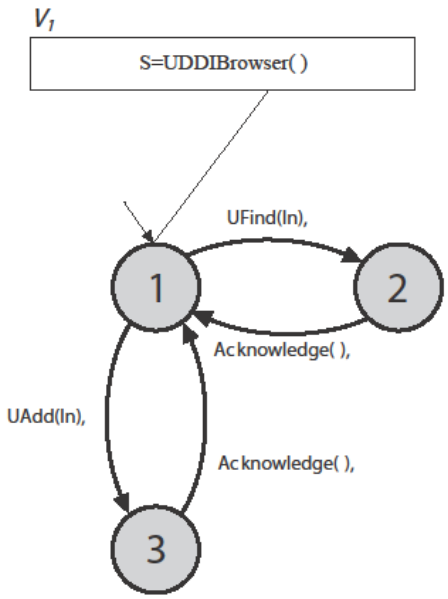


Figure 6-4: UDDI Discovery Automaton

$Q_1 = \{1\}$ and the memory cell set is

$$M = \{ (S : \{s_1, s_2, \ldots, s_l\}) \},$$

where *S* is a set of services in the static repository. The value function is defined as

$$V1 = \{ S = UDDIBrowser()\}.$$

The automaton transition relation (Δ) is

$$\left\{ \begin{array}{l} 1 \xrightarrow{UFind(In)} 2, \\ 1 \xrightarrow{UAdd(In)} 3, \\ 2 \xrightarrow{Acknowledge()} 1, \\ 3 \xrightarrow{Acknowledge()} 1 \end{array} \right\}.$$

The descriptor $\delta: (In, M) \mapsto (O, V(M))$ is

$$\delta(In, M) \mapsto \begin{cases} (\{w | w \in S \wedge SAT(w, In)\}, (S)) \; if \; \delta = UFind \\ (null, (S \cup \{In\})) \; if \; \delta = UAdd \\ (null, (S)) \; if \; \delta = Acknowledge. \end{cases}$$

Here, the *SAT(w; In)* function returns a true value if *w*, input web service, is satisfied by requester parameters *In*. *SAT( )* is defined formally by $SAT : (w, In) \mapsto \{true, false\}$ with $w \in S$.

6.5.5 Service Dynamic Manager

The service dynamic manager checks the QoS in the existing public registries, based on their dynamic attributes. If the service passes the quality phase, then it is registered in the dynamic repository. The dynamic repository is updated periodically by the service dynamics manager. Modeling of the service dynamic manager in automata is illustrated in Figure 6-5, where *Q = {a, b, c, d, e}* is the state, and the set of actions is *Σ = {Interval, Sensor, AcknowledgeSDM, UFind, Update}*.

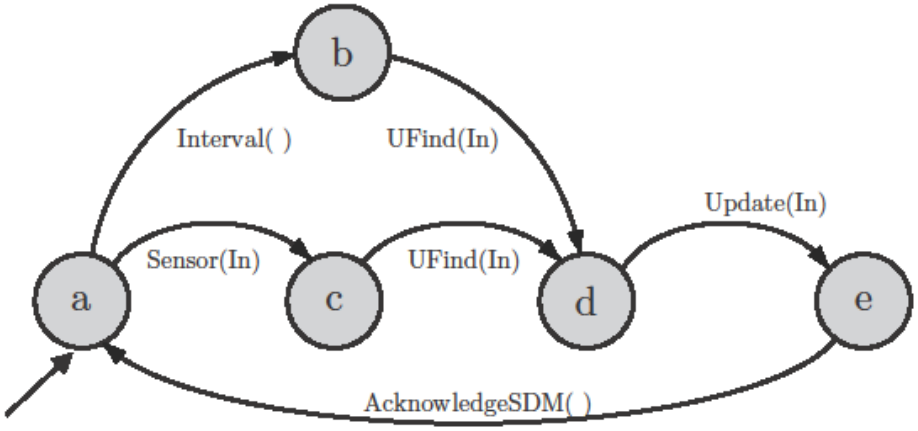


Figure 6-5: Service Dynamic Manager Automaton

For the initial state $Q_I = \{a\}$, the memory cell set is

$$\begin{Bmatrix} (D : \{d_1, d_2, \cdots, d_j\}; \\ S : \{s_1, s_2, \cdots, s_l\}) \end{Bmatrix},$$

where *D* is the set of services in dynamic repository, *S* is the set of services in the static repository, and $D \subseteq S$. Each d_i has a locked status. The value function is defined as

$$V_a = \{D = \emptyset, S = UDDIBrowser( )\}.$$

The automaton transition relation set (Δ) is

$$\left\{ \begin{array}{c} a \xrightarrow{Interval(In)} b, \\ a \xrightarrow{Sensor(In)} c, \\ b \xrightarrow{UFind(In)} d, \\ d \xrightarrow{Update(In),\left(\Lambda_{k=1}^{j} d_k.lock=true\right)} e, \\ e \xrightarrow{AcknowledgeSDM( ),(\Lambda_{k=1}^{j} d_k.lock=false)} a, \\ c \xrightarrow{UFind(In)} d \end{array} \right\}.$$

The descriptor $\delta: (In, M) \mapsto (O, V(M))$ is

$$\delta(in, M) \mapsto \begin{cases} \big(\{ w \mid w \in S \wedge SAT(w, In)\}, (D; S)\big)\ if\ \delta = UFind, \\ \big(null, (D; S)\big)\ if\ \delta = Interval, \\ \big(null, (D; S)\big)\ if\ \delta = Sensor, \\ \big(null, \big((D \setminus \{w\}) \cup \{In\}; S\big)\ if\ \delta = Update\ \wedge \{ w \mid w \in S \wedge SAT(In, R)\}, \\ \big(null, \big((D \setminus \{w\}); S\big)\ if\ \delta = Update\ \wedge \{ w \mid w \in S \wedge !\, SAT(In, R)\}, \\ \big(null, (D; S)\big) if\ \delta = AcknowledgeSDM. \end{cases}$$

Here, the *SAT(In;R)* function returns a true value if *In*, input web service, is satisfied by requester parameters *R*. *SAT( )* is defined formally by $SAT: (In, R) \mapsto \{true, false\}$ with $w \in s$.

6.5.6 SDDS Goal Properties

We show a user dynamic manager with $A_1 = \big(Q_1, \Sigma_1, \Delta_1, Q_{I_1}, M_1, V_1\big)$ in Figure 6-3. UDDI discovery is shown with $A_2 = \big(Q_2, \Sigma_2, \Delta_2, Q_{I_2}, M_2, V_2\big)$ in Figure 6-4 and a service dynamic manager is shown with $A_3 = \big(Q_3, \Sigma_3, \Delta_3, Q_{I_3}, M_3, V_3\big)$ in Figure 6-5. SDDS uses database sharing between these components. Composing A_1, A_2, and A_3 gives $B = \big(Q_B, \Sigma_B, \Delta_B, Q_{I_B}, M_B, V_B\big)$:

$$B = A_1 \bowtie A_2 \bowtie A_3$$

The *B* automaton—that is, the composition of A_1, A_2, and A_3—should fulfill SDDS goals. However, sharing databases with multiple components may lead to data inconsistency and invalid results. The typical example is that a record may be simultaneously read and written in our repositories. One resolution is to use mutual exclusion in the product algorithm so that databases cannot be accessed by more than one component at one time, ensuring correctness, consistency, and fairness. For instance, UDDI discovery—*Update(In)*—running with service dynamic manager—*UFind(In)*—may lead to an invalid result.

$$d1x \xrightarrow{Update(In)UFind(In)} e1y\ d1x, e1y \in Q_B$$

Furthermore, two different functions in a component cannot run on a shared memory set. Accordingly, this goal can be shown as below.

$$\text{For each } p_1 \xrightarrow{N,g} p_2 \in \Delta, \exists\ a_1 \in \Sigma_1, a_1 \in N \Longrightarrow \forall a \in N, a \neq a_1, (a \notin \Sigma_1)$$

$$\text{For each } p_1 \xrightarrow{N,g} p_2 \in \Delta, \exists\ a_1 \in \Sigma_2, a_1 \in N \Longrightarrow \forall a \in N, a \neq a_1, (a \notin \Sigma_2)$$

$$\text{For each } p_1 \xrightarrow{N,g} p_2 \in \Delta, \exists\ a_1 \in \Sigma_3, a_1 \in N \Longrightarrow \forall a \in N, a \neq a_1, (a \notin \Sigma_3)$$

An example of such a transition is shown in Figure 6-6, where both *UFind(In)* and *UAdd(In)* are functions of UDDI discovery working on the UDDI repository.

$$b1x \xrightarrow{UFind(In)UAdd(In),false} d1z\ b1z, d1z \in Q_B$$

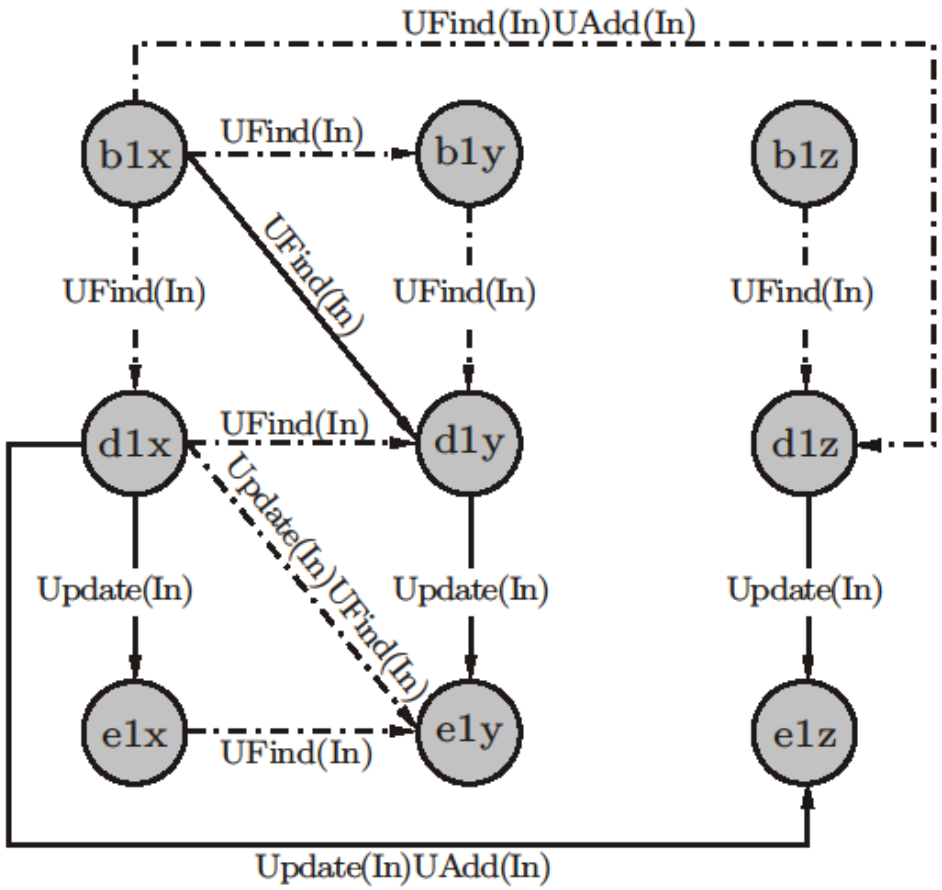


Figure 6-6: Partial View of the SDDS Automaton

Enforcing mutual exclusion during the composition of components guarantees data validity; however, this solution does not scale well and is slow to accommodate service discovery when speed and flexibility have become competitive advantages. About 281 transitions that may cause deadlock and starvation in SDDS can be trimmed from the *B* automaton; some of these transitions are shown by dotted lines in Figure 6-6. Thus, the *B* automaton shows that the composed system does not satisfy the goal of efficiency due to constraints and invalid results. An example of one of these transitions is shown below:

$$d3x \xrightarrow{Update(In)Find(In),(In=d_k \wedge d_k.lock=true)\wedge(\Lambda_{k=1}^{j} d_k.lock=false)} e2x\ d3x, e2x \in Q_B$$

The guard of this transition is false because $d_k.lock = true \ \wedge k = [1..j]$ and . $\left(\Lambda_{k=1}^{j} d_k.lock = false\right)$. However, this transition achieves high efficiency if the *Find()* action searches all dynamic repository elements except for the member that *Sensor()* works on. In other words, we are changing the granularity of mutual exclusion to the level of work sets (records of sharing resources). Thus, to achieve high efficiency while preserving data validity, the system must be permitted to run when two actions want to access a common resource with different work sets.

If Γ: $2^M \mapsto 2^M$ is a function that returns a subset of a component work set that will be changed by another component, then we have the following:

$$(\Gamma(M_1) \cap \Gamma(M_3) = \emptyset)$$

We modify SDDS by supplementing a synchronization and coordination method among components. We enrich our framework with a controller that guarantees not only validity but also satisfactory performance. The controller is a web-based synchronization component that enforces valid communications while supervising more transitions in the composed system, thus improving scalability [21, 22]. Figure 6-7 depicts the resulting model that is used for our implementation.

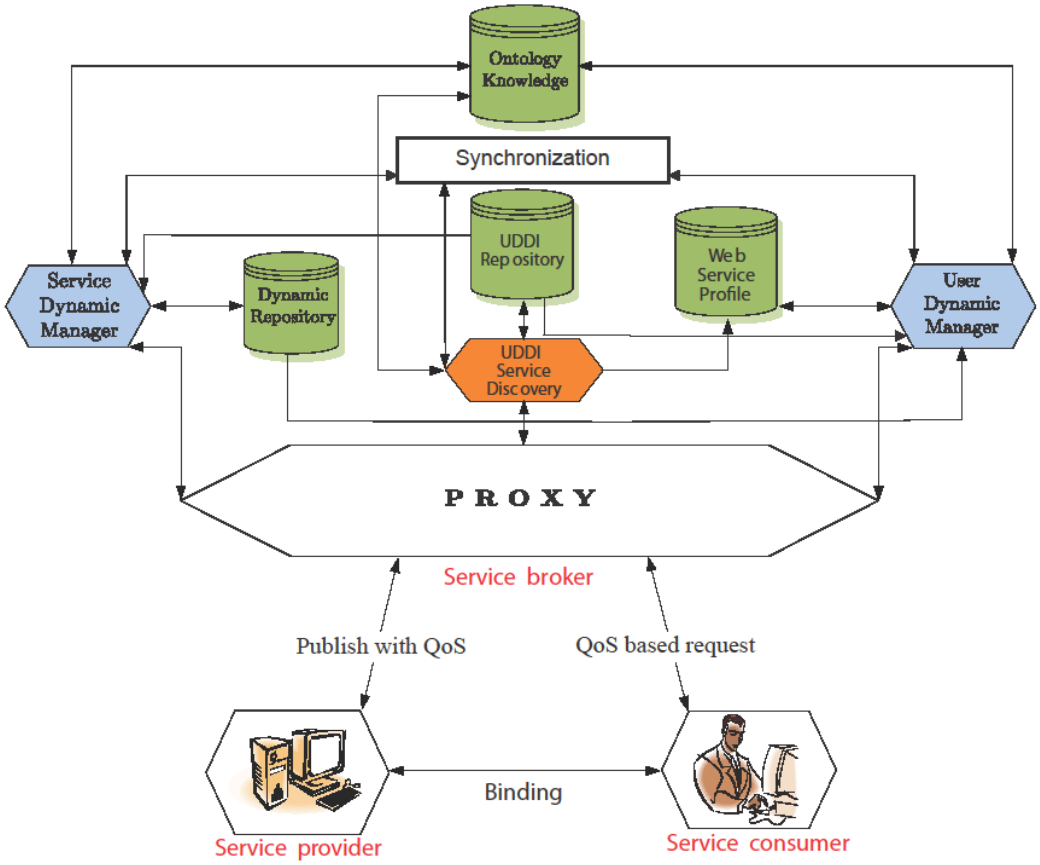


Figure 6-7: Modified SDDS Architecture

6.6 Conclusions and Future Work

To achieve high-quality dynamic web service discovery, QoS attributes must be taken into account. Because the value of QoS attributes varies in different domains and under different situations, we proposed an SOA-based model to select appropriate services in a specific domain and context. Our model, SDDS, is an effective technology that measures the relevance of services to a particular context. We investigated the validity of our framework by analyzing the behavior between and within its important elements and by modeling SDDS with automata. We found that the previous design of SDDS did not maintain the validity of registered services because of several invalid transitions. This paper discussed this problem and introduced as a solution an additional web-based synchronization component to avoid the incorrect transitions.

In the future, for our web service discovery framework, we will look at improving the quality of returned results through the following studies:

- a comprehensive measurement technique to rank the discovered services (i.e., factors that affect the result of service selection need to be ranked using a more robust mechanism)
- a broad algorithm to support all types of progressive processing, such as user preferences and arbitrary dimensionality, because all currently used algorithms have some serious shortcomings that limit their applicability in practice
- a self-adaptive mechanism to evaluate the service and user context and to change the service selection mechanism if better options are available
- an SDDS infrastructure that can search in a P2P environment to support large-scale, decentralized data, in which each dynamic repository acts as a peer on the network

References

[1] N. W. Lo and C.-H. Wang, "Web service QoS evaluation and service selection frameworks: a proxy-oriented approach," TENCON 2007, IEEE Region 10 Conference, pp. 1–5, 2007.

[2] S. Kim and M. C. Rosu, "A survey of public web services," E-Commerce and Web Technologies, pp. 96–105, 2004.

[3] Microsoft, IBM Research, Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl (2001).

[4] L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, and H. Mei, "Personalized QoS prediction for web services via collaborative filtering," IEEE International Conference on Web Services (ICWS), pp. 439–446, 2007.

[5] R. Tabein and A. Nourollah, "Dynamic broker-based service selection with QoS-driven recurrent counter classes," International Conference on Service System and Service Management, pp. 1–6, 2008.

[6] R. Hu, J. Liu, Z. Liao, and J. Liu, "A web service matchmaking algorithm based on an extended QoS model," IEEE International Conference on Networking, Sensing and Control (ICNSC), pp. 1565–1570, 2008.

[7] A. Pahlevan and H. A. Müller, "Static-Discovery Dynamic-Selection (SDDS) approach to web service discovery," IEEE Congress on Services (ICWS), pp. 769–772, 2009.

[8] W. Fokkink, Introduction to Process Algebra (2nd edition). Springer-Verlag, 2007.

[9] M. Dam, "Temporal logic, automata, and classical theories—an introduction," Notes for the 6th European Summer School in Logic, Language and Information, 1994.

[10] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, "Middleware for distributed context-aware systems," OTM Confederated International Conferences, pp. 846–863, 2005.

[11] T. Gua, H. K. Punga, and D. Q. Zhang, “A service-oriented middleware for building context-aware services,” Journal of Network and Computer Applications, vol. 28, pp. 1–18, 2008.

[12] H. Kim, Y.-J. Cho, and S.-R. Oh, “CAMUS: a middleware supporting context-aware services for network-based robots,” IEEE Workshop on Advanced Robotics and Its Social Impacts, pp. 237–242, 2005.

[13] P.-O. Osland, B. Viken, F. Solsvik, G. Nygreen, J. Wedvik, and S. E. Myklbust, “Enabling context-aware applications,” Proceedings of ICIN2006: Convergence in Services, Media and Networks, 2006.

[14] B. Han, W. Jia, J. Shen, and M.-C. Yuen, “Context-awareness in mobile web services,” Parallel and Distributed Processing and Applications, pp. 519–528, 2008.

[15] P. Costa and L. Botelho, “Generic context acquisition and management framework,” First European Young Researchers Workshop on Service Oriented Computing, 2005.

[16] A. Pahlevan and H. A. Müller, “Self-adaptive management of web service discovery,” 8th European Conference on Web Services (ECOWS), 2010.

[17] OASIS Consortium, UDDI Standard. http://uddi.xml.org (2004).

[18] L. D. Alfaroand and T. A. Henzinger, “Interface automata,” ACM SIGSOFT Software Engineering Notes, vol. 26, no. 5, p. 120, 2001.

[19] T. Liu and G. S. Zeng, “Adaptation of mismatching services based on labelled interface automata,” Proceeding of 5th International Conference on Semantics, Knowledge and Grid (SKG), pp. 326–329, 2009.

[20] M. Sama, S. Elbaum, F. Raimondi, D. S. Rosenblum, and Z. Wang, “Context-aware adaptive applications: fault patterns and their automated identification,” IEEE Transactions on Software Engineering, vol. 36, no. 5, pp. 644–661, 2010.

[21] W. Wonham and P. Ramadge, “Modular supervisory control of discrete event systems,” Mathematics of Control, Signals, and Systems (MCSS), vol. 1, pp. 13–30, 1988.

[22] R. Sengupta and S. Lafortune, “An optimal control theory for discrete event systems,” SIAM Journal on Control and Optimization, vol. 36, pp. 519–528, 1998.

