Decentralised QoS-Management in Service Oriented Architectures

Markus Schmid and Reinhold Kroeger

Wiesbaden University of Applied Sciences
Distributed Systems Lab
Kurt-Schumacher-Ring 18, D-65197 Wiesbaden, Germany
{schmid,kroeger}@informatik.fh-wiesbaden.de

Abstract. Traditional hierarchical Service Level Management (SLM) frameworks fail to cope with the challenges imposed by the runtime dynamics of Service Oriented Architectures (SOA). This paper introduces a decentralised management approach that successfully uses emerging self-management techniques to realise a flexible SLM system and presents an architecture that implements this approach. The architecture consists of a modular self-manager framework that provides the basis for component-level and workflow-level management. It provides sensor and effector modules to monitor and manage different classes of applications. Integration with existing SOA components is based on the Service Component Architecture (SCA). The presented framework has been prototypically implemented and is currently evaluated in terms of efficiency and scalability.

1 Motivation

Traditionally, *Service Level Management* (SLM) is the discipline concerned with monitoring and management of processes and applications according to agreed-upon *Quality of Service* (QoS) criteria. In service provisioning relationships, provider and customer agree on QoS criteria and failure penalties in formal contracts, called *Service Level Agreements* (SLAs). SLAs contain *SLA Parameters* that define QoS aspects to consider and *Service Level Objectives* (SLOs) to be met regarding these parameters.

At runtime the agreed-on SLOs are monitored by a dedicated SLM architecture. Based on this monitoring information administrators and operators take care of necessary system reconfigurations. In current installations, a SLM architecture is often integrated with large-scale enterprise management systems, e.g. *HP OpenView*, *IBM Tivoli* or *CA Unicenter*. These systems started as management frameworks, and today consist of a number of more or less closely integrated components. The frameworks originate from the network management area and therefore implement a centralised, relatively static, and strictly hierarchical management approach.

However, looking at SLM on the application-level, emerging Service Oriented Architectures tremendously increase the overall complexity of enterprise applications, both in terms of the number of components involved and in the overall

runtime dynamics of the resulting system. In addition, the current trend towards virtualisation of computing resources adds another layer with dynamic bindings and thus aggravates the matching of application level failures to physical resources in large-scale systems.

In this paper, we present a decentralised and adaptive SLM architecture for dynamic SOA-based applications. The architecture employs self-management techniques to realise SLM for individual services. The structure of the management system automatically adapts to the SOA's business architecture.

The paper is structured as follows: section 2 describes the characteristics of emerging SOAs and presents commonly used implementation technologies. In section 3 we discuss challenges for SLM in SOA environments and give a short introduction to currently emerging self-management approaches. Our decentralised SLM architecture, which in parts relies on self-management, is presented in section 4. This section also gives a description of implementation details. Related work is discussed in section 5. The paper closes with a conclusion and a description of future work.

2 Service Oriented Architectures

Traditionally, *multi-tier architectures* are used to implement large-scale enterprise applications. They provide a clear separation of presentation, business logic and data storage, which alleviates the impact of a change in one of these tiers regarding the rest of the application. Multi-tier architectures are often based on standard middleware, e.g. J2EE or CORBA, with relatively static component bindings. Benefit of a multi-tier architecture is the stability of the interfaces between components in different tiers. A drawback however is the inability to perform quick reorganisations as business needs change – the rather static design of a multi-tier application results in an inability to quickly follow changes in the overall organisational structure of an enterprise. For that reason a more flexible and dynamic enterprise software architecture has evolved in recent years:

Independent and loosely coupled services define the building blocks of a *Service Oriented Architecture*. All services in a SOA environment are accessible in a standardised way, as they inter-operate based on a formal interface definition which is independent of the underlying computing platform and programming language. Services are dynamically composed into business workflows to form applications. This breakup into workflows and (shared) services however makes the concept of strictly separated applications dispensable.

At runtime, workflow descriptions are interpreted by a *workflow management system (WfMS)* that invokes the participating services. A major design goal for SOA is to bring the architecture of enterprise IT applications in line with the enterprises' organisational structure. Thus, while services are considered as static entities, SOA workflows may be adapted to business needs and thus can change on a regular basis. In todays B2B scenarios SOA workflows can even span across administrative boundaries of organisations. From an IT management perspective this complicates the enforcement of quality-of-service parameters.