7 Characterizing Policies That Govern Service-Oriented Systems

Priyanka Gupta, University of Victoria, Canada (kpgupta@cs.uvic.ca)
Qin Zhu, University of Victoria, Canada (qinzhu@csc.uvic.ca)
Hausi A. Müller, University of Victoria, Canada (hausi@cs.uvic.ca)

Abstract

Service-oriented architecture (SOA) governance not only ensures that the concepts and principles for service orientation and its distributed architecture are managed appropriately and deliver on the stated business goals for services but also controls the evolution of service-oriented systems. Evolving services must be able to manage their own actions based on high-level global business goals and low-level local rules. One way to specify such goals is in the form of policies, which are operating rules to orchestrate and maintain order, security, and consistency throughout the service life cycle. In this paper, we categorize policies that govern service-oriented systems on the basis of their empirically observable behavior and their applicability to phases of the service life cycle. With this classification, we aim to provide a comprehensive overview of existing policies facilitating policy-based governance and evolution in distributed service-oriented environments.

7.1 Introduction

To cope with the ever-increasing complexity of managing distributed, heterogeneous computing systems, it has been proposed that such systems should manage their own behavior in accordance with high-level objectives from human administrators and enterprise-level business requirements [11]. Service-oriented architecture (SOA) is an architecture paradigm that enables a business to organize, define, implement, and orchestrate distributed services in such a way that their value can be directly correlated with the business goals and drivers of the enterprise [8]. SOA governance provides guidelines to ensure that the efforts during design, development, deployment, and operations of a service-oriented system come together to meet enterprise requirements.

To attain a structured approach to governing service-oriented systems, IBM has defined SOA governance using six dimensions. Governance is (1) guided by policies, (2) controlled by standards, (3) managed by a responsible party, (4) implemented by some process or procedure, (5) supported by a mechanism or method, and (6) monitored by a set of metrics [3]. Policies play a key role in enabling governance in any service-oriented environment. By adding policies, points of control, and agility for both business and IT, SOA is made more palatable, thereby aiding and possibly accelerating the adoption of SOA solutions. At the highest level, policies can be a simple set of requirements expressed in natural language. In general, policies express requirements that a well-defined set of services ought to follow or system constraints or capabilities that define the interaction between consumers and providers of services. As the service life cycle advances from design to operations, policies are refined from high-level goals into low-level objectives and rules understandable by machines.

Gleason et al. argue that policies are often created in an ad hoc manner and communicated through mechanisms that are incompatible with the underlying service model and architecture [23]. There is a need to characterize and categorize the plethora of diverse policies in an enterprise to realize controlled governance in service-oriented systems with sustainable benefits to the enterprise. Our literature review reveals various coarse-grained classifications of policies including design and runtime policies as well as business, process, and technical policies. Service governance spans and manages the entire life cycle of services. Hence, it is appropriate to classify policies according to service life-cycle phases.

To characterize policies that govern service-oriented systems, we collected and analyzed a large number of papers dealing with governance and policies for service-oriented systems. This paper presents the results of that survey. The life-cycle phases used in the classification are based on the Smart Grid Maturity Model (SGMM) and other SOA life-cycle models including SOMA, SOAF, SOMA-ME, and SODDM [3]. The categorization of policies reveals how different parts contribute to the governance architecture as a whole. Section 7.2 presents an overview of policies and outlines our research methodology. Sections 7.3 to 7.9 detail the different categories of policies. Section 7.10 discusses how governance policies relate to autonomic computing policies in the context of software evolution. Section 7.11 summarizes the survey and presents the distribution of papers by policy type. Finally, Section 7.12 concludes the paper.

7.2 Policies as Governance Components

Policies are guidelines that a well-defined set of services ought to follow or system constraints or capabilities that determine the interaction between consumers and providers of services. There are three types of policies: *absolute*, *derived*, and *compositional* [2]. Policies that are enforced before the service is deployed are referred to as *design-time policies*; policies that are enforced while the service is in operation or in the registry are referred to as *runtime policies*. To characterize different types of policies, we surveyed papers that deal with SOA governance in general and SOA governance policies in particular. This paper summarizes policies featured in these papers and categorizes them according to six phases of the service life cycle as defined by ITIL and the SGMM, resulting in six policy types: *service planning, service design, service development, service transition, service operation, monitoring,* and *multilevel policies* [3, 19].

7.3 Service Planning Policies

Service planning is usually the first stage of service-oriented development, wherein plans are made to engender reuse across lines of business and to facilitate agility and quick response to market opportunities [3]. To ensure that the transformation to SOA is smooth and efficient, strategic decisions have to be made. This process involves budgetary control mechanisms, clearly identified roles and responsibilities, compliance with industrial and the most appropriate standards, and adherence to reference architectures and design patterns [1, 2, 3, 31, 32, 42]. We categorize the resulting policies below. The diagram depicted in Figure 7-1 summarizes the service planning policies.

Organizational policies define decision rights and responsibilities of people involved in the development of services within and across ownership domains [44]. They also define the interaction patterns between organizations and across an enterprise.

Architectural policies address issues related to the framework of business processes and services and help control service development costs by identifying specific services or processes known to produce favorable results [1, 2, 3, 4, 5].

- **Enterprise architecture (EA) policies** are related to business strategies such as EA goals, objectives, and strategies. They also include the relationship of EA to capital planning and program management.
- **Business process architecture policies** refer to the logical and physical structure of infrastructure and components and relate strategic goals and business processes.

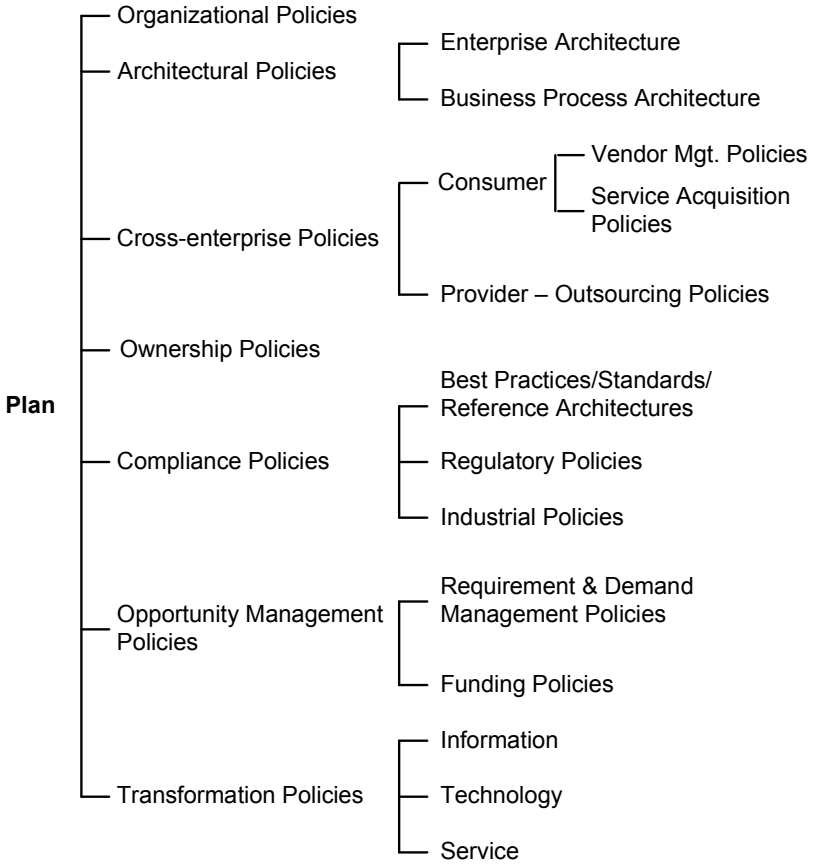


Figure 7-1: Service Planning Policies

Cross-enterprise policies concern the high-level relationships between consumers and providers.

- **Vendor management policies** provide consumers guidance with respect to vendor relationships that depends on operational, reputational, and strategic risks [3].
- **Service acquisition policies** provide guidance for acquiring services from providers depending on performance, cost, and schedule needs and requirements [18].
- **Outsourcing policies** mandate the assessment and management of risks associated with business process and service outsourcing.

Ownership policies define the owners of the different artifacts in the development of a service [2]. This is particularly useful during the delivery or consumption of services across distributed services both within and between ownership domains.

Compliance policies deal with conformance with respect to domain-dependent industry standards, regulatory policies and standards, best practices, and reference architectures [2]. Sarbanes–Oxley is an example of an industrial standard.

Opportunity management policies comprise requirements and demand management policies as well as funding policies. The former facilitate identifying, specifying, and prioritizing requirements based on the high-level business strategy, while the latter manage the investment of funds within and across the enterprise to maximize profits [2].

Transformation policies include the identification, classification, and prioritization of service domains and artifacts that satisfy the high-level business goals and strategies. The subcategories include the following:

- **Information transformation planning policies** identify needs to exchange and convert information among domains and services.
- **Technology transformation planning policies** specify the conformance of hardware and software with respect to life-cycle and service reference architectures, taking into consideration high-level business goals.
- **Service transformation planning policies** are useful in relating and correlating service domains, services, and business processes [2, 45].

7.4 Service Design Policies

Service design includes activities required to specify a service and smoothly transition into development or expose a service from an existing application [1, 2, 3, 27, 28, 29, 30, 31]. Services must be designed so that they meet the business requirements and goals set by consumers and providers alike. With this perspective, service design policies provide guidelines for models that demonstrate that the services conform to desired business goals, as shown in Figure 7-2.

Architectural policies include the identification and specification of design patterns that will be used for the implementation of services.

- **Application architecture policies** provide guidelines for specifying the functional interfaces between application packages, databases, and middleware systems and involve design patterns and domain-specific languages.
- **Technology architecture policies** provide interoperability guidelines for computing and network infrastructure [2].
- **Service architecture policies** provide a framework for refining the business process architecture into a set of services. In particular, these policies define how services are to be reused, transformed, and created. They also define the high-level interaction among services.
- **Information architecture policies** focus on the process of modeling and involve information models, semantic models, and logical and physical data models (i.e., data architecture policies) that are needed to support business processes and address

interoperability, integration, consolidation, and sharing of resources by correlating business processes to business goals (i.e., message flow policies).

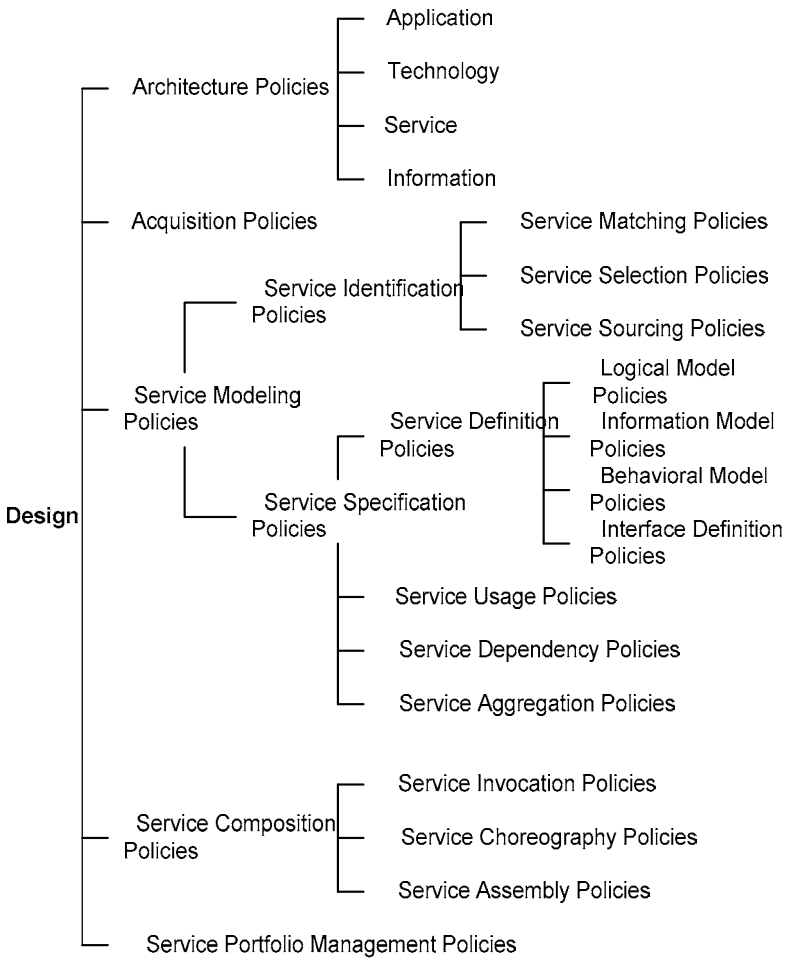


Figure 7-2: Service Design Policies

Acquisition policies comprise technology acquisition policies, such as the entire technical infrastructure needed to meet the requirements and demands of business, and resource acquisition policies, such as all the resources needed (e.g., people, funds, third-party services, and contracts).

Service modeling policies are probably the most important policies in this phase. They include *service identification policies* and *service specification policies*.

- **Service identification policies** provide guidelines for consumers to identify high-quality services that are of an appropriate granularity.
  - **Service matching policies** involve selection of effective algorithms to match the requirements from the pool of services.
  - **Service selection policies** relate to selecting the best service based on performance, infrastructure, and cost requirements.
  - **Service sourcing policies** facilitate the evaluation of potential suppliers and proactively narrow the list of suppliers to optimize business goals [49].

- **Service specification policies** ensure that the design and implementation meet the requirements, define the order in which the capabilities of a service can be called upon, and specify interfaces and constraints (e.g., quality of services, or QoS) properly [47].
  - **Service definition policies** address the messaging structure, port types, and data types present in the message flow using schema definition languages (e.g., XML schema).
  - **Logical model policies** describe service types, service domains, and complete service functionality. WS-Policy with its XML schema is a good example of the information model policies.
  - **Behavioral model policies** address the side effects of service operations (i.e., input or output by the service).
  - **Interface definition policies** provide guidelines for defining interfaces to minimize coupling and maximize cohesion among services. For example, if two services are logically cohesive, then the service interface ought to contain only port types.
- **Service usage policies** govern the usage of a service by the consumer. Rules for the costs of using a service, prohibited users, server abuse, and spam recognition are clearly specified within the service usage policy.
- **Service dependency policies** enable designers to clearly define the dependencies or relationships among services in a composite service. Also the order in which the services will be executed must be specified [19, 46].
- **Service aggregation policies** specify the granularity and reuse level of a service set to guarantee satisfaction and optimization of business objectives. Using protocols for representation, identity, and communication, they specify the degree and type of coupling and cohesion among related services.

Service composition policies specify the necessary rules for the aggregation of multiple services into a single composite service [7, 35]. These policies include the following:

- **Service invocation policies** provide rules for invoking services.
- **Service choreography policies** provide rules for orchestrating services. In particular, these rules specify interaction protocols, common behavior of services (e.g., using web services choreography description language, or WS-CDL), and agreements among services.
- **Service assembly policies** provide regulations to create new services based on architectural standards and design patterns [2].

Service portfolio management policies govern the management of collections of assets and artifacts for building and delivering services. Policies for managing funds, resources, applications, hardware, software infrastructure, and services fall under this category [16].

7.5 Service Development Policies

Service development is the phase in which teams develop and test the composite application in an iterative manner and upload the artifacts to the enterprise service repository [1, 2, 3, 8, 28, 29]. Service development policies, as depicted in Figure 7-3, guide the development of services to optimize the profits obtained from developing service-oriented systems.

Technology standards compliance policies spell out compliance requirements for services including web services standards such as WS-I, Simple Object Access Protocol (SOAP), representational state transfer (REST), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI).

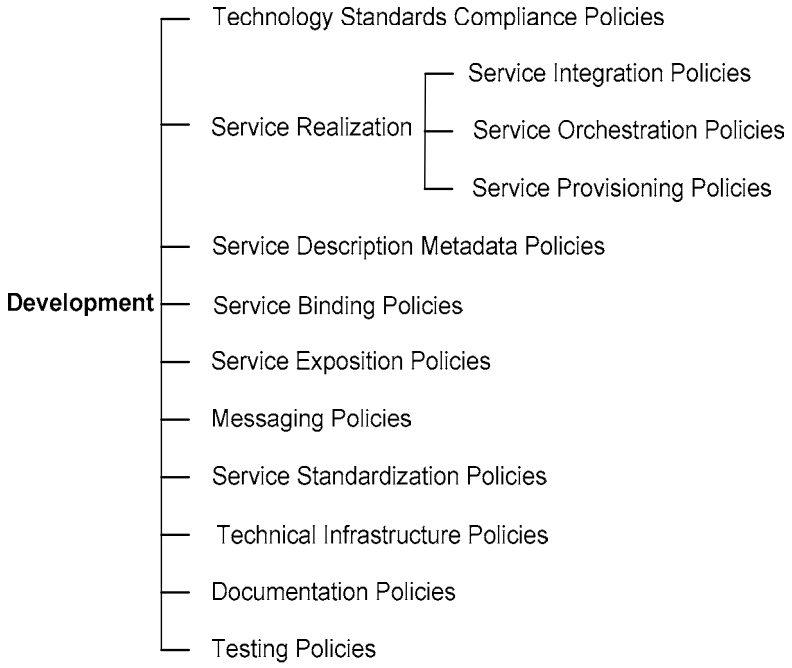


Figure 7-3: Service Development Policies

Service realization policies provide guidelines for developing services from the ground up or from legacy components (e.g., wrapping).

- **Service integration policies** govern the process by which individual services or collaborating services or legacy systems organize themselves to implement the entire service application. This category includes legacy system integration and wrapping policies, application server policies (e.g., Java, Oracle OC4J, or Apache), and integration server policies (e.g., WebSphere, webMethods, or B2B) [21]. These policies can vary considerably depending on the type of integration (e.g., UI integration, point-to-point integration, CGI integration, or data integration). For example, if the intention is to integrate UNIX environments with Microsoft technologies, and if the solution provided is Windows Services for UNIX, then policies and standards such as FTP, HTTP, IP, IPSec, LDAP, PKI, POSIX standards, S/MIME, SSL, TCP, Telnet, TLS, and XML must be taken into consideration [48].
- **Service orchestration policies** specify the interaction between services at the message level, assuring controlled execution flow of services [33]. These include XML-based process standard definition language policies such as Business Process Execution Language (BPEL) for web services.

- **Service provisioning policies** are expressed in service provisioning markup language (SPML) and are created by administrators based on the way that services are customized (e.g., using the IBM Tivoli Identity Manager) [52].

Service description metadata policies provide guidelines for user-defined metadata used to enhance the service metadata to explain its semantics [2, 53]. For example, the web services registry and repository (WSRR) policy defines service description entities (i.e., physical, logical, and conceptual) and the way in which these entities are related, classified, and defined.

Service binding policies specify the ways in which services must bind with other services [2, 54]. Examples include BasicHttp Binding, MessageBinding, SOAP binding, and SOAP addressing. These also include message encoding policies such as SOAP encoding or Message Transmission Optimization Mechanism (MTOM) encoding.

Service exposition policies include the different ways in which services can be exposed in a registry/repository (e.g., JSF or ESB) [55].

Messaging policies facilitate the interaction between services and involve standards and protocols such as JMS, ebXML, ActionScript Messaging Format (AMF), and message exchange patterns [2]. Messaging architecture policies include routing protocols, request-response, publish-subscribe, message channel patterns, and transformation patterns. Static and dynamic binding of services relies on messaging architectures [56].

Service standardization policies enable appropriate service interface definitions. For example, WSDL definitions specify the granularity, messaging structure, data structures, and naming conventions [1, 2].

Technical infrastructure policies specify the usage of a particular tool or technology. Examples are shown in Table 7-1 [1, 2, 3].

Table 7-1: Technical Infrastructure Policies

Infrastructure policies	Network infrastructure, application infrastructure, and system infrastructure policies
Registry and repository	WSRR, enXML, UDDI, and XML registry policies
Web services management	.NET, SilverLight, and Ajax Client Libraries
Service development tools	WSDL editor and Apache Axis policies, and web service testing tools—all the tools that help in developing, running, and maintaining a service-oriented system
Coding policies	Policies based on best practices
IDE policies	Eclipse, NetBeans, and Visual Studio
Application server policies	JBoss policies, WebSphere Application Server, and Zend Policies
Database policies	Storage, access, and retrieval polices

Documentation policies define time frames for when the service artifacts must be uploaded in the repository for reference and documentation styles (e.g., Javadoc) [57].

Testing policies outline testing and validation requirements including message and schema validation, WSDL validation, performance and load testing, end-to-end service and application integration testing, validation of BPEL orchestration, and protocol testing as well as validation of SLAs and QoS measures [2, 36].

7.6 Service Transition Policies

Service transition occurs after development and is when the services are deployed. In this section we concentrate on release and deployment of services [1, 2, 3, 8, 9]. Service transition policies provide guidelines for deploying services onto the repository/registry at the provider side as shown in Figure 7-4.

Service deployment policies come into play when releasing services to other participants, enterprises, applications, or services. These policies are further decomposed into service configuration, publishing, versioning, and infrastructure policies.