Looking at the technical realisation of a SOA we distinguish several abstraction layers within the architecture (see fig. 1). At the lowest layer are the operational systems, networked hardware resources, operating systems and so forth. These resources are utilised by enterprise components, which themselves provide service interfaces. The second layer, called service layer, which we can also find in traditional multi-tier enterprise applications, is the typical domain of existing hierarchical SLM-approaches for applications. On top of the service layer we find the workflow orchestration layer, which dynamically involves the underlying services. Service interfaces hide all implementation details from the workflows in this layer. Workflows do also provide a service interface to the outside world and thus may themselves be accessed by other workflows in the same way like basic services. This allows to design complex, nested workflows.
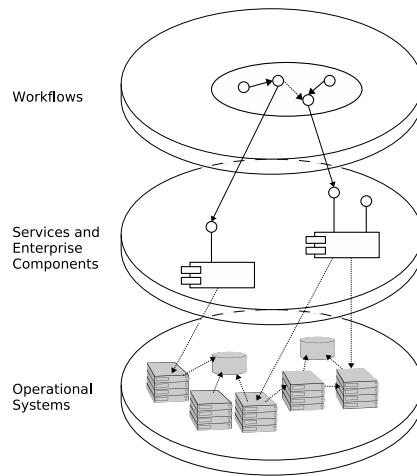
Today, common technologies for implementing a SOA environment are Web Services based on the Web Services Description Language (WSDL) for the description of service interfaces, SOAP as communication protocol, and the Business Process Execution Language (BPEL) [1] for the description of business processes and workflows. BPEL is an XML-based notation that defines a number of so-called BPEL activities. Activities represent single steps of a workflow, e.g. synchronous or asynchronous service invocations, variable assignment and evaluation, case differentiations, loops, etc. BPEL activities are divided into basic activities that include service invocations and other straight-forward operations, and complex activities which wrap a number of basic activities (e.g. loops). Regarding SLM on the workflow layer, BPEL activities are of special interest, as they reflect the progress of the real-world business activities. In terms of QoS observation, complex BPEL activities can be expressed as a combination of basic activities. QoS characteristics of these activities currently can be monitored, but a reconfiguration of services (accessed through BPEL activities) according to SLA requirements is difficult in dynamic environments (see[2]).

The *Service Component Architecture* (SCA) [3] is a specification that allows to create a standardised view on services, workflows and their static interdependencies within a SOA. As such SCA complements workflow modelling languages, like e.g. BPEL, which concentrate on runtime aspects of component interactions. SCA Revision 1.0 has been specified by the *Open SOA Collaboration*[1], an industry consortium that consists of a number of IT companies with SOA activities

Fig. 1. Technical layering of a SOA

---
[1] See http://www.osoa.org for details

(e.g. IBM, Sun, Oracle, SAP, and BEA). Further standardisation of SCA in the meantime has been transferred to OASIS.

SCA models services and workflows as *SCA Components*. These components comprise any number of interfaces, named *SCA Services*, and dependencies (*SCA References*) to other SCA services. A dependency between two components is named *SCA Binding*. In addition SCA components can specify a number of static *SCA properties* to be accessed during runtime.

SCA components and their bindings can be grouped into an *SCA composite*, which hides its inner structure from the outside and thus can be handled the same way as a plain SCA component. This allows to create recursive structures of SCA composites within a SOA. Services and references of SCA composites are specified by propagation of component interfaces or references.

Figure 2 depicts the graphical notation of an SCA composite `ABC`, which comprises three SCA components. The composite offers a service `a'`, which is propagated from the contained SCA component `A`. `A` also holds a Property P_1. In addition, the composite defines bindings between the components `A`, `B` and `C` and propagates the reference `d` of `C` to the outside.
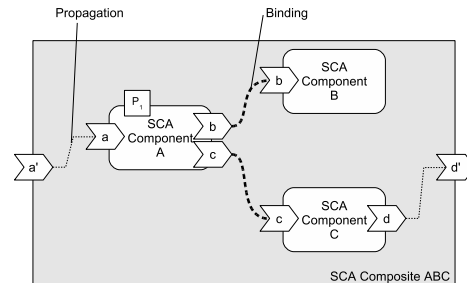

Fig. 2. Example SCA composite comprising three SCA components [3]

In order to support administrative tasks, SCA defines the concept of *SCA Domains*. Common policies can be applied to all domain entities. Currently SCA assumes that within one SCA domain components and composites of just a single vendor are deployed. This allows vendors to implement proprietary binding protocols. Currently SCA composites cannot spread across domain boundaries, however inter-domain communication between components is possible.

To date, SCA defines a number of different mappings for component implementation (*SCA implementation bindings*): there are Java and C++-bindings, but also bindings for BPEL, Enterprise JavaBeans (EJB), Java Messaging Service (JMS) and Spring. The component structure, bindings and services are detailed in an XML-based *Service Component Description Language* (SCDL) descriptor which is interpreted by an SCA runtime in order to instantiate the defined implementation bindings.

3   Necessity for Self-Management

For several reasons a strictly hierarchical and centralised approach is not applicable for establishing SLM within a SOA:

(1) Because of the flexible, dynamic and compositional structure of a SOA, traditional static management structures cannot adapt fast enough to the system dynamics.

(2) SOA environments implement large scale processes. Traditional centralised management approaches with semi-automatic problem solving strategies do not scale sufficiently to meet SOA demands.

(3) A SOA may well spread across enterprise boundaries and therefore also across management domains.

(4) SLAs may be defined on different abstraction layers (e.g., for parts of workflows, or single services). Conflict resolution strategies must take organisational boundaries into account.

An SLM architecture for SOA has to consider the complexity of a SOA environment while it has to cope with permanent changes in composition and cooperation.

Recently, *self-management* approaches have become popular, because they aim at reduced management complexity (for the human administrator) and increased scalability. In addition, the introduction of self-managing system components allows to establish a decentralised management architecture and thus to provide increased stability on a global level.

Self-management projects the principles of autonomic computing to the domain of IT-management. [5] gives a compact overview of current challenges in the self-management domain. Self-management summarises approaches for autonomic reconfiguration, error recovery and optimisation of system behaviour of hard- and software components. [4] describes relevant attributes of self-managed systems as *Self-X Properties*. In contrast to traditional management architectures, where a human administrator controls the system, self-managed systems are controlled by algorithms that – within certain constraints – operate autonomously.

Fig. 3. Structure of an autonomic manager, as defined by [4]

Figure 3 depicts the principle structure of an autonomic manager. The manager is loosely coupled with a managed system through well-defined sensor and effector interfaces. Sensors are used to retrieve information about the current state of the system, effectors are used to dynamically reconfigure the system with the aim to drive it to a desired state. A manager may for example change a systems' strategy in terms of CPU and memory allocation, or may trigger the reinitialisation of a certain sub-module.

In a self-management setting, the autonomic manager and the managed system form a unit: the self-managed system. Such a system can again offer high-level sensors and effectors to the outside world, thus reducing the globally visible complexity of the system. As a result, a self-managed architecture can consist of several layers of control loops with increasing levels of abstraction.
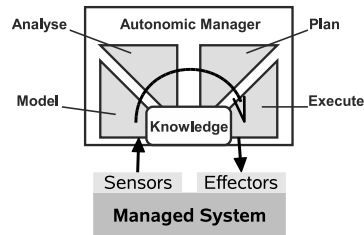
Self-management alone however does not provide sufficient adaptability to global business goals. For that reason we suggest to apply self-management techniques only to SLM on the service layer of a SOA environment and to align the overall management architecture to the structure of the SOA's business processes.

4 Decentralised Management Approach

4.1 General Approach

A flexible SLM architecture for SOA environments has to meet the challenges described in section 3. Our SLM approach for SOA applications aims at providing scalability and flexibility through its decentralised structure, which uses self-management mechanisms for management automation at the service-layer. The SLM architecture automatically aligns with the structure of the business processes defined, as each SOA business component (that is all corresponding workflows and services) is associated with a *Manager* component, which is responsible for monitoring the components' behaviour with regard to its previously defined QoS requirements. Each manager component offers an interface for communicating QoS requirements.

Our approach realises a logically layered SLM management architecture, as managers associated to workflow components communicate with the managers of the participating services in order to enforce the QoS requirements that have been defined for a workflow. QoS requirements are represented as SLAs, which can be specified for workflows or individual services. Each manager gets assigned one or more individual SLOs, in the following termed *iSLOs*. The approach uses WSLA [6], an XML-based specification for SLA description as a formal notation for SLAs.