- **Service configuration policies** define initial settings and arrangements of services, as well as other artifacts that affect service performance and functionality.
- **Service publishing policies** deal with service operations and service interfaces to aid service identification during service discovery, including communication protocols, data structures, and messages for service publishing.
- **Versioning policies** concentrate on maintaining the correct version of services being executed in a highly distributed environment and informing the consumers about service version updates. Version control is critical for service deployment. Service versioning involves XML extensions, UDDI subscriptions, and service adapters, and WS-Addressing providers must specify the different ways in which versioning is handled.
- **Infrastructure policies** specify the system necessary for the identification and execution of services. These include the messaging protocols, registry and middleware policies, and application server policies [39].

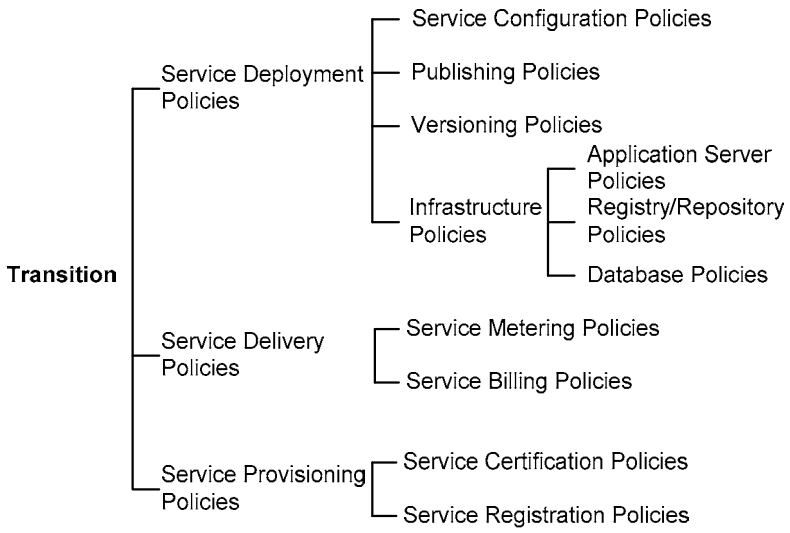


Figure 7-4: Service Transition Policies

Service delivery policies ensure less ambiguity in the conditions set between service providers and consumers.

- **Service metering policies** provide guidelines for measuring service usage and defining measures (e.g., subscription rate, usage event, quality, precision, leasing, lifetime, and hidden costs) for billing the clients.
- **Service billing policies** provide guidelines for customers and providers to agree on a billing scheme for services rendered. These policies also include retrieval, payment, and disbursement of payment to service providers.

Service provisioning policies in this phase include the following:

- **Service certification policies** specify properties to facilitate prediction of the behavior of the service.
- **Service registration policies** specify how to publish services in the UDDI—an XML file that specifies the path to the service DLL, the interface that defines the service contract, the class that implements the contract, and environment variables that the service needs [14].

7.7 Service Operations Policies

In the service operations phase, services are managed, supported, controlled, and adapted based on changing business contexts [3, 7, 8, 22]. Policies for this phase are shown in Figure 7-5.

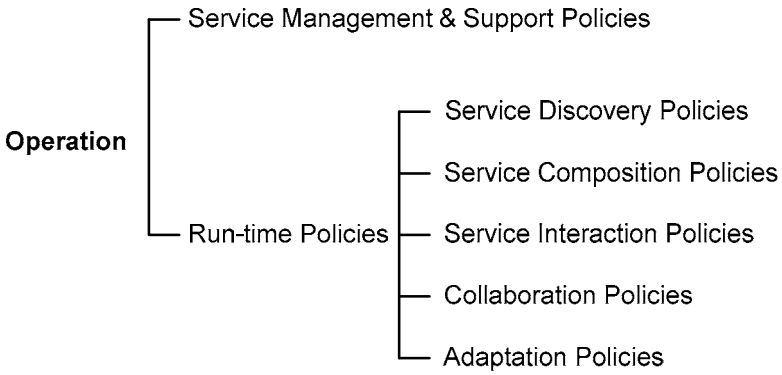


Figure 7-5: Service Operations Policies

Service management and support policies comprise all policies related to the following [2, 4, 12, 15, 38, 40, 41]:

- Change management policies
- Variability management policies
- Data management policies
- Configuration management policies
- Release management policies
- Knowledge management policies
- Asset management policies
- Incident management policies
- Capacity management policies

- Problem and fault management policies
- Availability management policies

Runtime policies are critical for the successful delivery and operation of services and deal with the discovery, composition, interaction, collaboration, and adaptation of services [1, 2, 3, 4].

- **Service discovery policies** assist in service discovery based on interface and behavioral models and operations [14]. This also involves interface, structure, and behavior patterns for matching services. UDDI policies and WS-Policy play major roles here.
- **Service composition policies** enable the realization of flexible and adaptable applications by properly selecting and combining components based on the user request and context. This also allows runtime integration of service endpoints (e.g., WS-Composition or CORBA policies) [34].
- **Service interaction policies** control the selection of services, service access, and message exchange between services (e.g., WSDL policies, Windows communication foundation (WCF) policies, or Apache Felix policies) [24].
- **Collaboration policies** accomplish tasks such as negotiation of collaboration protocols to be established between services at runtime and management of interaction of services at runtime [51].
- **Adaptation policies** facilitate service evolution due to changes in the underlying context, infrastructure, business rules, and environment [5, 6, 17, 25, 40]. These policies are critical for self-adaptive and self-managing service-oriented systems (e.g., dynamic attributes).

7.8 Monitoring Policies

Monitoring policies are generally policies that deal with the monitoring of resources, applications, and services in the organization for nonfunctional requirements including QoS, security, performance, and compliance [2, 10, 26, 40].

7.9 Multilevel Policies

Multilevel policies apply to multiple or all phases of the service development life cycle [2, 4]. Multilevel policies come in three different forms: business (plan time), process (design time), and technical (runtime). A security policy can consist of "Customer Trust" at its highest level and can be enforced at runtime using the WS-Security standard, SAML, XML Signature, or XML Encryption. Similarly, an SLA can be "Meet subscriber requests and demands quickly" at its highest level, and at a lower level it could be "Server response time must be less than 10 milliseconds."

Policy-based management for SOA governance is an emerging research area as it demonstrates capabilities that provide a solid foundation for service interoperability and evolution.

7.10 Governance Policies and Their Implication for Service Evolution

The preceding sections classified policies according to service life-cycle phases. This approach is appropriate because SOA governance spans the entire service life cycle, from early development to deployment and operation. Such an all-encompassing approach to policy specification and

management ensures that every phase of the service life cycle contributes to the overall governance goals of identifying, assessing, building, and managing business services and solutions so that they yield high value. However, the enforcement of a particular policy will vary with its business goals and its life-cycle phase. SOA governance is a level above the basic SOA infrastructure to ensure that the concepts and principles for service orientation and its distributed architecture can be controlled to deliver on the stated business objectives.

One cross-cutting aspect of the service life cycle not discussed so far is the evolution of a service-oriented system and its policies [37]. Based on the OASIS SOA reference architecture v3, SOA—just like other architectural paradigms—is subject to change and evolution. Setting policies that allow change management and evolution, establishing strategies for change, resolving disputes that arise, and ensuring that SOA-based systems continue to fulfill the goals of the business are all reasons why governance is important to SOA.

One approach to control the evolution of service-oriented systems and manage their flexibility is to introduce levels of indirection. Often a three-level hierarchy is employed to orchestrate policies. One prominent example is IBM's Autonomic Computing Reference Architecture (ACRA) [20]. Another approach to foster evolution of a service-oriented system is to equip its services with capabilities so that they can manage themselves and thus are able to adapt to changing environments. Such capabilities can be in the form of governance controllers and policies that realize control objectives. Control objectives are often reasonably nonspecific, in that they describe the ultimate goal but do not define the actual mechanisms or processes (controls) that are required to achieve this objective [13].

Governance policies make governance actionable, and monitoring policies provide continuous service improvement. Policies are not immutable and can be modified on the basis of feedback obtained from the later stages of the service life cycle. To optimize business functions, service-oriented systems are extensively instrumented to keep track of useful figures, such as execution time, availability, throughput, latency, and resource consumption. SOA governance processes use such figures to identify trends, adjust policies and processes, and manage service levels accordingly. In particular, such processes monitor and control the effects and outcomes of policies and processes using feedback loops. Selected results of SOA policies and processes are fed back to a controller, which then decides whether there is a need to adapt policies or processes in order to optimize outcomes [43].

Feedback loops can also be introduced to monitor policy changes and then evolve the underlying services according to the observed policy changes. These results can then be fed into governance controllers in the form of a three-level hierarchy. This approach of evolving policies not only guides service development but also maintains separation from the implementation of the entire application. Figure 7-6 presents a holistic view of a policy-driven model for SOA governance.

In this characterization, action policies are the high-level business requirements that are usually manually enforced. Action policies can be enterprise or business requirements and goals. These policies are created at design time, and people usually adapt them manually. Goal policies are the actionable design specifications translated from the business goals. These policies have the scope of being enforced with some technological assistance such as collaboration tools, governance dashboards, or process automation tools. These tools mainly facilitate the orchestration of a service-oriented business and the evolution of its applications and infrastructure. Utility policies

are the executable behavioral specifications derived from design specifications and based on the action policies. They are usually enforced automatically using policy engines, service registry/repository rules, messaging standards, as well as service and network infrastructure. These policies are more auditable than the more general policies because "proof of compliance" can be demonstrated more easily through event logs or audit trails [2]. Finally, monitoring policies provide a foundation for continuous service improvement.

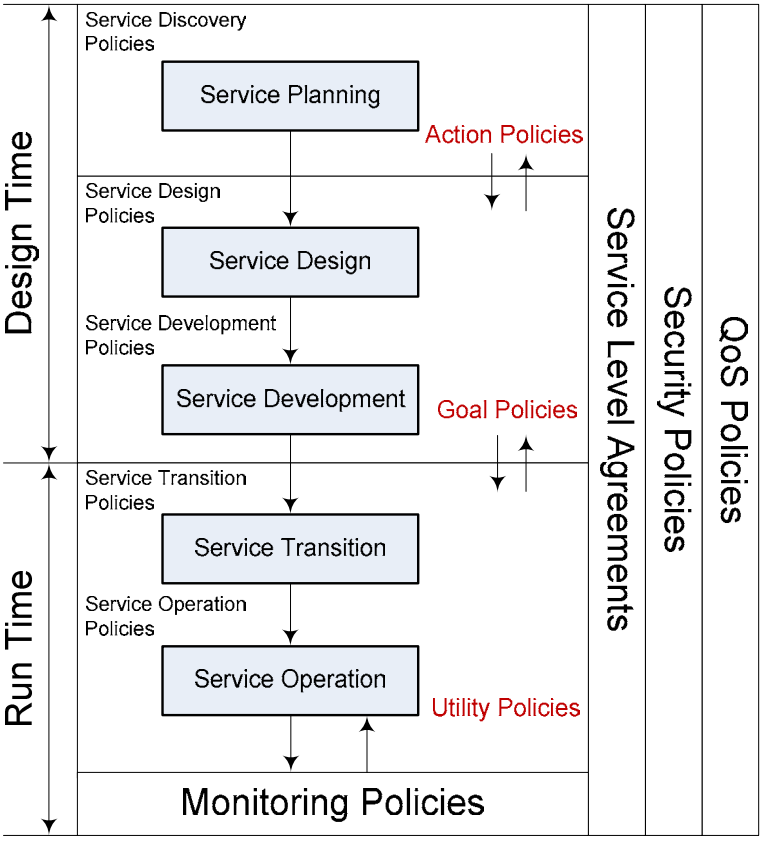


Figure 7-6: Service Category Dependencies

7.11 Survey Summary

This section summarizes our survey by presenting data from the selected references and the service life-cycle phases as depicted in Figure 7-7 to Figure 7-13. Figure 7-7 provides an overview of the overall distribution of papers by major service life-cycle policy type. Figure 7-8 to Figure 7-13 exhibit the distribution of papers for each major service life-cycle policy type by policy subtype. We collected 314 policy papers for this survey. Of these papers, 296 were categorized and used to produce Figure 7-7 to Figure 7-13.

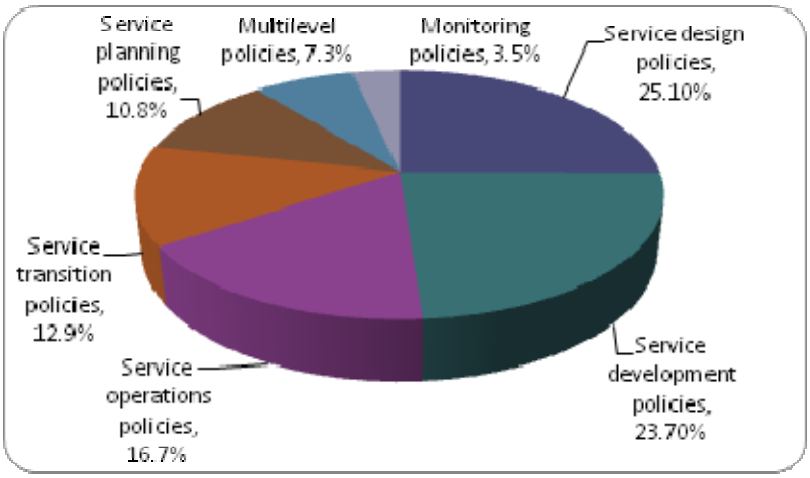


Figure 7-7: Overall Distribution of Contributions by Service Life-Cycle Policy Type

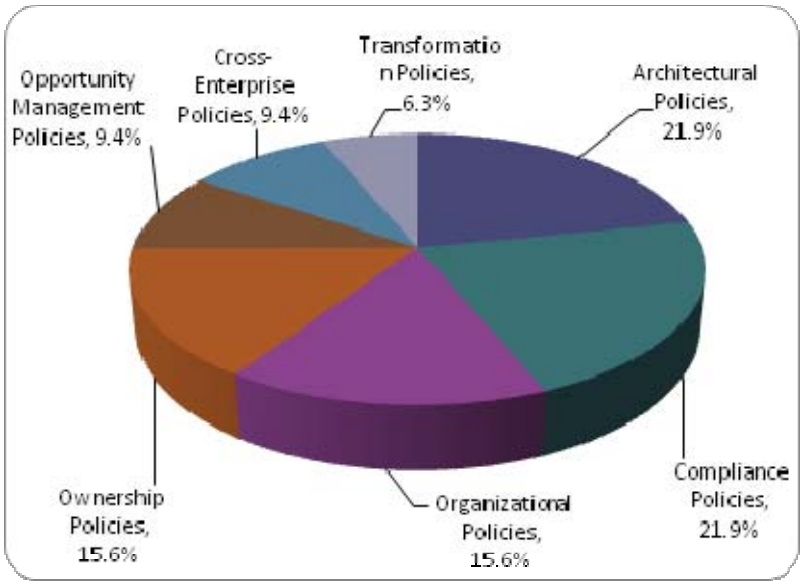


Figure 7-8: Distribution of Contributions of Service Planning Policies by Planning Subtype

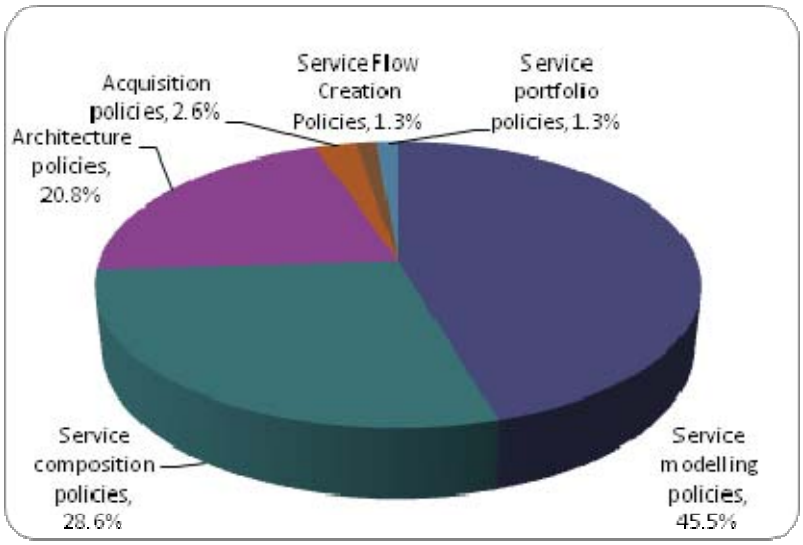


Figure 7-9: Distribution of Contributions of Service Design Policies by Service Design Subtype

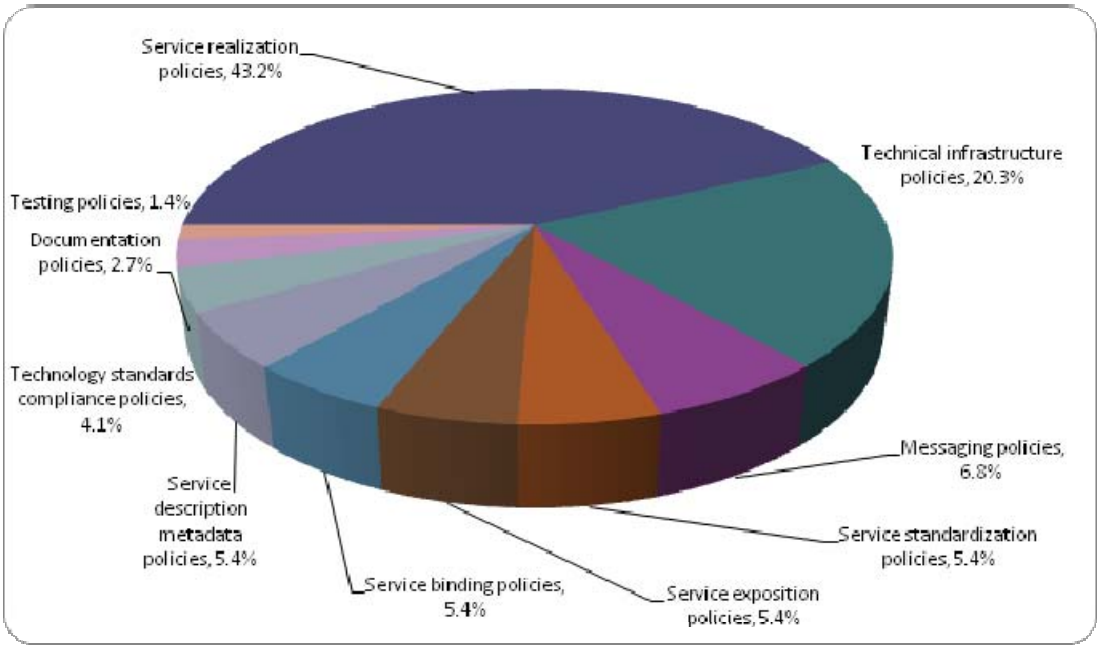


Figure 7-10: Distribution of Contributions of Service Development Policies by Development Subtype

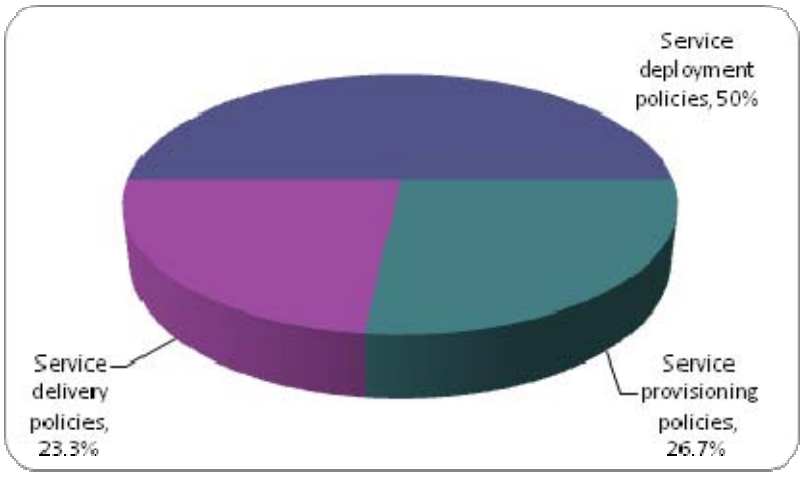


Figure 7-11: Distribution of Contributions of Service Transition Policies by Transition Subtype

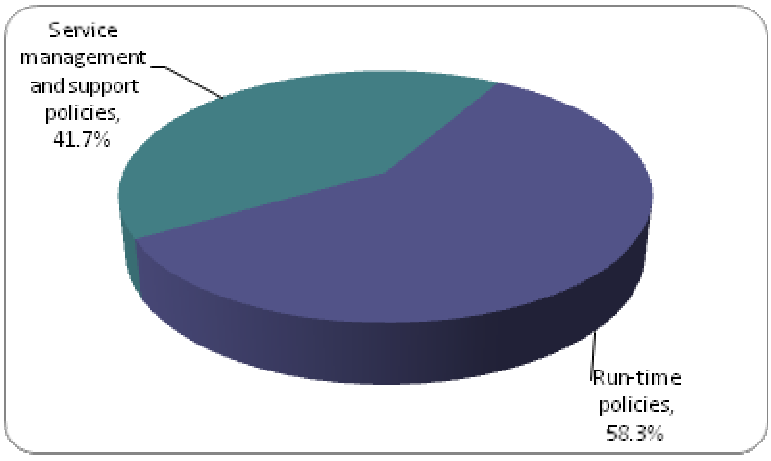


Figure 7-12: Distribution of Contributions of Service Operations Policies by Operations Subtype

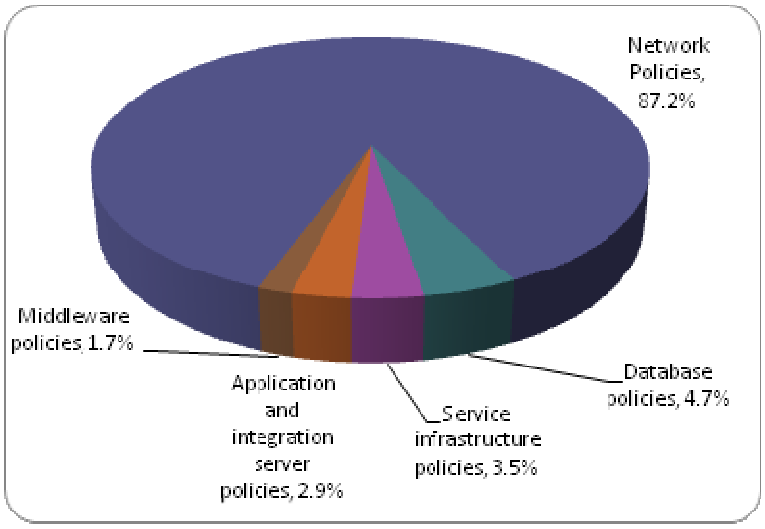


Figure 7-13: Distribution of Contributions of Domain-Specific Technical Policies by Domain-Specific Technical Policies Subtype

7.12 Conclusions

The goal of this survey was to characterize policies that govern service-oriented systems. SOA and SOA governance policies have had a tremendous impact in industry and have contributed significantly to the proliferation and success of service-oriented systems. While the contribution of governance as a whole is readily recognized, the contributions and relative merit of individual policies is not well documented and understood. It would be useful to assess the impact of individual deployed policies and identify the risks that can be remediated as a result. The classification of policies in this paper not only identified policies applicable to each phase of the service life cycle but also shed light on how policies can serve as feedback loops for managing, controlling, and adapting services in the context of changing business environments.

Acknowledgments

This work was funded in part by the National Sciences and Engineering Research Council (NSERC) of Canada, IBM Corporation, and CA Canada, Inc.

References

[1] M. Afshar, SOA Governance: Framework and Best Practices, Oracle White Paper, May 2007.

[2] E. A. Marks, Service-Oriented Architecture (SOA) Governance for the Services Driven Enterprise, Wiley, Sep. 2008.

[3] B. Woolf, Introduction to SOA Governance, IBM Developer Works, June 2006.

[4] M. Hondo, B. Portier, and F. Potepan, SOA Policy Management, IBM Redpaper, Sep. 2008.

[5] A. Erradi, Recovery policies for enhancing web services reliability. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2006), pp. 189–196, Sep. 2006.

[6] A. Erradi, P. Maheshwari, and V. Tosic, Policy-driven middleware for self-adaptation of web services compositions. In: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware, pp. 62–80, Nov. 2006.

[7] H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. Bauer, Issues in managing soft QoS requirements in distributed systems using a policy-based framework. In: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, LNCS 1995, pp. 185–201, Jan. 2001.

[8] S. Graham, G. Loewy, S. Murthy, and D. Potter, The cornerstones of SOA governance: Policies, registries and repositories, IBM White Paper, Oct. 2006.

[9] N. Damianou, A Policy Framework for Management of Distributed Systems, PhD thesis, Imperial College, London, 2002.

[10] M. Sloman, Policy driven management for distributed systems, Journal of Network and Systems Management 2(4):333–360, 1994.

[11] J. Kephart and W. Walsh, An artificial intelligence perspective on autonomic computing policies. In: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 3–12, June 2004.

[12] C. Shankar, A. Ranganathan, and R. Campbell, An ECA-P policy-based framework for managing ubiquitous computing environments. In: Proceedings of the Second Annual IEEE International Conference on Mobile and Ubiquitous Systems: Networking and Services, pp. 33–44, July 2005.

[13] J. Lamm, Under Control: Governance across the Enterprise, Apress Publications, Dec. 2009.

[14] T. Phan, J. Han, J. Schneider, T. Ebringer, and T. Rogers, Policy-based service registration and discovery. In: Proceedings of the 2007 OTM Confederated International Conference on the Move to Meaningful Internet Systems: COOPIS, DOA, ODBASE, GADA, and IS, Vol. I, pp. 417–426, Nov. 2007.

[15] Processdox, Configuration, Change and Release Management Policies and Procedures Guide, 2008. http://www.processdox.com/ConfigChangeReleaseMgmt.pdf

[16] M. Niemann, J. Eckert, N. Repp, and R. Steinmetz, Towards a generic governance model for service-oriented architectures. In: Proceedings of the Fourteenth Americas Conference on Information Systems (AMCIS 2008), Paper 361, 2008.

[17] S. Gorton, C. Montangero, S. Reiff-Marganiec, and L. Semini, StPowla: SOA, policies and workflows. In: Proceedings of the 3rd International Workshop on Engineering Service-Oriented Applications: Analysis, Design, and Composition, LNCS 4907, pp. 351–362, Jan. 2007.

[18] Policy Based Governance for the Enterprise, WebLayers. http://www.oasis-open.org/committees/download.php/15830/Introduction%20to%20Enterprise%20Policy%20Governance.doc

[19] ITIL v3, Service Portfolio, Service Catalog and Financial Management, http://advice.cio.com/reginaldlo/itil_v3_service_portfolio_service_catalog_and_financial_management

[20] IBM Corporation, An Architectural Blueprint for Autonomic Computing, IBM White Paper, 4th ed., 2006.

[21] R. Seiner, Data management policy: A cornerstone of data governance. Data Administration Newsletter, Jan. 2005. http://www.tdan.com/view-articles/5280/

[22] F. Chauvel, O. Barais, L. Borne, and J. Jezequel, Composition of qualitative adaptation policies. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), pp. 455–458, Sep. 2008.

[23] T. Gleason, K. Minder, and G. Pavlik, Policy management and web services. In: Proceedings of the Policy Management for the Web Workshop at IWWW Conference, May 2005.

[24] D. Weitzner, J. Hendler, T. Berners-Lee, and D. Connolly, Creating a policy-aware web: Discretionary, rule-based access for the World Wide Web. In: E. Ferrari, B. Thuraisingham (eds.), Web & Information Security. IDEA Group, pp. 1–31, 2006.

[25] T. Klie and L. Wolf, Autonomic policy-based management using web services. In: Proceedings ACM CoNEXT Conference (CoNEXT 2006) pp. 1–2, Dec. 2006.

[26] F. Li, F. Yang, S. Kai, and S. Su, A policy-driven distributed framework for monitoring quality of web services. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2008), pp. 708–715, 2008.

[27] O. Zimmerman, N. Schlimm, G. Waller, and M. Pestel, Analysis and design techniques for service-oriented development and integration, IBM, 2009. http://www.perspectivesonwebservices.de/download/INF05-ServiceModelingv11.pdf

[28] M. Papazoglou and W.-J. van den Heuvel, Service-oriented design and development methodology. International Journal Web Engineering Technologies (IJWET) 4:412–442, 2006.

[29] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, Service-oriented computing: A research roadmap. International Journal Cooperative Information Systems 17(2):223–255, 2008.

[30] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, S3: A service-oriented reference architecture, IEEE IT Professional 9(3):10–17, 2007.

[31] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, SOMA: A method for developing service-oriented solutions. IBM Systems Journal 47(3):377–396, 2008.

[32] L. Zhang, N. Zhou, Y. Chee, A. Jalaldeen, K. Ponnalagu, R. Sindhgatta, A. Arsanjani, and F. Bernardini, SOMA-ME: A platform for the model-driven design of SOA solutions. IBM Systems Journal 47(3):397–413, 2008.

[33] M. Salehie and L. Tahvildari, A policy-based decision making approach for orchestrating autonomic elements. In: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP 2005), pp. 173–181, Sep. 2005.

[34] C. Huang and N. Crespi, Extended policies for automatic service composition in IMS. In: Proceedings IEEE Asia-Pacific Services Computing Conference (APSCC 2009), pp. 254–259, Dec. 2009.

[35] B. Zhang, Y. Shi, and Y. Chen, A policy-based adaptation method for service composition. In: Proceedings 1st International Symposium on Pervasive Computing and Applications, pp. 619–624, Aug. 2006.

[36] A. Bertolino and A. Polini, SOA test governance: Enabling service integration testing across organization and technology borders. In: Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2009), pp. 277–286, April 2009.

[37] A. Sangroya, K. Garg, and V. Varma, SAGE: An approach to evaluate the impact of SOA governance policies. In: Proceedings of the IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA 2010), pp. 539–544, April 2010.

[38] T. Phan, J. Han, J.-G. Schneider, T. Ebringer, and T. Rogers, A survey of policy-based management approaches for service oriented systems. In: Proceedings of the 19th Australian Conference on Software Engineering (ASWEC 2008), pp. 392–401, Mar. 2008.

[39] F. Zulkernine, P. Martin, C. Craddock, and K. Wilson, A policy-based middleware for web services SLA negotiation. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2009), pp. 1043–1050, July 2009.

[40] N. Sheridan-Smith, J. Leaney, T. O'Neill, and M. Hunter, A policy-driven autonomous system for evolutive and adaptive management of complex services and networks. In: Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS 2005), pp. 389–397, April 2005.

[41] S. Illner, A. Pohl, H. Krumm, L. Luck, D. Manka, and F.-J. Stewing, Policy-based self-management of industrial service systems. In: Proceedings of the IEEE International Conference on Industrial Informatics (INDIN 2005), pp. 492–497, Aug. 2006.

[42] T. Schepers, M. Iacob, and P. Van Eck, A lifecycle approach to SOA governance. In: Proceedings of the ACM Symposium on Applied Computing (SAC 2008), pp. 1055–1061, 2008.

[43] H. Müller, P. Gupta, L. Nigul, R. Desmarais, A. Rudkovskiy, N. Villegas, and Q. Zhu, SOA governance optimizes the business and evolution of service-oriented systems. In: Proceedings of the 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2009), Edmonton, Canada, Sep. 2009.

[44] J. Prado Leite and M. Leonardi, Business rules as organizational policies. In: Proceedings of the Ninth International Workshop on Software Specification and Design (IWSSD '98), pp. 68–76, April 1998.

[45] SOA Governance and Service Lifecycle Management, http://www-01.ibm.com/software/solutions/soa/gov/policy/

[46] Service Usage Policy, http://www.discountasp.net/userpolicy_usage.aspx

[47] J. Amsden, Modeling SOA: Part 2. Service specification, http://www.ibm.com/developerworks/rational/library/07/1009_amsden/

[48] J. Amsden, Modeling SOA: Part 3. Service realization, http://www.ibm.com/developerworks/rational/library/07/1016_amsden/

[49] W. Zhen, L. Jianmin, Z. Nanrun, and L. Zhenrong, Semantic web service selection based on context and QoS. In: Proceedings of the International Conference on Web Information Systems and Mining (WISM 2009), pp. 332–335, Nov. 2009.

[50] Q. Lv, J. Zhou, and Q. Cao, Service matching mechanisms in pervasive computing environments. In: Proceedings of the International Workshop on Intelligent Systems and Applications (ISA 2009), pp. 1–4, May 2009.

[51] X. Zhou, W. T. Tsai, X. Wei, Y. Chen, and B. Xiao, Pi4SOA: A policy infrastructure for verification and control of service collaboration. In: IEEE International Conference on e-Business Engineering (ICEBE 2006), pp. 307–314, Oct. 2006.

[52] Tivoli Identity Manager, Provisioning Policies, http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itim.doc/cpt/cpt_ic_admin_provisionpolicy.html

[53] IBM WebSphere Service Registry and Repository, http://publib.boulder.ibm.com/infocenter/sr/v7r0/index.jsp?topic=/com.ibm.sr.doc/cwsr_overview_features25.html

[54] Web Services Addressing 1.0: SOAP Binding, http://www.w3.org/TR/ws-addr-soap/

[55] A. Abbas, M. El-Hilaly, and H. Harraz, Exposing RESTful Services Using an Enterprise Service Bus, http://www.ibm.com/developerworks/webservices/library/ws-RESTesb/index.html?ca=drs-

[56] S. Chatterjee, Messaging Patterns in Service-Oriented Architecture, Part 1, http://msdn.microsoft.com/en-us/library/aa480027.aspx

[57] How to Write Doc Comments for the Javadoc Tool, http://java.sun.com/j2se/javadoc/writingdoccomments/index.html#styleguide

8 Context-Driven Adaptive Monitoring for Supporting SOA Governance

Norha M. Villegas, University of Victoria, Canada, and Icesi University, Colombia (nvillega@cs.uvic.ca)
Hausi A. Müller, University of Victoria, Canada (hausi@cs.uvic.ca)

Abstract

The distributed nature of services and the continuous evolution of businesses make service-oriented architecture (SOA) environments highly dynamic, uncertain, and context dependent. Consequently, SOA governance requires effective monitoring mechanisms to manage the deployment, execution, maintenance, and evolution of service-oriented systems. Within these runtime dynamics, context awareness and control loops provide effective mechanisms for the automation of runtime and change-time governance. In this paper, we propose a model-based approach to represent context and its monitoring requirements for SOA governance and a reference architecture in which feedback loops are exploited as first-level components to control the adaptation of monitoring infrastructures and enable them to (1) monitor context information relevant for regulating the accomplishment of SOA-governance objectives and (2) reconfigure themselves to support new monitoring strategies according to environmental changes that affect SOA-governance objectives. As an application case, we propose a concrete architecture for implementing a monitoring infrastructure that supports the negotiation of service-level agreements at runtime.

8.1 Introduction

Service-oriented systems are highly affected by distributed, heterogeneous, transient, and volatile contexts. From the perspective of service-oriented architecture (SOA) environments, *context* can be defined as any information that characterizes the state of entities that affect the execution, maintenance, and evolution of service-oriented systems. This context information must be modeled in such a way that it can be discovered, preprocessed after its acquisition from the environment, classified according to the corresponding domain, provisioned based on the system's requirements, and maintained to support its dynamic evolution [1, 2]. Instances of context entities in SOA environments are services, businesses, service-level agreements (SLAs), users, and computational resources.

SOA governance defines the policies and enforcement mechanisms for implementing, executing, and evolving service-oriented systems. These policies and enforcement mechanisms include procedures, roles, and responsibilities for both design-time and runtime governance [3]. According to Lewis and Smith, procedures of SOA governance include various approaches to concerns, such as access to application and data repositories, replacement of services, control of the SOA-governance infrastructure, and management of SLAs [4]. To ensure governance procedures, SOA governance requires software and computing infrastructures to monitor the behavior of service-oriented systems and to apply enforcement mechanisms according to policies, procedures, and responsibilities. Thus, we define a *SOA-governance infrastructure* as the set of

software components and the interactions among them, as well as the processing nodes and data repositories that are required to manage the pertinence of service-oriented systems with respect to business processes, under the heterogeneous, uncertain, and dynamic conditions inherent to SOA environments. In addition, this dynamic affects policies, processes, business-level objectives, SLAs, and governance strategies at design time, runtime, and change time. Therefore, SOA governance urgently demands innovative monitoring strategies to support the dynamic adaptation of SOA-governance objectives (e.g., through runtime renegotiation of SLAs) and services according to changes in the environment [5, 6]. Most importantly, as businesses are highly dynamic, monitoring infrastructures for supporting SOA governance also need to adapt in order to manage monitoring requirements, as the environment and the SOA-governance objectives are continuously evolving.

Context monitoring for SOA environments requires research in context awareness and software engineering of self-adaptive systems. From the perspective of context awareness, monitoring infrastructures require effective and computationally efficient mechanisms for managing dynamic context [2].

For acquisition, representation, handling, and provisioning of context information for supporting SOA governance, research challenges include

- defining mechanisms to identify and represent context facts that affect the governance of service-oriented systems (e.g., automated inference of context information relevant for a specific set of dependent SLAs, machine-readable representations of context models)
- dynamic discovery of context sources according to context models
- runtime modification of context models
- context-handling strategies according to changes in SOA-governance objectives

With respect to software engineering of self-adaptive systems, context monitoring needs reference models, reference architectures, methods, and techniques for designing and implementing infrastructures that are able to

- support the dynamic modification of context models
- self-adapt to fulfill new monitoring requirements, according to environmental changes that affect SOA-governance objectives
- support the dynamic evolution of SOA-governance objectives (e.g., the runtime renegotiation of SLAs)

Valuable research projects provide evidence of the urgent necessity of advancements in monitoring mechanisms for SOA governance. Among them, the web service-level agreements (WSLA) framework, which was proposed by researchers from IBM, monitors SLAs for web services based on the WSLA language specification [7, 8]. However, although this framework provides an extensible runtime architecture composed of several SLA monitoring services, this approach lacks mechanisms for supporting dynamic context-based monitoring. Thus, information to be monitored and the corresponding monitoring strategies are statically defined at design time when SLAs are negotiated. Other approaches, such as the measurement service proposed by Schmietendorf et al., provide useful mechanisms to measure the performance, stability, and

availability of web services [9]. However, context information is not dynamically integrated into the measurement strategies. To deal with the runtime adaptation of SOA-governance objectives, Herssens et al. have proposed an SLA manager that supports the dynamic negotiation of SLAs as required by changes in the environment [10]. Although this approach involves the context that can affect the governance of an SLA, similar to other identified approaches, context modeling and monitoring strategies are not adaptable to guarantee the pertinence of context management with respect to current situations. In summary, none of the cited approaches take into account the complexity of monitoring context for a set of composed services.

To advance runtime monitoring for supporting SOA governance, we advocate the exploitation of feedback loops as first-level software components. Feedback loops provide generic mechanisms for dynamic adaptation [11]. Thus, mapping monitors, analyzers, planners, and actuators as interoperable and distributed components (e.g., services) contribute to the engineering of adaptive monitors by separating the concerns of system functionality from the concerns of context management and adaptation of monitoring infrastructures. In the same way, the decoupling of feedback loop elements enables SOA-governance infrastructures to also be self-adaptive. In addition, the explicit separation of these concerns is required to support the distributed nature of context information in service-oriented systems as well as to outsource monitoring services to third parties to maximize the objectivity.

We propose a service component reference architecture in which *governance feedback loops* co-operate with each other to monitor the dynamic environment of service-oriented systems and to support the adaptation of the monitoring infrastructure accordingly. Our proposed reference architecture is applicable to the design and implementation of concrete self-adaptive monitoring architectures, in which monitors are designed to be deployed at runtime and planners dynamically define the strategy for handling context information, according to the monitoring objectives. To achieve the desired levels of dynamism, we also propose a model-based approach for representing context entities and context-monitoring requirements, based on the application of context modeling and context-management features, such as that proposed by Villegas and Müller [2]. These context representations are required to evolve and to be processable at runtime to maintain the pertinence between SOA-governance objectives and the context information to be monitored.

Finally, with our proposed reference architecture, we aim to assist SOA practitioners in the engineering of distributed monitoring infrastructures for SOA governance that are able to (1) monitor context information relevant for regulating the accomplishment of SOA-governance objectives and (2) reconfigure themselves to support new monitoring strategies according to environmental changes that affect SOA-governance objectives. The next sections are organized as follows. Section 8.2 illustrates identified challenges of context monitoring to support the dynamic negotiation of SLAs through a SOA-governance scenario; Section 8.3 presents our feature-based approach for context modeling and a general explanation of the application of feedback loops to context monitoring for the automation of SOA governance. Section 8.4 explains the elements of our proposed monitoring reference architecture. Section 8.5 presents a concrete software architecture for implementing a monitoring infrastructure as required by the SOA-governance scenario described in Section 8.2. Section 8.6 discusses related approaches, many of them useful for the implementation of infrastructures based on our proposed reference architecture, and presents future perspectives. Finally, Section 8.7 concludes the paper.

8.2 Application Case: Toward Dynamic Negotiation of SLAS

The architectures presented in this paper are service component architectures (SCA) based on the assembly model specification for SCA version 1.0 [12]. Figure 8-1 shows the legend for the main artifacts of SCA.