In the following, the underlying common architecture for service and workflow managers is presented. Afterwards, we describe the functionality offered by managers for services and managers for workflows. Last, we present the integration of our architecture with SCA-based SOA components.

4.2 Generic Manager Architecture

In compliance with the IBM reference architecture in [4], we have developed a modular self-manager framework that provides a customisable basis for the managers on the service and the workflow layers.

The core manager framework supports three different kinds of extension modules (see fig. 4 for details): `event modules`, `action modules`, and `control modules`. Event modules possess their own threads and thus are able to react actively to changes within the environment, e.g. by creating internal messages. `Action modules` are passive; they act – triggered by internal messages – by analysing application-specific sensors, or performing management tasks. Sensors can be realised using either `event modules` (push model) or `action`

`modules` (pull model). Application-specific actuators are realised through `action modules`.

`Control modules` form the "brain" of the self-manager as they contain the management knowledge and implement control algorithms. `Control modules` act periodically or are triggered by incoming messages. Management decisions are communicated to other modules using the internal messaging capabilities.
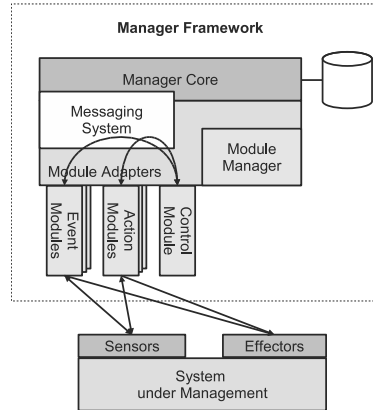

Fig. 4. Modular architecture of the underlying management framework

At startup the manager core starts a `module manager` component, which then instantiates the configured extension modules and controls their lifecycle. Each instantiated extension module is in one of the states `DOWN`, `UP`, or `ERROR`, the module manager regularly checks the state of the modules and is able to stop and reinstantiate modules that are in the `ERROR` state. Dependencies on the availability of other modules are also handled by the module manager (e.g. relevant event and action modules are to be started usually before the corresponding control module). Module configuration is remotely accessible through a management interface, which in principle allows runtime reconfiguration and reinstantiation of manager modules.

Managers on the service and workflow layers are designed to integrate into an SCA-based SOA as SCA components (see fig. 5). Each manager consists of the core manager described above, a number of extension modules, and an SCA adapter that provides SCA-compliant connectivity to other components, e.g. by offering a management interface m that is used for communicating QoS requirements.
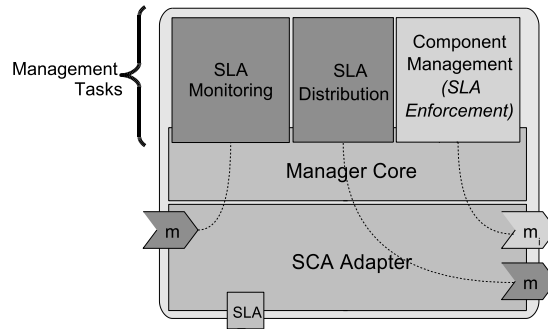

Fig. 5. Overall architecture of a manager component

Depending on the position of the manager in the architecture, its functionality is enhanced with one or more task-specific extension modules, namely for (A) `SLA distribution`, (B) `SLA monitoring and escalation`, and (C) `SLA enforcement`.

In the following we discuss the assignment of these management tasks to service and workflow managers.

4.3 Service Managers

For service managers, SLA monitoring (B) is performed with the help of appropriate event and action modules. Usually SOA services are realised based on enterprise software stacks that also provide standardised monitoring and management interfaces (e.g. [7–9]).

Several `action` and `event` modules have been implemented, that allow a powerful interaction with management interfaces, typically available in the domain of business-critical applications:

- A *Web Services Distributed Management (WSDM)* [10] module handles generic Web Services management invocations, implementing the *Management of Web Services (MOWS)* part of WSDM.
- The *Web-based Enterprise Management (WBEM)/Common Information Model (CIM)* [11] module allows the self-manager to act as a CIM client.
- A *Java Management Extensions (JMX)* [12] module allows to control any JMX-instrumented application.
- An *Application Response Measurement (ARM)* [13] module can retrieve performance-related information (e.g. response times) from ARM-instrumented applications.
- Command-execution supports the execution of shell scripts and other locally available executables.

Due to the modular architecture of the framework additional modules can be implemented without much effort.

SLA monitoring and escalation is performed by each manager in a separate SLA parameter-specific control module. A manager permanently monitors whether the iSLOs defined for the service are met, and – in case of a violation – notifies the requesting party (the manager, which communicated the SLA).

SLA enforcement (C) is solely performed by managers on the service layer. In order to fulfil the iSLOs that have been agreed on, a manager uses self-management techniques to reconfigure its managed service. Reconfiguration makes use of application specific interfaces (symbolically depicted as m_i in fig. 5) and may comprise dynamic resizing of application clusters, reallocation of resources, migration of virtual machines, or other highly component-specific tasks. Therefore, it is impossible to specify a generic SLA enforcement module, however managers typically implement a specialised controller module for the management algorithm and can possibly utilise existing action and event modules.

In order to customise a manager for controlling a software component that provides a service, a service vendor has to select appropriate event and action modules for monitoring and control of the service. As an example, one would probably choose the JMX module to control a Java-based service implementation. Performance monitoring of a service that runs on an IBM WebSphere application server can be achieved using the ARM event module as WebSphere offers an ARM-compliant performane monitoring interface. In addition, the control algorithm that is used for SLA enforcement has to be customised for the

management of the software component, i.e. to reflect the possible reconfigurations offered by the software components' JMX interface.

In [14] we give an example for the customisation of a service manager component by describing an example SLA enforcement mechanism for a dynamically resized Cluster of JBoss application servers that uses a predecessor of the presented management framework: The approach minimises the resource consumption of the JBoss servers while still granting a maximum response time for the requests served. For this management scenario a state machine acts as self-management controller for the cluster, the communication with the JBoss cluster is realised based on ARM and CIM/WBEM modules.

4.4 Workflow Managers

SLA distribution (A) is a task primarily assigned to managers that are responsible for workflow components. SLAs that are assigned to a workflow need to be adequately distributed to the services that are involved in executing the workflows' steps. The distribution of the SLA comprehends an SLA parameter-specific fragmentation of the original SLA into iSLOs for the individual services and can e.g. take historical data into account. Afterwards, the iSLOs are communicated to the managers of the participating services via the SCA adapter.

SLA distribution has to take into account, that services offered by external providers are typically accessed with a fixed SLA (that has been previously negotiated by the parties), which cannot be manipulated by the SLM system. Such an SLA is treated as constant in the fragmentation process. In addition, also fully unmanaged services can be invoked by a workflow. Such services introduce a certain degree of uncertainty regarding the overall QoS behaviour of the workflow. Unmanaged services initially are assigned a random iSLO which is later adjusted according to monitoring information.

Fig. 6 gives an example for the process of SLA fragmentation and distribution. We discuss the fragmentation exemplarily for an SLO t_{max} that limits the maximum response time of the workflow. Fig. 6, part a) depicts the graphical representation of a workflow that invokes four different services, C1 - C4. After the request to C1 has returned, C2 and C3 are executed in parallel in a loop. Afterwards the service C4 is invoked.
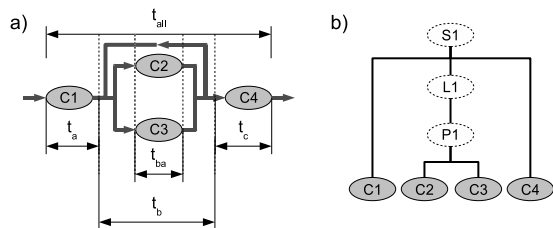

Fig. 6. SLA fragmentation example

Fig. 6, part b) shows the overall structure of the workflow, which is used to fragment the global SLO into iSLOs for the participating services. Initially, t_{all} is composed of three components t_a, t_b and t_c, where t_a describes C1 and t_c describes C4. $t_b = n * t_{ba}$ represents the loop execution time, being n the number of iterations and t_{ba} the maximum execution time of C2/C3. Ideally, t_a, t_{ba}, t_b, and n are estimated from historical

data, which allows to proportionally fragment the SLO into iSLOs for C1 - C4. If no historical data is available, the proportions for the fragmentation have to be selected randomly and need to be readjusted later.

As for managers on the service layer, SLA monitoring and escalation (B) is executed by each manager on the workflow layer in a separate control module. In order to increase the overall system stability, a workflow manager only sends a notification in case the SLA for the workflow is violated, but does generally not forward notifications from individual participating services. In case a workflow manager receives a notification from a participating service, SLA distribution is triggered again to perform a restructuring of the existing SLA fragmentation, aiming at relaxing the iSLO of the component that sent the notification.

For workflow managers SLA monitoring is implemented generically for each supported SLA parameter. Workflow managers internally monitor workflow progress and calculate SLA parameters like workflow response times and throughput from this data.
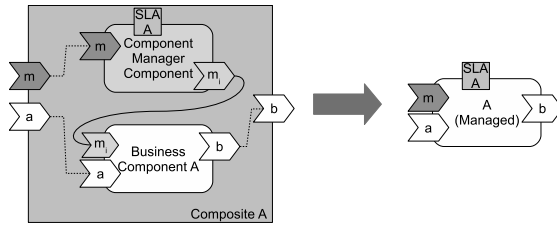
4.5 SCA Integration

Fig. 7. Managed SCA Composite

In order to integrate managers transparently with existing business components, we make use of the SCA composition feature. Figure 7 depicts the association between a business component and a manager: Manager and business component are grouped into a single SCA composite, which propagates both the interface of the business component and the managers' management interface to the outside world. In addition, existing references to other components are also propagated by the SCA composite. We assume that workflow components are realised using the SCA BPEL implementation binding as the manager component at runtime needs to access structural workflow information; services may use any SCA binding available. The major advantage of the composition of business component and manager into a *managed SCA composite* is, that the existence of the management component remains transparent for management-unaware services that reference the business component. A drawback is an increase in the response time of requests sent to the managed SCA composite, that is in parts caused by the transport protocol used (e.g. one additional Web Service call between composite and service) but also depends on the overhead of the SCA runtime. We performed measurements for a worst case scenario that consists of an empty service implementation, which provides an interface with two double parameters. On an Intel Pentium M, 1.6 GHz (Apache Tuscany SCA Runtime, Apache Tomcat 5.5 AS) we measured a mean response time of 3.02 ms per invocation for the pure service implementation and 8.75 ms

for a composite that references this service. This response time contains an overhead of $\tilde{2}$ ms for SCA processing. In a real world SOA setting these calls however typically take much longer as services perform complex business tasks while the SCA processing overhead remains constant.

In a SOA, services are typically accessed by multiple workflows at a time. The presented management architecture is able to cope with several concurrent SLAs, by using a QoS-Proxy mechanism as depicted in fig. 8. Here a management proxy component offers several service queues, one for each SLA to be met. The proxy references the business interface of the managed service and propagates this interface multiple times, thus offering the same service in different qualities. External services reference one of the QoS proxys' interfaces instead of the managed service itself.
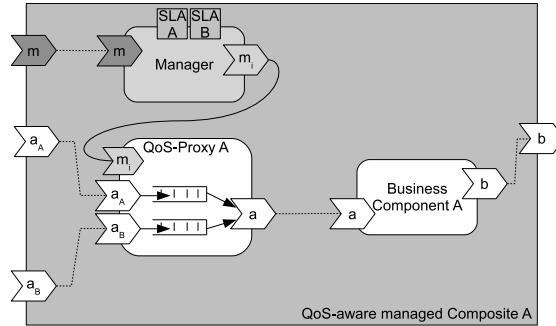

Fig. 8. Managed SCA composite with QoS proxy component

The manager however can implement a number of different strategies for SLA enforcement, e.g. priority-based enforcement, or approaches known from the networking area such as weighted fair queueing or bandwidth management.

4.6 Prototypical Implementation

The presented self-management framework has been prototypically implemented in Java. Internal communication is based on a lightweight, process-local *Java Message Service (JMS)* implementation. The integration with SCA is based on the Apache Tuscany SCA project runtime. The SLA distribution functionality is based on the BPEL parser of the Apache ODE BPEL engine.

In our lab the architecture has been applied to example scenarios for SLA enforcement that have been implemented based on Apache Tomcat and JBoss as underlying Middleware. In this context, we used Apache Axis2 and JBoss for Web Service provisioning. JBoss and Tomcat were equipped with fine-grained performance monitoring sensors using the ARM API (see [9] for details). We implemented an actuator module to start and stop server instances on different hosts. In addition, we designed a rule-based control module that uses the ARM performance monitoring data in combination with the actuator module to increase or reduce the size of the managed server cluster according to the average response time measured. The control module was able to keep the service response times within a predefined range under different load conditions.

The global stability of the SLM framework depends on constant response times of the participating services. In cases where we observe a high standard deviation from the average response time, workflow managers tend to unneces-

sarily recalculate iSLOs. However this can be minimised by defining appropriate thresholds for workflow managers or alternatively by adding a predefined safety margin to the iSLOs assigned to individual services. In addition we observed that it is essential that service managers pause after executing a reconfiguration in order to allow the changes to take effect. In the example above, the start of a new server instance took about 40s – the manager had to take this into account in order not to trigger the start of additional instances in the meantime.

5 Related work

Many architectures for SLM-enforcement do exist for multi-tier environments [15–17] or single enterprise components [18, 19]. Some of these architectures already employ controllers that are able to manage certain aspects of the system without human interaction (self-management, autonomic management). To give an example, [19] uses a feedback-control approach for autonomic optimisation of Apache Web server response times. In contrast to our architecture these approaches mainly focus on the service layer, i.e., they are not capable of dealing with changing SOA workflows.

[20] discusses the topic of QoS enforcement in Web Services environments and points out current challenges in this area. A management architecture that itself is organised as a SOA is discussed in [21]. The authors describe their SOA-based management approach as a novel way to integrate different management applications but do not provide automatic alignment with changes in the business architecture.[22] presents a method for analysing the effects of service-local SLAs on global business processes. The approach gives hints for future investments (in terms of resources) to improve the overall QoS. It could therefore complement our work as it assists long-term management decisions on business restructuring.

In [23] the authors present WSQoSX, an SLM architecture for SOA environments. WSQoSX consists of a number of management components that control the lifecycle of SOA components, e.g. service discovery, selection, and workflow assembly. The architecture focuses on QoS-dependent service binding, i.e. the management system evaluates workflows and selects participating services based on their response time to fulfil predefined SLOs. When compared to our SLM architecture for SOA management, WSQoSX focuses on scenarios where different services with equivalent functionality are available and does not deal with the possibility of QoS-improvement for individual services. Thus WSQoSX focuses on B2B scenarios where multiple providers offer standardised services to choose from, whereas our approach focuses on SLM in inner-enterprise scenarios. In contrast to our approach, WSQoSX is realised as a number of centralised services, which may eventually lead to scalability issues.

6 Summary and Conclusions

We presented a decentralised management approach for SOA environments that uses emerging self-management techniques to realise a flexible SLM system. The

SLM architecture consists of a modular self-manager framework that provides the basis for component-level and workflow-level managers. The framework provides a number of sensor and effector modules to monitor and manage different classes of enterprise components. The seamless integration with existing SOA components is based on the Service Component Architecture (SCA).

The underlying manager framework has also been used in a different context: we designed an autonomic management framework for virtual machines [24], which is going to be integrated with the work presented in this paper. We are also working on basing our inter-manager SLA communication on WS-Agreement (see [25]). In addition, future work concentrates on exploiting the potential for system-wide optimisations, which are made possible by the homogeneous view on all applications of an enterprise that is provided by a SOA. We currently work on enhancing the existing architecture with self-organisation aspects for service managers. We aim to minimise global service resource usage by establishing a P2P-based trading mechanism for iSLO parts. A first approach that uses auction and bazaar protocols for transferring iSLO shares between participating components of a workflow has already been described in [26].

References

1. Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language Version 2.0 - OASIS Standard. (April 2007) `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf` (Last visited 10/12/2007).
2. Rud, D., Schmietendorf, A., Dumke, R.: Performance Modeling of WS-BPEL-Based Web Service Compositions. IEEE Services Computing Workshop (2006)
3. Open SOA Collaboration: SCA Service Component Architecture – Assembly Model Specification Version 1.0. (March 2007) `http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications` (Last visited 10/12/2007).
4. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. Computer **36**(1) (January 2003) 41–50
5. Herrmann, K., Muehl, G., Geihs, K.: Self-Management: The Solution to Complexity or Just Another Problem? IEEE Distributed Systems Online **6**(1) (2005)
6. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management **11**(1) (Mar 2003) 57–81
7. Schaefer, J.: An Approach for Fine-Grained Web Service Performance Monitoring. In: Distributed Applications and Interoperable Systems : 6th IFIP WG 6.1 Int. Conf., DAIS 2006, Proceedings, IFIP, Springer (June 2006)
8. IBM: IBM Systems Software Information Center: Application instrumentation. `http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp?topic=/ewlminfo/eicaaappinstrument.htm` (Last visited Aug. 2007).
9. Schmid, M., Thoss, M., Termin, T., Kroeger, R.: A Generic Application-Oriented Performance Instrumentation for Multi-Tier Environments. In: IM 2007 - IFIP/IEEE Int. Symp. on Integrated Network Management, IEEE (May 2007)
10. OASIS: Web Services Distributed Management: Management of Web Services 1.0. (2005) `http://docs.oasisopen.org/wsdm/2004/12/wsdm-mows-1.0.pdf`.

11. Distributed Management Task Force, Inc.: Common Information Model Specification 2.2. (1999) `http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf`.
12. Sun Microsystems: Java Management Extensions Instrumentation and Agent Specification, V1.2. (2002) `http://jcp.org/aboutJava/community\-process/final/jsr003/index3.html`.
13. The OpenGroup: Application Response Measurement (ARM) Issue 4.0, V2 - C Binding. (2004) `http://www.opengroup.org/management/arm/`.
14. Debusmann, M., Schmid, M., Kroeger, R.: Model-Driven Self-Management of Legacy Applications. In Kutvonen, L., Alonistioti, N., eds.: 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2005), Athens, Greece, June 2005, IFIP, Springer (June 2005) 56–67
15. Menasce, D.A., Barbara, D., Dodge, R.: Preserving QoS of e-commerce sites through self-tuning: A performance model approach. In: Proceedings of the 3rd ACM Conference on Electronic Commerce, ACM Press (Oct 2001) 224–234
16. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P.: Dynamic provisioning of multi-tier internet applications. In: Proceedings of the 2nd International Conference on Autonomic Computing (ICAC 2005). (Jun 2005)
17. Ranjan, S., Rolia, J., Fu, H., Knightly, E.: QoS-driven server migration for internet data centers. In: Proceedings of the 10th International Workshop on Quality of Service (IWQoS 2002). (May 2002) 3–12
18. Diao, Y., Eskesen, F., Froehlich, S., Hellerstein, J.L., Spainhower, L.F., Surendra, M.: Generic Online Optimization of Multiple configuration Parameters With Application to a Database Server. In: Proceedings of the 14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2003). (Oct 2003)
19. Diao, Y., Gandhi, N., Hellerstein, J.L., Parekh, S., Tilbury, D.M.: Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the Apache Web Server. In: Proceedings of Network Operations and Management 2002 (NOMS). (2002) 219–234
20. Ludwig, H.: Web services QoS: external SLAs and internal policies or: how do we deliver what we promise? In: Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on. (2003)
21. Mayerl, C., Vogel, T., Abeck, S.: SOA-based integration of IT service management applications. In: Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on. (2005)
22. Moura, A., Sauv, J., Jornada, J., Radziuk, E.: A Quantitative Approach to IT Investment Allocation to Improve Business Results. In: Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06). (2006)
23. Berbner, R., Grollius, T., Repp, N., Heckmann, O., Ortner, E., Steinmetz, R.: An approach for the Management of Service-oriented Architecture (SoA) based Application Systems. In: Enterprise Modelling and Information Systems Architectures, Proceedings, 2005. (Oct 2005) 208–221
24. Marinescu, D., Kroeger, R.: Towards a Framework for the Autonomic Management of Virtualization-Based Environments. In: Erstes GI/ITG KuVS Fachgespraech Virtualisierung, Paderborn. (February 2008)
25. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu., M.: Web Services Agreement Specification (WS-Agreement). Open Grid Forum. (2007) GWD-R (Proposed Recommendation).
26. Schmid, M.: Ein Ansatz fuer das Service Level Management in dynamischen Architekturen. In Braun, T., Carle, G., Stiller, B., eds.: KiVS 2007 - Kommunikation in Verteilten Systemen, VDE Verlag (March 2007) 255–266 (in German).