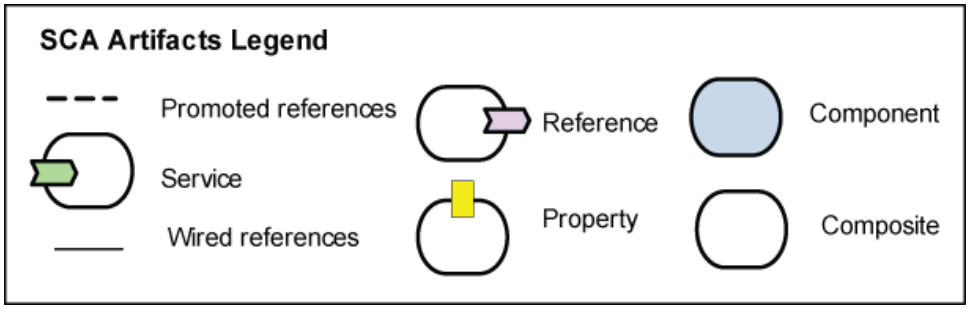


Figure 8-1: Artifacts of the SCA Assembly Specification

According to the SCA specification, *components* are the basic artifacts that implement the program code for providing business functions. *Services* are the interfaces that expose these business functions in order to be used by other components. *References* enable components to consume services. *Properties* are settable values that influence the operation of business functions. *Composites* provide a logical grouping to deploy components. *Wires* interconnect components within the same composite. *Promoted references* are references that must be resolved by services outside the composite. Composites are deployed within an *SCA domain* that generally represents a set of services that provide functionalities related to a same business area controlled by a single organization. Although the *deployable* annotation is not defined by the SCA specification, we use it to indicate that components are dynamically deployed at runtime.

Management of SLAs is an important concern of runtime and change-time SOA governance [4, 10]. Thus, to illustrate the application of our proposed control-based reference architecture, we introduce the following SOA-governance application case in which SLAs are managed and renegotiated at runtime. Imagine a hotel that offers a comprehensive catalog of context-aware services related to guest's interests such as indoor and outdoor facilities, dining and meeting schedules, and tourist and shopping information. As presented in Figure 8-2, different service brokers are available for providing the hotel's system with relevant services related to nearby facilities according to the guest's interests. SLAs are defined between the hotel and service-facilities brokers to guarantee service-level objectives (SLOs) such as a minimum transaction rate.

Service brokers provide context-aware access to nearby facilities services according to users' concerns. SLAs between Shopping Broker A and each store affect the accomplishment of the initial SLA.

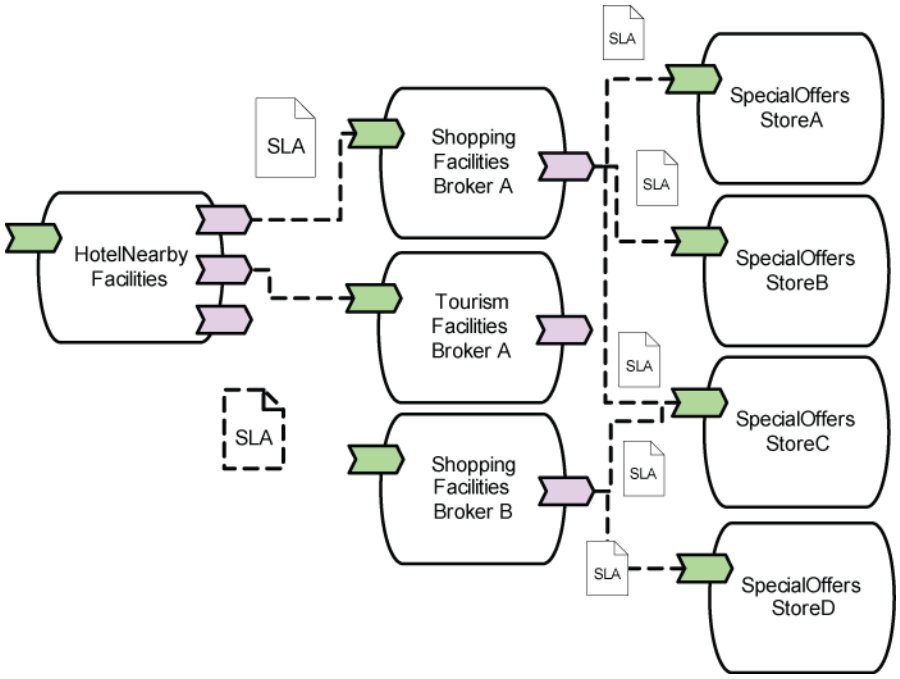


Figure 8-2: Application Case Scenario

The initial SLA between the hotel and Shopping Broker A involves the *HotelNearbyFacilities* and *ShoppingFacilitiesBrokerA* services. In order to provide the *HotelNearbyFacilities* services with information about special offers from nearby boutiques according to the user's preferences, *ShoppingFacilitiesBrokerA* composes services from different providers. These providers are stores that correspond to the user's shopping interests. Thus, an SLA is defined between each shopping broker and each boutique to enable shopping brokers to provide the desired functionality under the negotiated conditions.

For the SLA that exists between the hotel and Shopping Broker A, a throughput SLO obligates the *ShoppingFacilitiesBrokerA* service to guarantee a minimum transaction rate of 10 transactions per second for the months between 2010-06-01 and 2010-08-31 every day (summer season). For the remainder of the year, the minimum transaction rate is 5 transactions per second every day. We assume that two metrics were defined for the verification of the accomplishment of the throughput SLO. The first metric, *number of transactions*, is measured by counting the number of successful operations. The second metric, *time spent*, is measured by counting the time spent for each transaction from the instant the consumer performs the request until the answer is received by the consumer. However, whenever the initial SLA is violated, the SOA-governance infrastructure should support the definition of a new SLA between the hotel and another shopping broker (e.g., a new SLA to consume the *ShoppingFacilitiesBrokerB* service) able to fulfill the SLOs.

Context-Aware Dynamic Negotiation of SLAs. The throughput SLO defined in the example is clearly dependent on context information. Metrics such as *number of transactions* and *time spent* correspond to activity- and time-context categories according to the taxonomy presented by Villegas and Müller [2]. Date and time also provide important contextual facts because the thresholds for the transaction rate have a seasonal variation. Therefore, context information that can affect the accomplishment of SLOs goes beyond the context facts directly related to the metrics stated in the SLAs. For instance, weather variations can affect the transaction rate if

summer begins before the date-time interval that is defined by the SLO. In the same way, special events at the hotel (e.g., conferences, special discounts) can dramatically impact the *HotelNearbyFacilities* service load as a consequence of an increase in the number of registered guests. In addition, artificial context, such as the computational environment (e.g., connectivity, bandwidth), can also impact the accomplishment of the SLO.

A more complex scenario arises from the composition of multiple services and the dependencies among the corresponding SLAs. Thus, SLAs between the *ShoppingFacilitiesBrokerA* service and each *SpecialOffersStore* service are also part of the context that can affect the accomplishment of the monitored SLA. Although the negotiated transaction rate is affected by context and depends on the accomplishment of related services' SLAs, current SLA specifications do not include the definition of these context-aware parameters at negotiation time. Under these dynamic conditions, by monitoring only the *number of transactions* and *time spent* metrics, the current SLA will no longer be able to fulfill the business-level objectives. On one hand, the provider will not be able to guarantee the transaction rate under these changing conditions. On the other hand, the system will operate with no guarantee of quality of service (QoS) until the parties manually identify the situation and manually perform a new negotiation. In the same way, the SLO is not taking dynamic conditions into account to demand a more appropriate transaction rate.

Monitoring SOA infrastructures for supporting SOA governance (e.g., the dynamic negotiation of SLAs) assumes many open challenges. First, the specification of SOA-governance objectives must include the specification of relevant context and monitoring objectives (e.g., for SLAs, relevant context can be specified in the form of SLA parameters and corresponding metrics). Second, some aspects, such as context sources, are not always well known in advance. Third, users are not able to know the complete set of context facts that can affect the accomplishment of SOA-governance objectives. Finally, context models must include relationships among context entities to infer relevant context facts (e.g., accomplishment dependencies among SLAs to infer what we need to monitor from third parties).

Based on this application case, we will illustrate how feedback loops and context-aware monitoring mechanisms are useful for supporting runtime and change-time governance from an architectural point of view. In particular, we illustrate how they can support the dynamic renegotiation of SLOs and the reconfiguration of monitoring infrastructures accordingly, as well as the dynamic negotiation of SLAs when a service able to better fulfill the business-level objectives is discovered at runtime.

8.3 Context-Aware Governance Feedback Loops

SOA governance ensures that the concepts and principles of service-oriented infrastructures are appropriately managed for regulating the accomplishment of business objectives. Thus, the goals of SOA governance at runtime and change-time are to control, monitor, and adapt the components of these infrastructures to optimize business and evolution [6]. To regulate the accomplishment of business goals, SOA governance requires instrumentation to keep track of the SOA infrastructure and environmental changes that affect SOA-governance objectives at both runtime and change time. Monitoring is a crucial component of runtime governance for ensuring that services are executed according to policies and responsibilities. Also, for change-time governance, monitoring

tracks facts in the environment to identify current and future disruptions and, with this, the need to evolve services or SOA-governance objectives.

Researchers and practitioners from the maintenance and evolution of service-oriented systems (MESOA) community have pointed out the importance of feedback loops to support the automation and enforcement of change-time governance. Three main challenges have been identified [13]:

- models for incorporating feedback loops into service-oriented systems
- mechanisms for managing and leveraging uncertainty
- mechanisms for making control loops explicit in service-oriented systems

In addition, according to researchers from the self-adaptive and self-managing systems (SEAMS) community, adaptation mechanisms of dynamic systems can be exploited by analyzing, modeling, and implementing feedback loops as first-class software components [11, 14]. Thus, improving the visibility of feedback loops is crucial for addressing the design, implementation, maintenance, and evolution of service-oriented systems under the dynamics of SOA environments.

Explicit feedback loops built into runtime and change-time governance, as well as features of context modeling and management techniques, constitute the core of our proposal intended for the optimization of context monitoring in SOA governance [2, 6]. As depicted in Figure 8-3, context monitors gather context information from the internal and external environments to identify relevant context facts according to SOA-governance objectives. Based on symptoms identified by the context monitor, SOA-governance controllers control governance objectives by adjusting policies or renegotiating responsibilities and agreements. In the same way, a context monitor controller manages the adaptation of the monitoring infrastructure, according to context observations. These observations include contextual facts that characterize the situation of services, business-level objectives, people, SLAs, and the SOA-governance infrastructure itself.

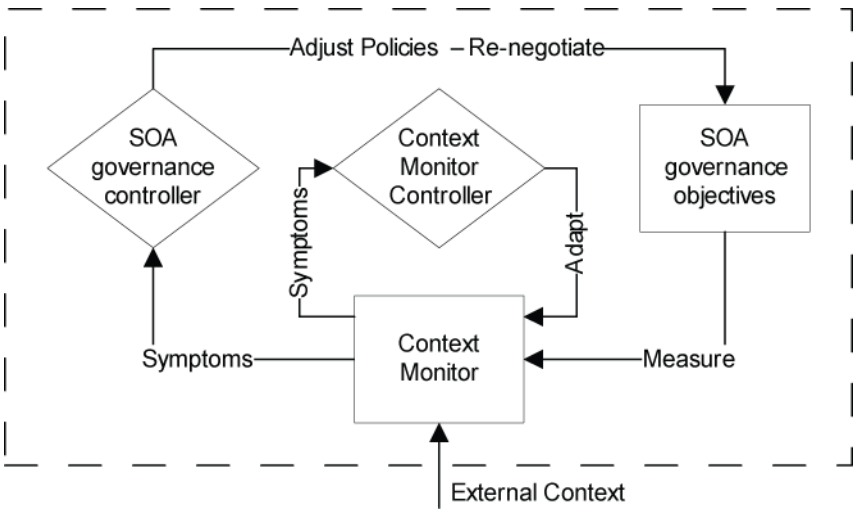


Figure 8-3: Context-Aware Governance Feedback Loops

In this section, we propose *context-aware governance feedback loops* that are explicitly defined as design drivers for the implementation of context-driven adaptive monitoring infrastructures that support SOA governance at runtime and change time. We also propose a model-based approach to

represent context—and its monitoring requirements—that supports the comprehensive set of context-awareness features proposed by Villegas and Müller in their framework for evaluating and implementing context modeling and context-management mechanisms [2].

8.3.1 A Context Meta-Model for Monitoring SOA Governance

Monitoring infrastructures for SOA governance require appropriate context-modeling mechanisms to represent both the relevant aspects of entities that can affect the accomplishment of SOA-governance objectives and context-monitoring objectives. We use the term *context-monitoring objectives* to refer to the aspects of the involved entities that must be monitored and their corresponding metrics. In this endeavor, the main modeling challenge is the identification of relevant context, for instance, from policies and agreements. The management of SLAs is one of the main activities of SOA governance [4]. However, current approaches for specifying machine-readable SLAs lack an explicit definition of contextual entities and context-monitoring objectives beyond the particular metrics for deciding on the accomplishment of SLOs under static conditions [3, 8]. If we intend to dynamically monitor context information for supporting SOA governance, context information that can affect the accomplishment of SOA-governance objectives should be specified at negotiation time, in the same way as SLA parameters, metrics, and SLOs.

Most importantly, the representation of context must include not only context entities and monitoring metrics but also the relationships among the involved entities as well as context-management aspects, such as available context sources and provisioning and acquisition mechanisms. Thus, user-centric context-modeling tools are required for supporting context specification as part of SOA-governance objectives, either at negotiation time or whenever is necessary.

Figure 8-4 presents the context meta-model that we propose to instantiate context models that can be integrated into monitoring infrastructures based on our reference architecture. An appropriate representation of context enables the monitoring of context information that is relevant for the accomplishment of SOA-governance objectives, even when they are renegotiated dynamically (e.g., by changing SLOs for an SLA), as long as the model is modified accordingly. For instance, for managing SLAs, the *MonitoringCondition* meta-class enables the instantiation of classes from SLOs; the *MonitoringCondition* meta-class together with *Threshold* and *Observation* provides a way of instantiating classes for representing SLA metrics; and the *PostCondition* meta-class supports the instantiation of entities that represent action guarantees and adaptation requirements of either the governance infrastructure or the involved services. The remaining meta-classes are required by the monitoring infrastructure to manage context. Our reference architecture does not depend on this specific meta-model. Other modeling approaches can be integrated into concrete architectures.

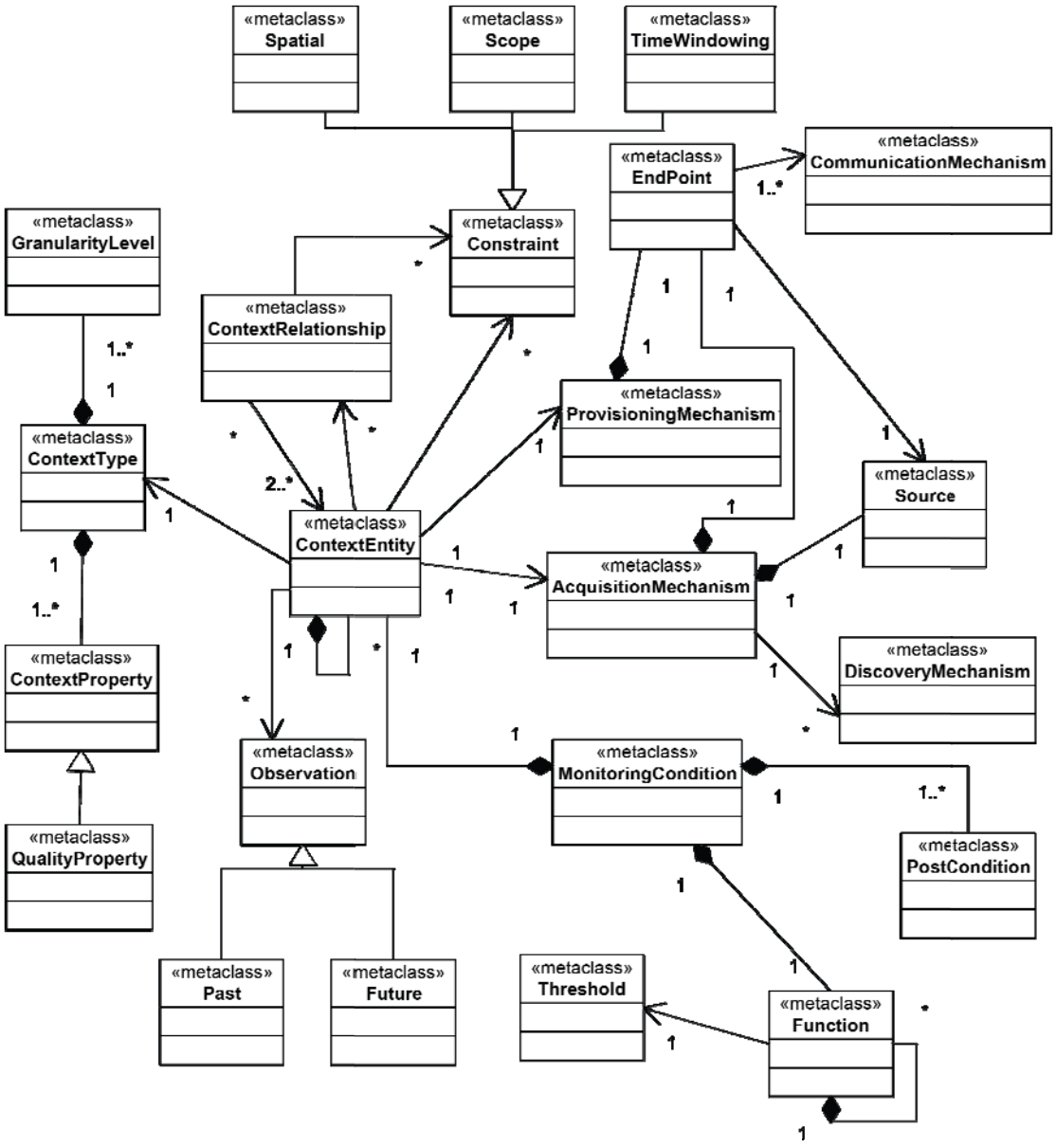


Figure 8-4: A Meta-Model for Representing Context Entities and Context-Monitoring Objectives for Supporting SOA Governance

8.3.2 A Reference Model Based on Governance Feedback Loops

To guide the definition of a reference architecture that assists in the design of dynamic monitoring infrastructures for supporting runtime and change-time governance, we introduce the reference model presented in Figure 8-5. This reference model proposes feedback loops as first-class elements to be mapped into software architectures to implement dynamic monitoring infrastructures that support changes in monitoring requirements.

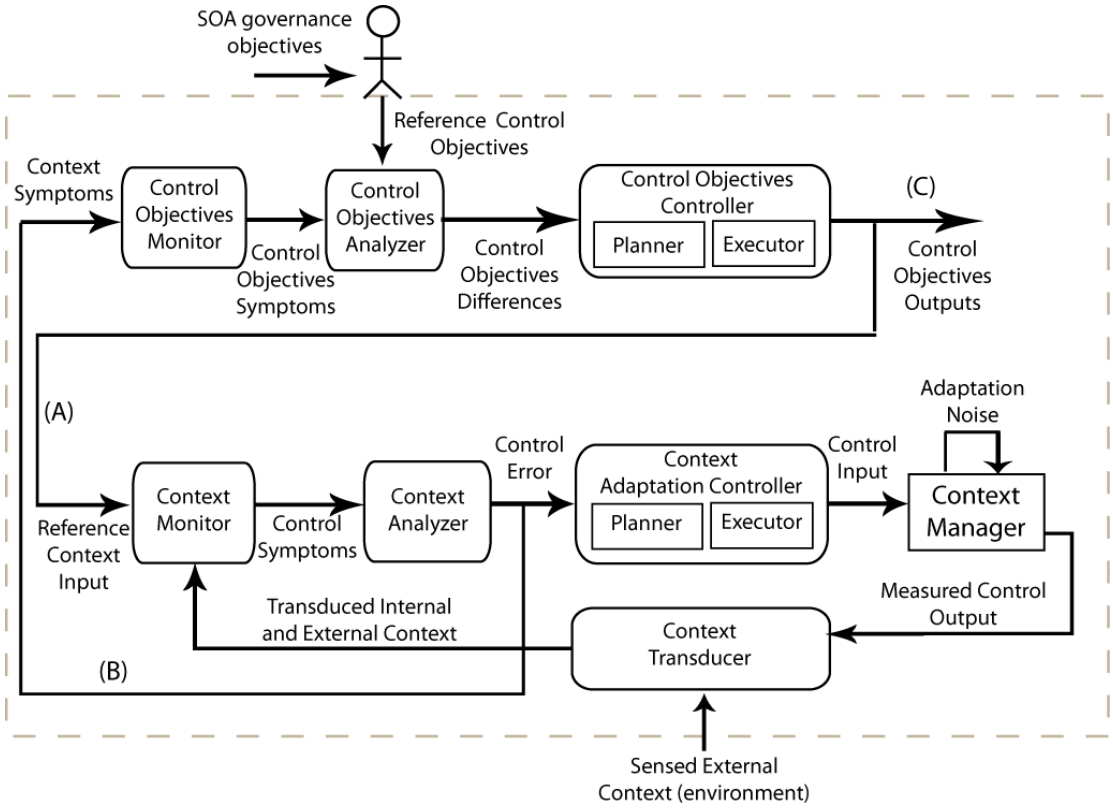


Figure 8-5: Control-Based Reference Model for Monitoring Dynamic SOA Environments

As a result, these infrastructures are able to continuously enhance and adjust both themselves and the monitoring strategy, taking context observations that affect the redefinition and accomplishment of SOA-governance objectives into account. The accomplishment of governance objectives is regulated by the feedback loop depicted in the upper part of Figure 8-5.

This *control-objectives feedback loop* is in turn supported by the *context manager feedback loop* depicted in the lower part of the same figure. The latter is in charge of controlling the monitoring of context information that can affect the fulfillment of SOA-governance objectives and the behavior of the monitoring infrastructure itself. Labels (A), (B), and (C) represent important interactions among architectural components derived from this monitoring infrastructure based on our reference architecture and are independent of specific meta-models. However, feature-based frameworks, as proposed by Villegas and Müller, provide a comprehensive tool to evaluate context models that can be integrated into our reference architecture [2].

Interaction (A) provides the reference context input (i.e., context-monitoring objectives derived from SOA-governance objectives) for the context manager loop to gauge the relevance of context information, decide how to manage and provision that environmental information, and provide the context symptoms for the control-objectives feedback loop to regulate the accomplishment of SOA-governance objectives.

Interaction (B) enables the control-objectives loop to support decision making about changes in SOA-governance strategies and even service-oriented systems, whenever the control-objectives

controller detects that, given the current context, the actual set of SOA-governance objectives should be dynamically adjusted or renegotiated.

Label (C) refers to an external interaction that notifies involved services about conditions that can trigger their adaptation. For the automation of SOA governance, control objectives (i.e., SOA-governance objectives), relevant context, and monitoring requirements must be specified in the form of machine-readable models [2, 3]. Thus, a context-management infrastructure (i.e., context manager control loop) must be able to process these models that represent the relevant context facts, as well as the context handling and provisioning features that must be managed for monitoring the environment. Finally, the context adaptation controller controls the adaptation of the context manager to support new context-management requirements, according to changes in SOA-governance objectives.

8.4 The Control-Based Reference Architecture

Figure 8-6 presents our proposed reference architecture. Two SCA domains are required to monitor context in SOA environments. Both domains define composites based on the elements of feedback loops.

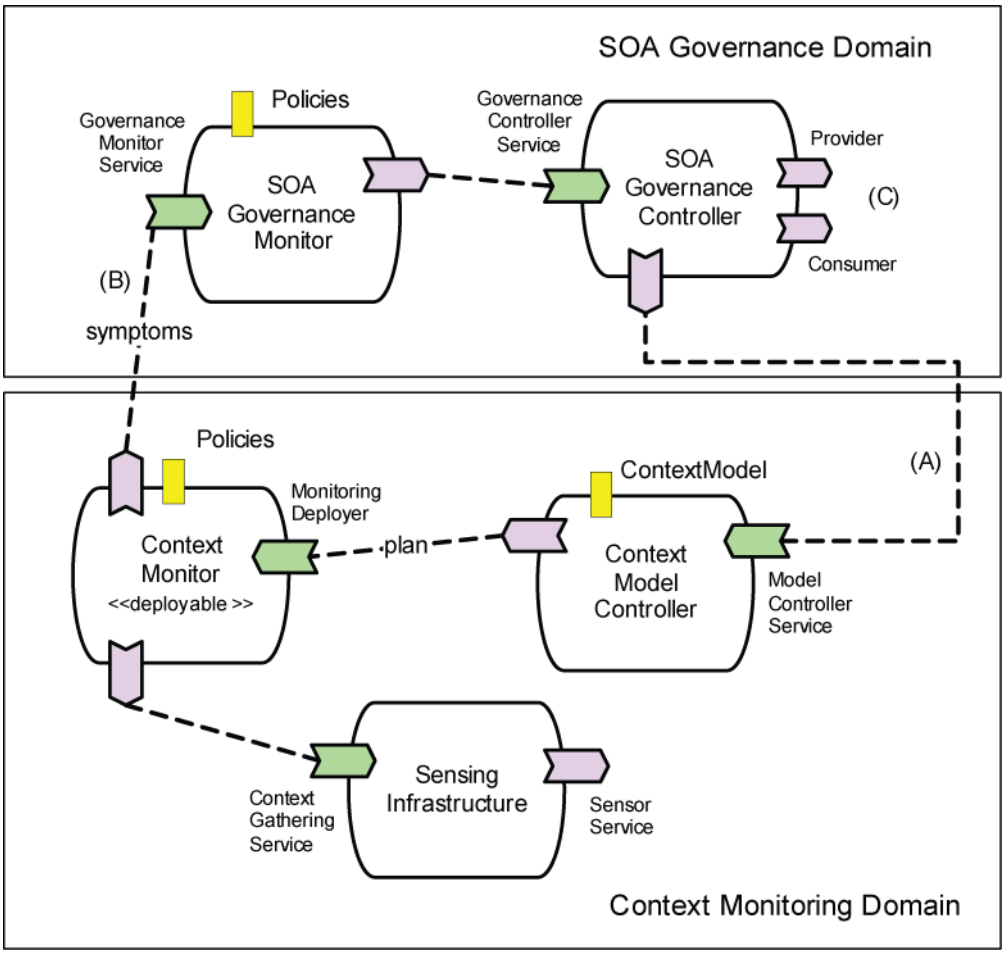


Figure 8-6: Control-Based Service Component Reference Architecture for Monitoring SOA Governance

The first domain, *SOA Governance*, defines the composites in charge of controlling the SOA infrastructure for regulating the accomplishment and evolution of SOA-governance objectives. The second one, *Context Monitoring*, defines the core composites in charge of gathering, monitoring, and providing the relevant context information that can affect the accomplishment of SOA-governance objectives. As defined by our reference model described in Section 8.3.2, two main interactions must be implemented between these two domains. Interaction (A) takes place when a *SOAGovernanceController* requests monitoring services from a *ContextModelController*. Then, a *ModelControllerService* initiates components within the corresponding *ContextModelController* to infer both relevant context and monitoring strategies based on the *ContextModel* property. The *ContextModel* property corresponds to the machine-readable representation of context information and monitoring requirements that can be instantiated, for instance, from meta-models such as the one we proposed in Section 8.3.1. Subsequently, interaction (B) takes place when a *ContextMonitor* provides the *SOAGovernanceMonitor* context symptoms, based on the context-monitoring requirements inferred from the context model. Analyses based on these symptoms are used by the *SOAGovernanceController* to support decision making regarding the accomplishment of SOA-governance objectives and whether service-oriented systems should adapt. *Provider* and *Consumer* references enable the *SOAGovernanceController* to notify service-oriented systems about conditions that can trigger their adaptation. This capability corresponds to interaction (C) defined in our reference model in Figure 8-5.

Within the context-monitoring domain, once the *ContextModelController* defines the monitoring plan, the *ContextMonitor* is in charge of deploying and performing the monitoring strategy according to that plan. The *ContextGathering* service exposes the *SensingInfrastructure* to enable the acquisition of context information from the environment.

The SOA-Governance Domain Reference Architecture. Figure 8-7 presents the core components of the SOA-governance domain architecture. Figure 8-7(a) details the *SOAGovernanceMonitor* presented in Figure 8-6. The *GovernanceMonitor* service enables the *SOAGovernanceMonitor* to identify, based on policies, governance symptoms from the context symptoms received from the context-monitoring domain. The *SOAGovernanceAnalyzer* uses these symptoms to make decisions about the need for requesting a new governance plan.

Figure 8-7(b) details the *SOAGovernanceController*. Once the *SOAGovernancePlanner* is requested by the *SOAGovernanceAnalyzer*, the former uses policies to define a plan depending on whether it is required to adapt the context model and consequently the monitoring infrastructure, or to report the need for adaptation of services from consumers, providers, and third parties.

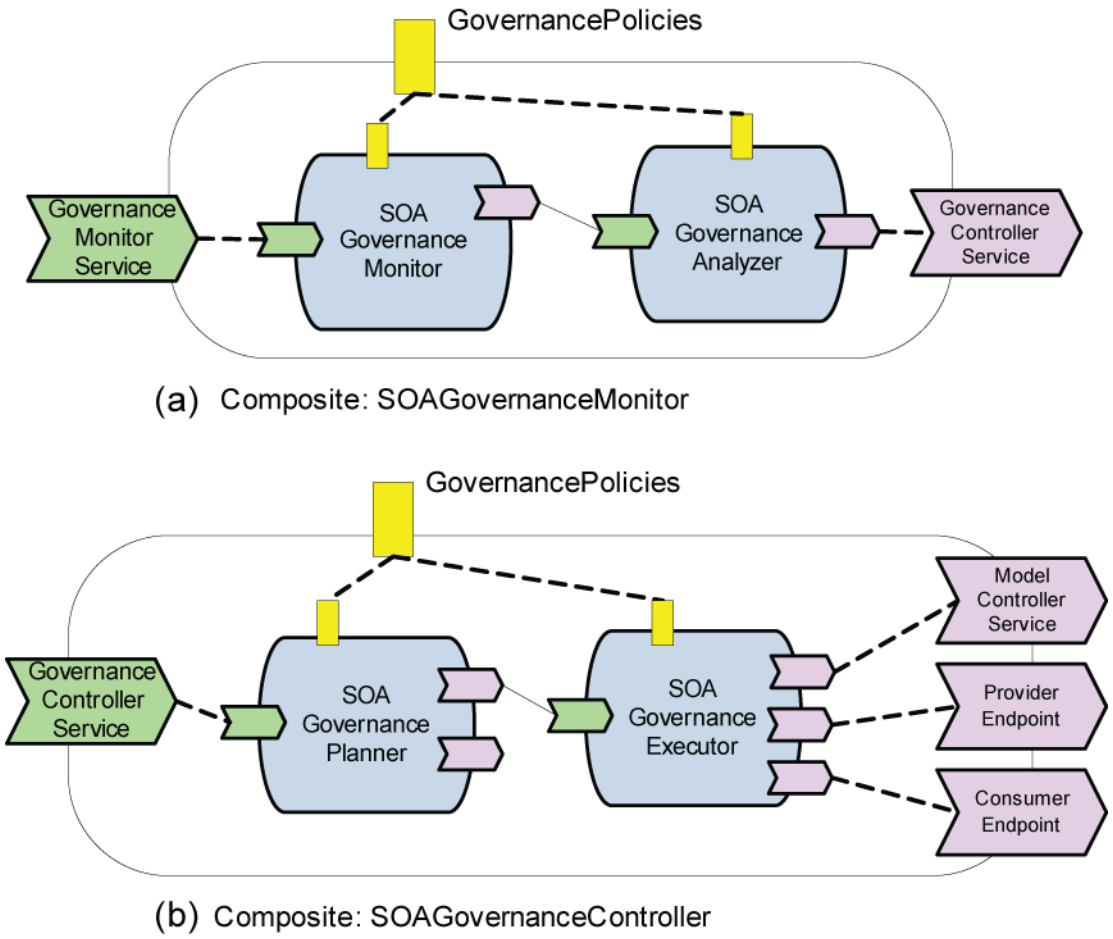


Figure 8-7: SOA-Governance Architecture Domain

The Context-Monitoring Domain Reference Architecture. Figure 8-8 and Figure 8-9 represent the core of our monitoring architecture. Figure 8-8 depicts the *ContextControlObjectivesInterpreter* and the *MonitoringPlanner* that compose the *ContextModelController*.

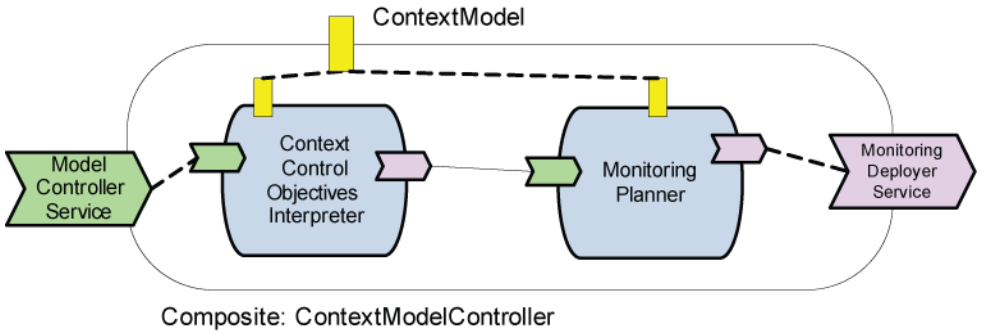


Figure 8-8: Context Model Controller Architecture

The *ContextControlObjectivesInterpreter* is in charge of the runtime processing of the context model in order to infer the context control objectives. Context control objectives provide the monitoring infrastructure with the relevant context information, observations to be monitored,

metrics to be used by the monitoring strategy, and gathering and provisioning mechanisms. Subsequently, the *ContextControlObjectivesInterpreter* provides the *MonitoringPlanner* with the required information to define the monitoring plan.

The monitoring plan defines (1) the number and types of monitors to be deployed with the corresponding monitoring algorithms; (2) the number of context acquisitors according to the context types to be gathered; and (3) the number and types of context handlers with the corresponding context handling and composition algorithms. Afterward, the *MonitoringPlanner* invokes the *ContextMonitorsDeployer* to trigger the deployment process at runtime, as shown in Figure 8-9. As soon as the deployment is completed, each *ContextAcquisitor* starts to gather individual context entities by using the *ContextGathering* service provided by the sensing infrastructure. According to the gathering mechanism (i.e., pulling or pushing), each *ContextHandler* runs and starts to interact with each other, as defined by the monitoring plan.

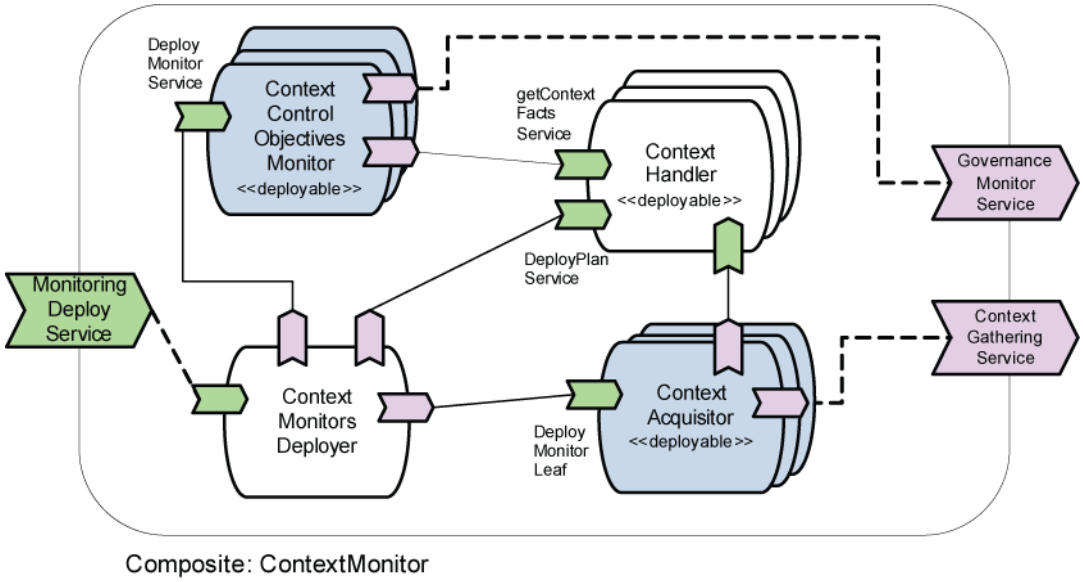


Figure 8-9: Context Monitor Architecture

Once the appropriate context composition level is reached, the *ContextControlObjectivesMonitor* correlates the context facts provided by the context handlers and notifies the *SOAGovernanceMonitor* of the symptoms that describe the monitored situation.

Finally, the *SensingInfrastructure* composite presented in Figure 8-10 defines the sensor discoverers, mediators, and endpoints that are required to acquire context information from the environment. This composite also supports preprocessing of raw context.

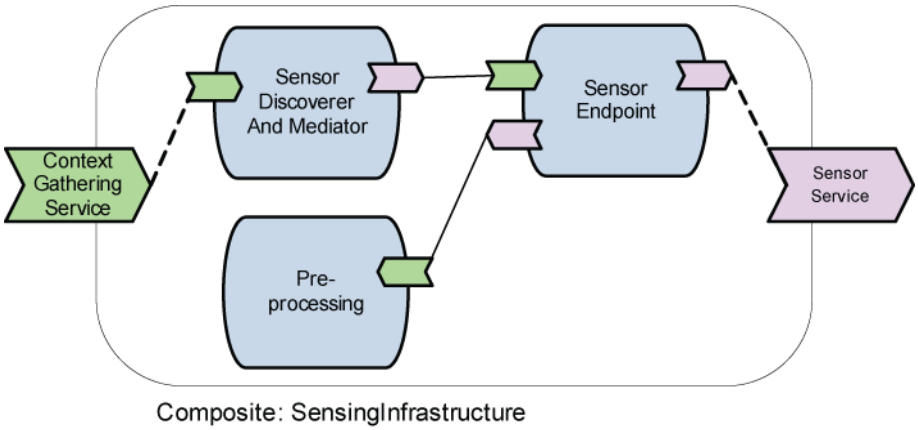


Figure 8-10: Context Sensing Architecture

8.5 Applying the Reference Architecture

This section illustrates the application of our reference architecture proposed in Section 8.4 by defining a concrete SCA that, based on feedback loops, supports the dynamic monitoring of context information that can affect the accomplishment of SLOs for the SLA described in Section 8.2.

A Concrete Architecture for Supporting SOA Governance through SLA Monitoring. Figure 8-11 presents a concrete architecture based on our control-based reference architecture proposed in Section 8.4 for implementing an infrastructure for monitoring SLAs. As this section focuses on the dynamic reconfiguration of monitoring components, the SOA-governance domain is not mapped in Figure 8-11. For the SOA-governance application case described in Section 8.2, this concrete architecture supports monitoring concerns according to the initial SLA that is defined between the hotel and Shopping Broker A to guarantee a minimum transaction rate of 10 transactions per second in the summer and 5 transactions per second for the remainder of the year. The context to be monitored is inferred from the two metrics defined by the initial SLA (number of transactions and time spent) and the corresponding SLO (date/time to control the seasonal variation).

The bottom part of the figure describes the architecture for an outsourced monitoring domain. To achieve the negotiated throughput SLO, Shopping Broker A also establishes an SLA with each store, as shown in Figure 8-2. Thus, the monitoring infrastructure is required to monitor the SLOs stated between the broker and each store. For this particular case, SLAs between the broker and each store also state a minimum throughput similar to the conditions of the SLA between the hotel and the broker. Therefore, we assume the same monitoring strategy for each case.

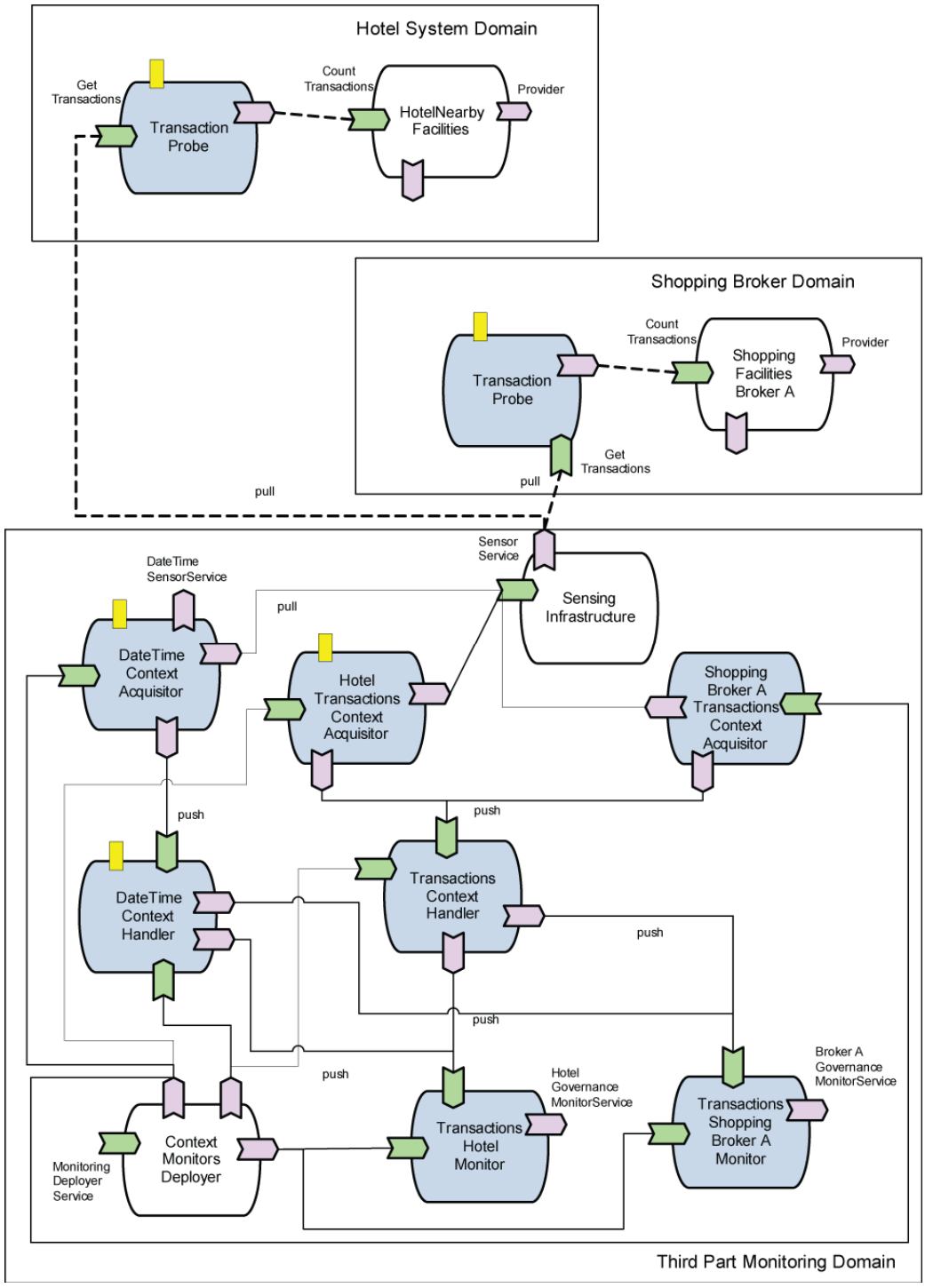


Figure 8-11: Concrete Architecture for the Monitoring Infrastructure before Renegotiating the Initial SLA

A *ContextMonitorsDeployer* composite is in charge of deploying the required components according to the monitoring strategy depicted in Figure 8-12. Monitoring strategies are dynamically defined based on context-monitoring objectives and conditions represented by models derived from the meta-model depicted in Figure 8-4. As presented in Figure 8-12, a monitoring strategy is composed of SLO monitors (gray nodes), context handlers (white nodes),

context acquisitors (rounded rectangular nodes), communication mechanisms, and corresponding algorithms for each handler and monitor. Communication mechanisms can be either bottom-up (i.e., push) and top-down (i.e., pull). For the example SLA from Figure 8-2, two acquisitors are required: *Transactions and DateTime*. Once the strategy is defined by the *MonitoringPlanner* shown in Figure 8-8, acquisitors, handlers, and monitors are dynamically deployed at runtime. As depicted in Figure 8-11, *ContextAcquisitors* continuously gather context from the *SensingInfrastructure*, according to the measurement intervals defined by the context model.

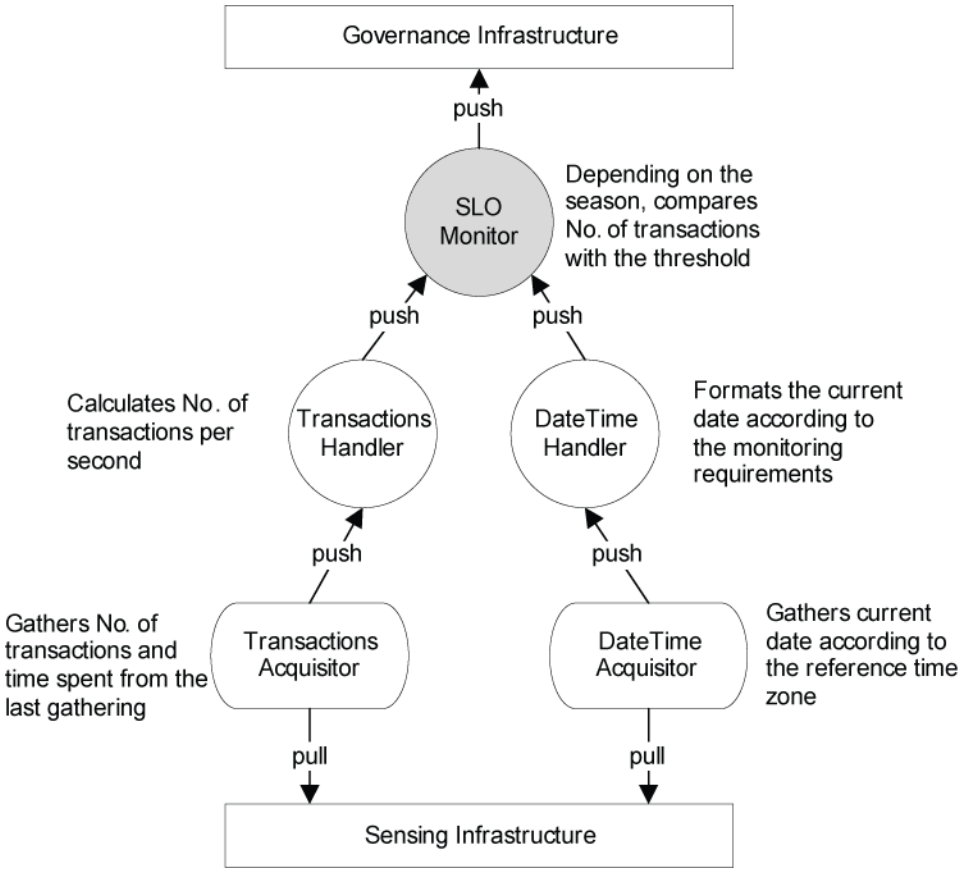


Figure 8-12: Monitoring Strategy for Governing the Initial SLA

Every time context is gathered, *ContextAcquisitors* send context data to the corresponding *ContextHandler*. *ContextHandlers* process context according to the handling algorithm. It is important to point out that many levels of context handlers are possible as they can be composed to generate context facts. High-level *ContextHandlers* push context facts onto the corresponding *ContextMonitor*. Finally, *ContextMonitors* notify the governance infrastructure with the symptoms inferred from the environment according to the monitoring strategy.

Dynamic Adaptation of the Monitoring Architecture. The initial SLA no longer applies under dynamic conditions. Thus, after its renegotiation, new contextual facts must be monitored to support the SOA-governance infrastructure in the regulation of the SOA-governance objectives. Consequently, the context model, the monitoring strategy, and the monitoring infrastructure should be adapted. Suppose the monitoring infrastructure is continuously reporting violations of the throughput SLO. As the governance of the current SLA is not context aware, the parties are

not able to adapt to fulfill the negotiated SLOs. Consequently, the current SLA is manually renegotiated by including context parameters to guide the adaptation of involved services as required by SOA-governance objectives and context. For this particular case, imagine that the parties have identified that the throughput SLO depends not only on the season but also on the occupancy of the hotel. As a result, an *occupancy level* context parameter is included in the SLA.

The addition of this context parameter requires the dynamic adaptation of the monitoring infrastructure. To support this new monitoring requirement, (1) the context model is modified at run-time, in this case, based on the context meta-model shown in Figure 8-4; (2) the SOA-governance infrastructure consumes the *ModelControllerService* to generate a new monitoring plan as shown in Figure 8-8; and (3) the monitoring architecture depicted in Figure 8-11 is reconfigured and redeployed according to the new plan that corresponds to the strategy presented in Figure 8-13. To avoid system degradation, the *MonitoringPlanner* must implement efficient strategies to deploy new components. We propose a planner that is able to calculate differences between the current monitoring strategy and the new one in such a way that only new components are deployed and obsolete ones are destroyed.

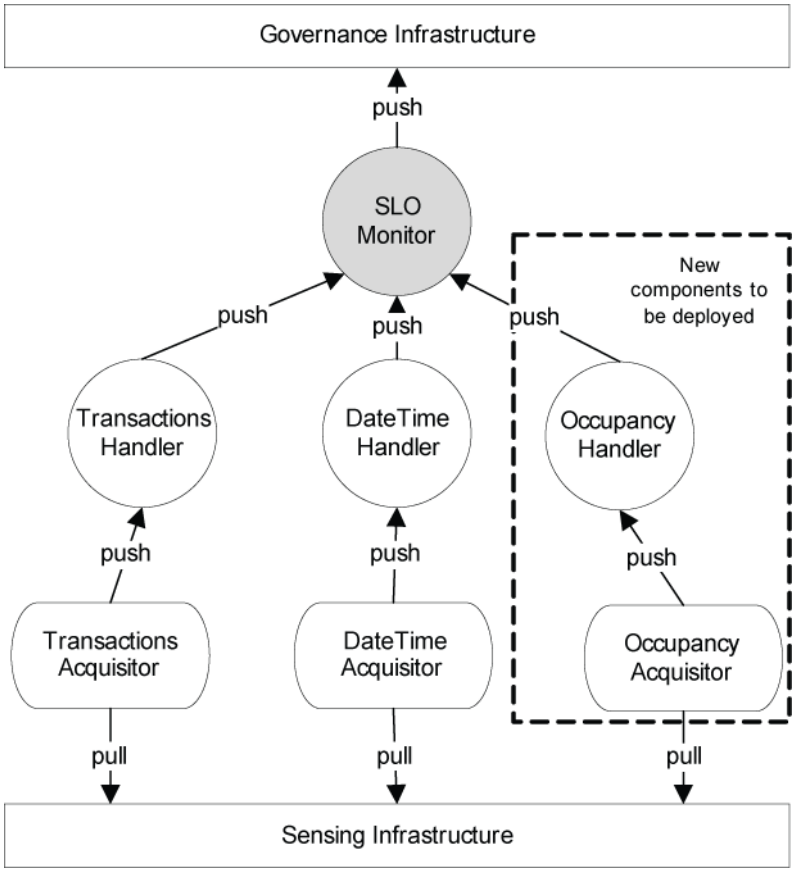


Figure 8-13: Reconfigured Monitoring Strategy for the Renegotiated SLA

8.6 Discussion and Related Work

As the complexity of service-oriented systems is continuously increasing due to the dynamic environment that surrounds chains of service compositions, the problem of monitoring services

for supporting SOA governance, and ensuring the behavior of service-oriented systems in general, remains unsolved. However, some advances have been reached, thanks to research initiatives such as the WSLA framework proposed by IBM [7, 8]. Although the WSLA framework and the corresponding specification seem to not have been widely adopted, both contributions are the foundation of many other approaches, such as the monitoring infrastructure that is part of the enterprise service bus solution for SOA proposed by the European Petals project and others surveyed by Bianco et al. [15, 3]. The WSLA framework provides a set of services, specifications, and tools that support (1) the management of an SLA throughout the stages of its life cycle; (2) SLA deployment to communicate the relevant information for configuring services and components to the involved parties; and (3) SLA monitoring through the measurement and condition evaluation services. With respect to monitoring concerns, the WSLA runtime architecture provides a tool to support SLA negotiation and establishment by retrieving metrics from service providers, generating SLA parameters from metrics, defining parties and their responsibilities, and supporting runtime processing.

Although this framework provides a good foundation, it lacks mechanisms to (1) process related SLAs to identify parameters that must be monitored from third parties when an SLA is affected by many others in chains of service compositions; (2) specify context parameters that can affect the accomplishment of SLOs; and (3) dynamically reconfigure monitoring services to support SLA negotiation at runtime. Our proposed reference architecture constitutes a step toward dynamic monitoring for SOA environments. We propose the evolution of SLA negotiation and establishment from very static SLAs to context-aware SLAs, where context information that can affect SLOs is specified at negotiation time and monitored at runtime. In addition, SOA-governance infrastructures based on our control-based reference architecture are able to reconfigure themselves to support the monitoring of new context requirements stated at runtime as a consequence of changes in SOA-governance objectives.

Other approaches have arisen from concerns related to the management of SOA workflows. Baresi and Guinea propose an aspect-oriented approach that implements self-supervising capabilities for BPEL processes [16, 17]. Their approach is based on the definition of monitoring expressions in the form of web service constraint language (WSCoL) assertions that define facts to be monitored from the process internals and the environment surrounding it. Recovering strategies are specified in the form of web service recovery language (WSReL) assertions and are executed according to the monitored conditions. The system enables human designers to vary the degree of supervision by adjusting parameters through an administration console.

Although this system addresses self-recovering concerns that take into account environmental conditions, monitoring is not dynamic. Thus, self-supervising aspects can be further exploited by applying monitoring strategies based on context-driven feedback loops as proposed in our approach. With respect to the control of the degree of supervision (i.e., priority and level), control loops can automatically adjust these parameters according to violated conditions, therefore avoiding the redeployment of the affected BPEL processes. For this, initial values can be defined for supervision parameters and desired context conditions as reference inputs.

In the same way, variations between measured outputs and reference inputs of contextual conditions, such as service response in Baresi's case study, can be controlled (e.g., by applying transfer functions) to adjust supervision parameters accordingly [18]. Another important aspect

that can be improved is the separation of concerns between business process logic and monitoring logic. Baresi and Guinea's approach is based on Dynamo, which is their previous aspect-oriented approach for context monitoring. Therefore, once monitoring annotations are defined, they must be statically incorporated into the process logic, including the specification of the corresponding probes [19]. Consequently, monitoring strategies in their approach are not dynamically reconfigurable because monitoring assertions are defined at design time and the process redeployment required for the supervision manager takes into account new conditions. In contrast, our approach will leverage context dynamics by applying the context features we proposed in our previous work to the modeling of relevant context at design time [2]. This way, context models are adapted at runtime to support new monitoring requirements by means of reconfiguring context-management strategies.

Toward a Dynamic Context-Monitoring Infrastructure for Supporting SOA Governance. To realize dynamic monitoring for SOA governance, our current research work is focused on three main objectives: (1) the specification and modeling of relevant context and context requirements; (2) the dynamic reconfiguration of the monitoring infrastructure; and (3) the dynamic recomposition of context acquisitors, context handlers, and monitors.

First, with respect to context modeling and context management, our main challenge is the representation of relevant context and context requirements in such a way that they can be derived from SOA-governance objectives at design time and modified at runtime. To face this challenge, we proposed the context meta-model presented in Section 8.3.1 [2]. This meta-model can be instantiated for particular context models at design time when users are specifying context-monitoring requirements and at runtime when environmental changes trigger the adaptation of the monitoring infrastructure.

Second, to address the dynamic reconfiguration of the monitoring infrastructure, service recomposition capabilities provided by the SCA specification can be used [20]. SCA is widely implemented in industrial service-oriented platforms such as Apache Tuscany, IBM WebSphere Application Server Feature Pack for SOA, and FraSCAti [21, 22, 23]. We are evaluating the integration of our proposal into these platforms.

Third, concerning the dynamic recomposition of context acquisitors, context handlers, and monitors, context-management frameworks such as COSMOS provide the basics for implementing handlers as service-oriented context nodes and monitoring strategies as context policies [24]. Similarly, techniques for adaptive algorithms are required to support the adaptation of context handler logic at runtime. Finally, we are defining a set of case studies in collaboration with our industrial partners to analyze the benefits of our approach.

8.7 Conclusions

Runtime monitoring of service-oriented systems is critical for their execution, maintenance, and evolution due to the dynamic, heterogeneous, uncertain, and distributed nature of SOA environments. To advance in the automation of monitoring for runtime and change-time SOA governance, we proposed a reference architecture that exploits elements of feedback control loops to assist SOA practitioners in the design and implementation of distributed and dynamically reconfigurable monitoring infrastructures. Monitoring infrastructures based on our reference architecture are able to reconfigure themselves for supporting new context-management strategies

according to changes in the environment that affect SOA-governance objectives. Moreover, interactions between monitoring and governance feedback loops enable SOA governance to automate the regulation of SOA-governance objectives under dynamic conditions. Thus, our reference architecture can also be extended to support the dynamic adaptation of SOA infrastructures and service-oriented systems.

We are currently working on the definition of mechanisms to specify, represent, and manage relevant context [2]. We are also evaluating available technologies to efficiently support the modification of monitoring plans at runtime and the dynamic reconfiguration of context monitors, context handlers, and context acquisitors accordingly. Taking into account the complexity of context that can affect service-oriented systems, these two activities are required in order to advance in dynamic monitoring for supporting SOA governance.

Acknowledgments

This work was funded in part by the National Sciences and Engineering Research Council (NSERC) of Canada (CRDPJ 320529-04 and CRDPJ 356154-07), IBM Corporation, CA, Inc., and Icesi University (Cali, Colombia).

We would like to thank Gabriel Tamura for his valuable contributions to the definition of our proposed control-based reference architecture, as well as Romain Rouvoy and other researchers from the ADAM team at INRIA Lille Nord Europe for the discussions about topics related to this paper.

References

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC 1999), ser. LNCS, vol. 1707. Springer, Sep. 1999, pp. 304–307 [Online]. Available: http://www.springerlink.com/content/pwpmm42n3krr1f3a/

[2] N. M. Villegas and H. A. Müller, "Managing dynamic context to optimize smart interactions and services," in M. Chignell, J. Cordy, J. Ng, and Y. Yesha, Eds., The Smart Internet: Current Research and Future Applications, ser. LNCS, vol. 6400. Springer, 2010, pp. 289–318 [Online]. Available: http://portal.acm.org/citation.cfm?id=1980659.1980680

[3] P. Bianco, G. Lewis, and P. Merson, "Service level agreements in service-oriented architecture environments," Carnegie Mellon University Software Engineering Institute, Tech. Rep. CMU/SEI-2008-TN-021, 2008 [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/08tn021.cfm

[4] G. A. Lewis and D. B. Smith, "Service-oriented architecture and its implications for software maintenance and evolution," in Proceedings of Frontiers of Software Maintenance (FoSM 2008). IEEE Computer Society, 2008, pp. 1–10 [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4659243

[5] G. A. Lewis, D. B. Smith, and K. Kontogiannis, "A research agenda for service-oriented architecture (SOA): Maintenance and evolution of service-oriented systems," CMU/SEI,

Tech. Rep. CMU/SEI-2010-TN-003, 2010 [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/10tn003.cfm

[6] H. A. Müller, P. Gupta, R. Desmarais, A. Rudkovskiy, N. M. Villegas, Q. Zhu, and N. Leho, "SOA governance optimizes the business and evolution of service-oriented systems," in Proceedings of the 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2009), CMU/SEI, Spec. Rep. CMU/SEI-2010-SR-004, 2010, p. 67 [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/10sr004.cfm

[7] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," IBM Corporation, Tech. Rep. RC22456 (W0205-171), 2002 [Online]. Available: http://www.research.ibm.com/wsla/

[8] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, "Web service level agreement (WSLA) language specification," IBM Corporation, Tech. Rep., 2003 [Online]. Available: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[9] A. Schmietendorf, R. Dumke, and D. Reitz, "SLA management-challenges in the context of web-service-based infrastructures," in Proceedings of the IEEE International Conference on Web Services (ICWS 2004). Washington, DC: IEEE Computer Society, 2004, p. 606.

[10] C. Herssens, S. Faulkner, and I. J. Jureta, "Context-driven autonomic adaptation of SLA," in Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC 2008). Berlin: Springer-Verlag, 2008, pp. 362–377.

[11] H. Giese, Y. Brun, J. D. M. Serugendo, C. Gacek, H. Kienle, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive and self-managing systems," in Software Engineering for Self-Adaptive Systems, ser. LNCS, B. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds. Springer, 2009, pp. 47–69.

[12] OASIS: Advancing Open Standards for the Information Society, "SCA Assembly Model version 1.0," 2007 [Online]. Available: http://www.oasis-opencsa.org/sca-assembly

[13] G. A. Lewis, D. B. Smith, N. Chapin, and K. Kontogiannis, "MESOA 2009: Proceedings of the 3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems," CMU/SEI, Tech. Rep. CMU/SEI-2010-SR-004, 2010 [Online]. Available: http://www.sei.cmu.edu/library/abstracts/reports/10sr004.cfm

[14] H. Müller, M. Pezzè, and M. Shaw, "Visibility of control in adaptive systems," in Proceedings of the 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008). Workshop at the 30th IEEE/ACM International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, 2008, pp. 23–26.

[15] OW2: Open Source Middleware Consortium, "Petals ESB, the open source ESB solution for service oriented architectures," 2010 [Online]. Available: http://petals.ow2.org/index.html

[16] L. Baresi and S. Guinea, "Self-supervising BPEL processes," IEEE Transactions on Software Engineering, vol. 37, no. 2, 2010, pp. 247–263.

[17] OASIS: Web Services Business Process Execution Language WSPEL Technical Committee, "Web services business process execution language WSPEL," 2003. [Online]. Available: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[18] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, Feedback Control of Computing Systems. John Wiley & Sons, 2004.

[19] L. Baresi and S. Guinea, "Dynamo: Dynamic monitoring of WSBPEL processes," in Proceedings of the International Conference on Service-Oriented Computing (ICSOC 2005). Springer, 2005, pp. 478–483.

[20] OASIS: Open Composites Architecture (Open SCA) Member Section, "Service component architecture (SCA)," 2007 [Online]. Available: http://www.oasis-opencsa.org/sca

[21] The Apache Software Foundation, "Apache Tuscany," 2010 [Online]. Available: http://tuscany.apache.org/home.html

[22] C. Sadtler, G. Crooks, J. Laskowski, and S. Mitchell, Getting Started with Websphere Application Server Feature Pack for Service Component Architecture. IBM Redbooks, 2010 [Online]. Available: http://www.redbooks.ibm.com/redpapers/pdfs/redp4633.pdf

[23] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.-B. Stefani, "Reconfigurable SCA applications with the FraSCAti platform," in Proceedings of the 2009 IEEE International Conference on Services Computing (SCC 2009). Washington, DC: IEEE Computer Society, 2009, pp. 268–275.

[24] D. Conan, R. Rouvoy, and L. Seinturier, "Scalable processing of context information with COSMOS," in Proceedings of the 7th IFIP WG 6.1 International Conference Distributed Applications and Interoperable Systems (DAIS 2007), ser. LNCS, vol. 4531. Springer, June 2007, pp. 210–224 [Online]. Available: http://www.springerlink.com/content/t15098547n1311g5/

9 Workshop Review and Next Steps

This section summarizes each workshop session and includes highlights from the discussions that took place among the 17 workshop attendees, who represented both industry and academia. All presentations can be found on the MESOA 2010 web page (http://www.sei.cmu.edu/workshops/mesoa/2010).

The workshop started with an introduction of the SOA research agenda, some of the challenges under maintenance and evolution in the agenda, and how the topic under discussion at MESOA had contributed to the evolution of the agenda. The rest of the day was then divided into seven sessions:

- Session 1: Evolution of Service-Oriented Systems
- Session 2: Migration of Legacy Systems to SOA Environments
- Session 3: Dynamic Adaptation
- Session 4: Longer Term Research Topics in Maintenance and Evolution of Service-Oriented Systems
- Session 5: SOA Governance for Evolution
- Session 6—Invited Session: Challenges for Maintenance and Evolution of Service-Oriented Systems at Credit Suisse
- Conclusion: Workshop Review and Next Steps

Each session included one or more presentations plus a guided discussion, with the goal of identifying remaining research challenges in each area. At the end, the results of the workshop were summarized and next steps were presented.

9.1 Workshop Introduction

Dennis Smith from the Software Engineering Institute (SEI) presented the goals for MESOA 2010, which were to (1) build on previous workshops, (2) continue discussion of current efforts in the maintenance and evolution of service-oriented systems, and (3) identify areas of future work to address existing gaps and provide updates to the research agenda.

Next, Grace Lewis from the SEI presented the SEI SOA research agenda, including specific research topics in the area of maintenance and evolution of service-oriented systems.

Since this was the fourth instance of the MESOA workshop, the introduction summarized challenges that were identified in previous workshops and discussed how these are being followed up. Several of these issues are being actively addressed by the research community, and the research agenda reflects it. For example, *Runtime Monitoring of Service-Oriented System Evolution* has been added to the research agenda under the topic of *Maintenance and Evolution*, and most of the discussion during the introduction focused on this topic.

The main thrust of the discussion was that, given the strong links between business strategy and SOA, runtime monitoring is a recommended practice that can be used to verify that business goals

are being met. However, the real challenge is not how to measure, but what to measure. Current SOA infrastructures provide capabilities to define metrics and collect data, and there is a large amount of currently available monitoring and instrumentation technology. Therefore, the identification of runtime metrics to measure alignment with business goals, as well as a mapping to specific technologies for runtime data collection, is a valuable research area that would contribute to strategic SOA adoption. A suggestion was even made that runtime monitoring should be a cross-cutting concern because it contains elements of engineering, business, and operations.

9.2 Session 1: Evolution of Service-Oriented Systems

The MESOA workshops have placed a focus on the long-term needs for the evolution and maintenance of service-oriented systems. Session 1 discussed how two approaches that have been successful in other domains can be applied to the evolution of service-oriented systems. Marin Litoiu proposed the use of feedback loops and simulation models to provide data for the evolution of business processes. Hamzeh Zawawy addressed a concern from MESOA 2009 for the application of root cause analysis to service-oriented systems.

Simulation Models to Evolve Business Processes

Marin Litoiu from York University focused on mechanisms for fulfilling one of the promises of service orientation, which is runtime agility. The presentation proposed adding a feedback evolution loop to standard business process modeling notation (BPMN) and business process execution language (BPEL) models. The proposed approach creates a simulation model that matches the real system in order to create trust in the results of these what-if scenario analyses.

In this approach, business metrics are defined in the modeling phase at the service and business process level and are then mapped to the simulation model. The simulation model is used for what-if scenario analyses of changes in business processes. In addition, actual system performance is compared to the results of the simulation model, and the model is adjusted accordingly to create a feedback loop. The monitoring of the business processes during deployment provides the basis for evolution and optimization because the data enables runtime tuning of the simulation model as well as development of alternative deployments for better meeting defined key performance indicators (KPIs).

The model was prototyped through a tool that is based on several IBM products (IBM WebSphere Business Modeler, IBM WebSphere Integration Developer, IBM WebSphere Process Server, and IBM Monitor) plus a custom module for estimation, decision making, and integration. Next steps include validation at a larger scale, runtime optimization, implementation of an optimization algorithm, and prediction of future states and KPIs. During the discussion, the challenge of using KPIs that are not time related was analyzed.

Requirements-Driven Framework for Root Cause Analysis

Hamzeh Zawawy from the University of Waterloo presented an approach that applies root cause analysis (RCA) to the analysis of faults in a SOA environment. The goal of RCA is to discover the first or true cause of an incident (such as a software fault). The analysis relies on either

deductions from developing general analytical rules (rule based) or inductions from previous experience (case based).

Zawawy proposes a model-driven diagnostic framework to use goal models for RCA. In this approach, the problem of diagnosing software systems is transformed into a satisfiability problem. In essence, a system failure becomes the failure to deliver the functional or nonfunctional system requirements or goals. Generic log data is annotated with the ability to query goal model information to see if goals have been met. This is an alternative to instrumenting a complete system, which is a much greater and more error-prone effort. To illustrate this approach, a notional goal model of a loan application process was presented. Current challenges in this work include the lack of standards for log data, the large size of the log data, and logging components that fail when errors are injected as part of testing.

The discussion identified three potential avenues for future work: (1) the use of information retrieval, domain-specific languages (DSLs), or metamodels to deal with lack of standards for log data; (2) automation of the creation of the goal model; and (3) extending current work from the business process layer to the creation of new patterns to deal with the infrastructure layer.

9.3 Session 2: Migration of Legacy Systems to SOA Environments

Because of its characteristics of loose coupling, published interfaces, and a standard communication model, SOA enables existing legacy systems to expose their functionality as services, presumably without making significant changes to the legacy systems. While in many cases, the migration to services does not require significant changes to legacy systems, incompatibilities between data types often arise. Harry Sneed from ANECON GmbH proposed an approach for solving this problem that is targeted at web service implementation or any implementation that uses XML as the format for data exchange between consumers and services.

SOA implementations do not require the migration of existing applications to a common language. However, data types from legacy and SOA environments may be incompatible. For example, COBOL uses packed and binary types while PL/I uses floating and pointers, and Java and XML do not recognize any of these data types.

To avoid the ASCII-to-binary conversion that takes place when the service implementation is called from the web service stubs, the paper proposes that all special data types be converted to ASCII format before their migration to services. Once the data types are converted to a common ASCII format, the databases would also be converted using a data conversion utility.

The paper presents the results of a case study of a CICS conversion for a Swiss bank. Examples showed how packed data fields and other data fields are identified by the utility, commented out, and converted to ASCII.

The approach of data conversion to ASCII avoids the problem of runtime data casting that often leads to errors from incompatible data types. However, this approach does require conversion to binary if data is to be used in computation. Discussion centered on the performance cost of these internal conversions to binary as well as a set of decision rules to determine whether these conversions are acceptable.

9.4 Session 3: Dynamic Adaptation

This session had two presentations, but unfortunately, none of the authors of the paper "Towards Proactive Adaptation: A Journey along the S-Cube Service Life Cycle" could attend the workshop.

The presentation by Hausi Müller from the University of Victoria continued the theme introduced by Marin Litoiu concerning the need for a stronger emphasis on runtime validation and verification in service-oriented systems. Müller's position is that we tend to look at SOA in a very static way and place too much focus on design time. This position contrasts with a 2010 U.S. Air Force report on Technology Horizons [1], which states that one-third of the research challenges focused on adaptive systems.

The presentation started with the recognition that, for dynamic software systems, the execution environment is fully known only at runtime rather than at design time. A strategy for dealing with the situation is to push some development activities from design time to runtime so that the system has more information about its execution environment. As a result, validation and verification for these development activities must occur at runtime. Traditional control science focuses on the runtime validation and verification of systems after they adapt at runtime, due to certain conditions. However, a challenge remains in dynamic systems, such as service-oriented systems, as the state space is much larger than in static environments.

To adopt this approach, there are important decisions that need to be made, including the determination of the amount of uncertainty that can be afforded as well as tradeoffs between flexibility and assurance. The presentation showed some of the current research in this area and identified the importance of autonomic feedback loops and adaptive controllers for enabling dynamic control of service-oriented systems. It also emphasized governance and runtime adaptation because of the need to understand, control, and manage the uncertainty and runtime dynamics of service-oriented systems.

9.5 Session 4: Longer Term Research Topics in Maintenance and Evolution of Service-Oriented Systems

Kostas Kontogiannis from the National Technical University of Athens gave a presentation on longer term research topics in the maintenance and evolution of service-oriented systems. The goal for this presentation was to identify ways in which service-oriented systems are evolving and their corresponding maintenance and evolution challenges for the future.

In the end, the targets for service-oriented systems development and deployment are

- simplified development of business services
- simplified assembly and deployment of business solutions built as networks of services
- increased agility and flexibility
- protection of business-logic assets by shielding from low-level technology changes
- improved testability

The following are the top research areas identified by Kontogiannis in the area of maintenance and evolution of service-oriented systems, presented in no particular order.

Convergence of Programming Models for Service-Oriented Systems

A SOA programming model can be defined as a collection of models, techniques, methodologies, and tools for implementing services and assembling them into solutions. Examples of SOA programming models include IBM's Service Component Architecture (SCA), Microsoft's Indigo, and Sun's Java Business Integration (JBI) [2, 3, 4]. The main goals of SOA programming models are to be able to express business logic at a high level and generate code through model transformation that implements services that express the business logic as well as to support the infrastructure to manage these services.

With the increased adoption of representational state transfer (REST) as a web service implementation technology, an interesting topic to explore is the relationship and convergence of lightweight RESTful architectures with heavyweight SOAP/WS-* protocol stack approaches. Related challenges include migration to REST, mixed deployments (WS-* and REST), transaction management, and statefulness issues.

Tools for Supporting the Service-Oriented Systems Life Cycle

Two particular topics of interest in this research area are

- robust and customizable code generators based on industry-segment-specific (or domain-specific) models. Challenges include the need for the generated code to be efficient (e.g., from a performance perspective) and well structured (e.g., it follows proven design patterns).
- model-based tools that support (1) model transformation and code generation, (2) model co-evolution and synchronization, and (3) model round-tripping (forward and backward transformations) to achieve traceability.

New Deployment and Business Models for Service-Oriented Systems

The emergence of cloud computing and social computing brings new opportunities for the deployment of service-oriented systems and the creation of business models around these systems. Topics under this research area include understanding the role or effect that the following technologies (and their corresponding policies) have on service-oriented systems, such as

- cloud computing and virtualization for deploying and testing service-oriented systems
- Software-as-a-Service (SaaS)
- social network infrastructures
- Internet marketing
- mobile computing
- context awareness and context sensitivity

Service Versioning—Diversity and Complexity

Service versioning so far has focused on techniques for maintaining different versions of a same service without disrupting service consumers. Research in this area should expand this concept to

account for the complexity and diversity of services as well as the complexity and diversity of service infrastructures:

- models and techniques that enable the automatic or semiautomatic identification of mismatches or potential mismatches between the service-as-specified and the service-as-deployed that go beyond syntactic differences to focus on behavioral and operational aspects as well
- techniques that enable global service update management in the form of models and analysis tools for performing different types of what-if analyses, such as the effects of component substitution or interface changes

Property Tracing—From Design Time to Runtime

Impact analysis determines the impact of system change on service consumers. An extension of this work could include risk analysis and might focus on overall system behavior and system properties from design time to runtime. Examples of research topics include

- techniques to assess the impact of a change in a service or connections between services, such as a change in transaction policies, endpoint implementations, or connectivity properties
- techniques and models to assess the vulnerability of a service-oriented system with emphasis on security issues, such as denial of service (DoS) attacks or access control
- techniques and models to assess the risk and impact of service unavailability (i.e., what is the risk and what would happen if a service becomes unavailable?)

Logging, Monitoring, and Diagnostics

The focus of this research topic is the use of logging, monitoring, and diagnostics not only to cover all elements of service-oriented systems but also to help with early detection and reaction to failures:

- techniques for the nonintrusive collection of events in a customized or adaptive manner (e.g., increase or decrease the level of monitoring according to policies, perceived risks, and vulnerability criteria) that also facilitate the processing and analysis of collected events (e.g., efficient event reconciliation, log filtering, and log amalgamation)
- techniques for “preflight” and “in-flight” checks and RCA
- techniques for seamless and continuous cross-cutting monitoring of service-oriented systems at infrastructure, application, and business process abstraction layers
- techniques for runtime KPI monitoring and selection of remedial actions when KPIs are not being met

9.6 Session 5: SOA Governance for Evolution

Governance policies, monitoring, and enforcement have long been recognized as factors that distinguish successful SOA implementations from unsuccessful ones. Because services, service consumers, and infrastructures often reside in different organizations, there is often no single controlling authority, and responsibilities of different sets of stakeholders need to be clearly

defined. Many SOA infrastructures have the capability for runtime monitoring of the performance of services and applications. In this session, the insights from the control theory and the self-adaptation communities were applied to service-oriented systems. The session presented three perspectives that propose to introduce governance policies and monitoring into the operations of service-oriented systems.

Characterizing Policies That Govern Service-Oriented Systems

Hausi Müller from the University of Victoria presented the results of a comprehensive survey of SOA-governance literature that resulted in a categorization of policies for the governance of service-oriented systems. Policies are divided into the phases of the service life cycle as proposed by IBM and others: service planning, service design, service development, service transition, service operation, service monitoring, and multilevel policies.

The purpose of this survey was not only to identify the policies that apply to each phase of the service life cycle but also to understand how policies can be used as feedback loops for managing, monitoring, controlling, and adapting services to better adjust to changing business needs.

The discussion focused on how the taxonomy is useful in providing insight on KPIs and other aspects that can be managed within a system.

Context-Driven Adaptive Monitoring for Supporting SOA Governance

Starting with the premise that service-oriented applications are highly dependent on environmental information, Norha Milena Villegas from the University of Victoria proposed an approach for runtime monitoring of context in dynamic situations.

The approach includes

- a feature-based model to represent context and requirements for monitoring
- SOA-governance runtime-monitoring objectives that are adaptable to environment changes
- monitoring infrastructure along with adaptable monitoring mechanisms and feedback loops
- runtime renegotiation of governance objectives and SLAs based on current environmental conditions

The talk presented a reference architecture with feedback loops to implement dynamic context modeling. The model includes an internal loop to monitor context and an external loop to control governance objectives. The loops interact to adjust control governance objectives in reaction to changes in context.

The discussion focused on the fact that a major differentiation of this approach from others is the adaptation of the monitoring strategy as opposed to just the managed system.

A Dynamic Framework for Quality Web Service Discovery

Hausi Müller from the University of Victoria presented ongoing work on incorporating quality attributes into matchmaking algorithms for dynamic service discovery. The proposed approach combines static discovery with dynamic selection. This presentation talked about updates performed to their Service Discovery Dynamic Selection (SDDS) architecture to include a new

web-based synchronization component to enforce valid communication patterns. The original SDDS architecture had been effective in measuring the relevance of services for a particular context. However, because of some invalid transitions, there were cases in which it did not maintain the validity of registered services. The updated approach defines dynamic service attributes and then uses these dynamic attributes as a secondary criterion for service selection. It also adds service consumer context to identify selection criteria automatically.

The discussion emphasized that this approach is very domain specific. The creation of ontologies or data models for representing context will be required to make it feasible. There are also potential security and privacy concerns due to context propagation.

9.7 Session 6—Invited Session: Challenges for Maintenance and Evolution of Service-Oriented Systems at Credit Suisse

Carl Worms from Credit Suisse gave an invited talk on the challenges, lessons learned, and future directions of the Credit Suisse SOA implementation that has been underway since 1998. It currently serves 67,500 users in 550 locations and is supported by an IT organization of 14,000 employees.

The initial implementation was based on CORBA plus WebSphere Message Broker and MQ for messaging. They are currently migrating to web services. The initial goals for their SOA implementation were to create greater efficiency for their 6,700 mainframe applications and to make reusable business data and functionality available across the organization. Over the years, they have established an enterprise architecture that drives all systems decisions. As of 2010, there are 1,200 available services.

The main challenges identified by Worms in the maintenance and evolution of their SOA implementation include

1. managing complexity that is due mainly to size (~6,700 applications; 10 host CPUs; 93,500 workstations).
2. finding the proper IT architecture governance and structure. This includes making decisions on appropriate business processes that are candidates for services; mapping between business processes, applications, and components; and determining the most effective IT infrastructure.
3. anticipating the future in 5 to 10 years. For example, one of their goals is to go from monolithic to loosely coupled components/sub-domains with interfaces along the borders of sub-domains instead of direct access between consumers and components.
4. managing and federating multiple integration infrastructures—service, messaging, file transfer, and portal infrastructures.
5. making SOA scale for more than 1,000 services. They created an in-house Interface Management System (IFMS) that provides a service catalog, a design tool, governance enforcement, life-cycle management, and code generation.
6. adapting the software engineering life-cycle models for SOA. An example is the creation of an Interface Engineering Process that includes early generation of test cases, generation of mocks for testing and support, preparation of test environments, and compilation of test reports.

7. testing and versioning of interfaces. It was necessary to establish clear criteria for what constitutes a major and a minor version, as well as testing processes for major and minor versions for the service provider and the service consumer.
8. migration from regional and national services with single backend platforms to global services with broadly distributed backend platforms.

9.8 Next Steps

The SOA research agenda has proven to be a robust and evolving approach for identifying and organizing research in this area.

In 2011, there are two events related to the evolution of the SEI SOA Research Agenda:

- The 3rd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS 2011) in conjunction with the 33rd ACM/IEEE International Conference on Software Engineering (ICSE 2011) on May 23–24, 2011, in Hawaii, USA (http://www.s-cube-network.eu/pesos-2011).
- The IEEE Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2011) colocated with the 27th IEEE International Conference on Software Maintenance (ICSM 2011) on September 26, 2011, in Williamsburg, VA, USA (http://www.sei.cmu.edu/workshops/mesoca/2011). This workshop will expand the discussions to include the relationship between service orientation and cloud computing, challenges of migrating systems to the cloud, and challenges of using cloud resources and services.

References

[1] United States Air Force Chief Scientist. Report on Technology Horizons: A Vision for Air Force Science and Technology During 2010–2030. AF/ST-TR-10-01, 15 May 2010.

[2] IBM. Service Component Architecture. 2006. http://www.ibm.com/developerworks/library/specification/ws-sca

[3] Microsoft Corporation. Introducing Indigo: An Early Look. 2005. http://msdn.microsoft.com/en-us/library/aa480188.aspx

[4] Sun Microsystems. “Java Business Integration.” In Sun Java System Application Server 9.1 Administration Guide, 49–52. Santa Clara: Sun Microsystems, 2010.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2011	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Proceedings of the Fourth International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2010)	FA8721-05-C-0003

6. AUTHOR(S)

Grace A. Lewis, Dennis B. Smith, and Kostas Kontogiannis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2011-SR-008

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The Fourth International Workshop on Maintenance and Evolution of Service-Oriented Systems (MESOA 2010), organized by members of the Carnegie Mellon Software Engineering Institute's technical staff, was held at the 26th International Conference on Software Maintenance (ICSM 2010) in Timi oara, Romania, on September 17, 2010. The goal for MESOA 2010 was to share current research efforts and discuss emerging technologies in the maintenance and evolution of service-oriented systems. A second goal of the workshop was to identify areas of future work needed to address existing gaps and problems in the taxonomy of research topics in service-oriented architecture (SOA). This report summarizes the workshop and includes the accepted papers that were the basis for the presentations given during the workshop. Topics include using simulation models to evolve business processes, a requirements-driven framework for root cause analysis in SOA environments, SOA integration as an alternative to source migration, proactive adaptation as illustrated by the S-Cube service life cycle, a dynamic framework for quality web-service discovery, a characterization of policies that govern SOAs, and context-driven adaptive monitoring for supporting SOA governance. The report concludes with highlights from the discussions among workshop attendees.

14. SUBJECT TERMS	15. NUMBER OF PAGES
service-oriented architecture, software architecture, service-oriented systems, software maintenance, software evolution	157

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL