In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

Self-Organisation in the Context of QoS Management in Service Oriented Architectures

M. Schmid
University of Applied Sciences Wiesbaden
Distributed Systems Lab
Kurt-Schumacher-Ring 18
65197 Wiesbaden,
Germany
schmid@informatik.fh-wiesbaden.de

K. Geihs
University of Kassel
Distributed Systems Group
Wilhelmshoeher Allee 73
34121 Kassel,
Germany
geihs@uni-kassel.de

Abstract

This paper describes the need for a self-organising Service Level Management (SLM) in future Service Oriented Architectures (SOA). Due to the complexity of SOA environments, self-management and self-organisation are fundamental requirements for an SLM architecture in this context. The SLM architecture presented consists of two types of autonomic managers. Autonomic managers on the service layer are responsible for enforcing quality of service constraints on a local level. These constraints are broken down from a global goal by an autonomic manager on the business process layer, which is responsible for distributing the quality of service requirements and for controlling the autonomic managers on the service level. We analyse the limitations of the current architecture, by discussing ongoing work concerning manager communication and collaboration based on principles adopted from the domain of organisational theory.

Keywords

SLM, SLA, self-management, self-organisation, SOA

1. Motivation

Looking at the increasing complexity of today's IT applications and IT processes and the overall importance of IT services for business success, an effective service level management is vital for competitive enterprises. Service level management (SLM) focuses on maintaining quality of service (QoS) agreements of IT services that usually comprise a variety of hardware and software components forming a complex and challenging management environment.

Quality of service constraints for SLM are defined in service level agreements (SLAs) [8]. An SLA represents a contract agreed between the two parties involved in a service provisioning scenario: service provider and service customer. SLAs define QoS parameters of the provided service. Typically, an SLA defines SLA parameters that are related to performance and availability, e.g., a threshold or average for the response time of service functions. At runtime these high-level SLA parameters must be continuously monitored in order to detect violations. Obviously, monitoring and controlling the QoS of services requires an appropriate runtime infrastructure that comprises sensors for service parameter values and actuators for influencing the service performance.

In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

Currently, business-critical enterprise application systems are commonly designed as multi-tier client-server applications. These distributed applications are typically implemented on standardised middleware frameworks. Frequent technology changes as well as the increasing application complexity significantly complicate the SLM of today's distributed applications. In addition, becoming an ”adaptive enterprise” is one of the key goals for many corporate organisations. Adaptiveness is needed in order to facilitate the rapid implementation of innovative solutions and products, to reduce the operational cost, to better utilise the existing resources, and to support change and transformation. Permanent restructuring, outsourcing, acquisitions, and a tighter integration of business to business (B2B) processes between different organisations demand increasing flexibility in terms of IT service design. We need IT architectures that are adaptive to new hardware and software infrastructures. Large, monolithic applications with hard-coded dependencies and business logic do not meet these requirements. This situation has sparked interest in building enterprise applications based on independent services and workflow descriptions, leading to a Service Oriented Architecture (SOA).

Service level management of SOA applications is even more challenging than the management of traditional client-server applications. The reasons for that will be discussed in the following sections. Thus it will become clear that innovative management approaches are needed to cope with the challenges. In this paper we argue that self-management and self-organisation in services and applications could be answers to the open questions. In section 2 we discuss the technical structure of SOA environments and their inherent challenges for SLM. Section 3 presents our SLM approach for SOA environments which relies on separation of concerns and self-management techniques. In section 4 we discuss deficiencies of the current approach and present possible solutions.

2. Service Oriented Architecture

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. They inter-operate based on a formal interface definition which is independent of the underlying computing platform and programming language.

Services in a SOA are treated as software components that are composed into workflows. High-level workflow descriptions are interpreted by a workflow engine to invoke the services. Workflows themselves may be established at runtime (e.g., as the result of a service invocation) and may invoke other workflows in the same way they use plain services. In B2B scenarios SOAs span across administrative boundaries of organisations, which makes it very difficult to enforce quality of service parameters.

Today, common technologies for implementing a SOA environment are Web Services based on the Web Services Description Language (WSDL) [18] for the description of service interfaces, SOAP [19] as communication protocol, Universal Description, Discovery and Integration (UDDI) [2] for service lookup, and the Business Process Execution Language (BPEL) [1] for describing business processes and workflows. BPEL is an XML-based workflow description language that defines a number of so-called BPEL activities. These activities represent single steps of a workflow, e.g., synchronous or asynchronous

In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

web service invocations, variable assignment and evaluation, case differentiations, loops, etc. These BPEL activities are divided into simple activities that include service invocations and other straight-forward operations and complex activities which wrap a number of simple activities (e.g. loops).

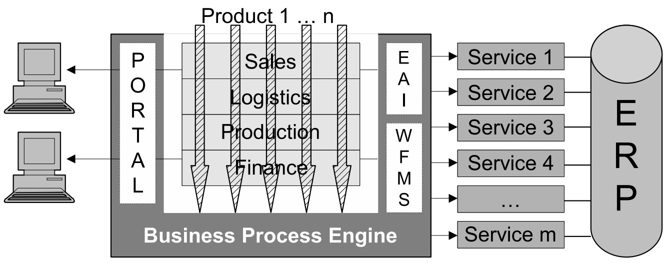

Figure 1: A Business Process Platform according to [16]

Figure 1 shows a business-oriented view of a SOA. Clients access the functionality through a portal, provided by the business process engine. This engine manages the business processes for different departments and products and invokes the relevant services. The services may interact with the company-wide ERP system, however this is transparent to the business process engine.

A design goal for SOA is to bring the architecture of enterprise IT applications in line with the enterprises' organisational structure.

A technical view of a SOA environment is displayed in figure 2. In a SOA services are generally realised by enterprise components which operate on a variety of operational systems. These enterprise components offer service endpoints which may themselves be part of composite services. Invocations of these services are orchestrated in workflows that implement the business processes. Clients can access the environment through a portal. Typically, SOA application scenarios are large-scale, enterprise-wide, and therefore often business critical applications, or large B2B scenarios that span several autonomous enterprises.

Agreeing upon and maintaining a desired Quality of Service is a major concern in service-oriented environments. QoS management becomes even more complex here because there is no methodology available for managing the QoS of workflows in inter-enterprise scenarios taking into account different strategies and policies of the involved actors.

3. Approach for SOA Management

Looking at the technical realisation of a SOA we can distinguish several abstraction layers within the architecture. 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

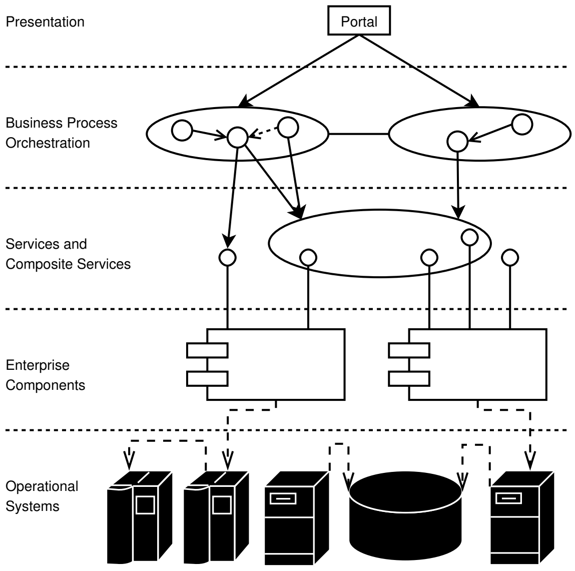
Presentation
Portal
Business Process Orchestration
Services and Composite Services
Enterprise Components
Operational Systems

Figure 2: Technical realisation of a SOA

the typical domain of existing hierarchical SLM-approaches for multi-tier applications. On top of the service layer we can find a third layer in SOA environments: the workflow orchestration layer. The service interfaces hide all implementation details of the underlying systems from the workflows in this layer. Additional complexity may be added as BPEL workflows may themselves be accessed by other workflows like basic or compound services, making complex nested workflows possible.

For several reasons a strictly hierarchical approach is not applicable for establishing SLM within a SOA: (1) A SOA may well spread across enterprise boundaries and therefore also across management domains. (2) Because of the flexible, dynamic and compositional structure of a SOA, traditional static management structures cannot adapt fast enough to the changing management requirements. (3) SOA environments implement large scale processes. Traditional, centralised management approaches with semi-automatic problem solving strategies do not scale sufficiently to meet SOA demands. (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.

Therefore we suggest a two-part SLM architecture for SOAs: One part is concerned with SLM on the service layer, a second part is concerned with SLM on the workflow layer.

As a 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, there is a need to build up a scalable and self-organising management infrastructure that does not rely on human intervention. Recently self-management approaches have become popular, because they aim at reduced management complexity and increased scalability. For that

In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

reason we suggest to apply self-management techniques at least to SLM on the service layer of a SOA environment.

A prerequisite for a successful joint SLM in a SOA environment is the specification of well-defined communication interfaces between the different managers. Managers should be able to hide the internal details of local QoS enforcement to other managers. In addition the management architecture must not make assumptions concerning the usage of certain management algorithms.

Many architectures for SLM-enforcement do exist for multi-tier environments [11, 17, 15] or enterprise components [6, 12]. Some of these architectures already have some self-management capabilities, i.e. controllers that are able to manage certain aspects of the system without human interaction However we do not know of any other approach which aims to enforce QoS constraints across several abstraction layers within a SOA environment.

Section 3.1 presents our QoS-management solution for the workflow layer which interacts with managers on the service layer in order to enforce global QoS constraints. In section 3.2 we present a SLM solution for the service layer. However, for the reasons described above, this solution in principle may be replaced by any other appropriate SLM solution that provides the relevant interfaces. Section 3.3 summarises the architecture of our autonomic manager which was used for prototypically realising the QoS-management approach.

3.1 SLM on the workflow layer

Global QoS demands (for workflows or parts of them) are handled by a manager on the workflow layer. Here, QoS management involves all the services that are composed into a workflow. Workflow-overarching QoS concerns (e.g. total response time, availability) and QoS policies need to be specified and realised for the workflow as a whole by breaking it down to the services on the service layer. As implementation details are encapsulated by the particular service, the autonomic manager on the workflow layer cannot directly enforce QoS demands within a particular service, but has to negotiate QoS requirements with the manager which is responsible for the SLM of the service.

For example, in order to break a global response time requirement down to requirements on the service level, we analysed BPEL activities regarding their potential for response time optimisations. Relevant Basic Activities include `Receive`, `Reply`, and `Invoke`. Relevant Complex Activities are `Switch`, `Pick`, `Flow`, `While`, and `Sequence`. The `Wait`-Activity has to be treated separately, as it consumes a significant, but predictable amount of time.

We propose the following algorithm for distributing the maximum workflow execution time $ET_{max}(P)$ of a QoS-critical section of a workflow P to the activities involved:

We define the initial maximum distributable time $DT(P)$ as $DT(P) = S * ET_{max}(P) - \sum_{i=1}^{k} t_{wait}(i)$, where $t_{wait}(i)$ is the time the process halts for a particular `Wait` to execute. $0 < S < 1$ specifies a safety overhead to be defined by the provider. On the first execution, $DT(P)$ is distributed equally to the following activities: `Receive`, `Reply`, `Invoke`,`Switch`, `Pick`, `Flow`. Statements in a `Sequence` are treated as plain statements, and statements in `While` loops are treated with the weight one (as if the loop will execute exactly once).

For subsequent executions of the business process, the maximum response times of the activities are adjusted as follows:

The average response time for each relevant activity is measured. The algorithm also detects the average number of cycles performed in a while loop. For `Flow` activities, the longest response time is used to build the average (worst case scenario). Then the algorithm corrects the maximum response times, weighing services in `While` loops according to the average number of cycles, thus trying to equally satisfy the services' needs.

In case $ET_{max}(P)$ cannot be met, an event of type $\$event\_overload$ is sent.

The results of the algorithm mainly depend on a deterministic response time of the services involved as well as on an ideally constant number of iterations in loops. Taking a closer look at the kinds of workflows which are subject to SLA negotiations these requirements turn out not to be too restrictive, as service providers would generally only agree on SLAs regarding response times when dealing with either very deterministic processes or a significant amount of time for safety overhead.

The workflow-layer SLM approach has been prototypically implemented using the ActiveBPEL[*] workflow engine and the autonomic manager described in section 3.2. The workflow engine has been instrumented for performance monitoring using the Open Group Application Response Measurement (ARM) [14, 13] API. In our lab environment, the autonomic manager on the workflow layer cooperates with the service-level SLM setting presented in section 3.2. Figure 3 depicts the overall architecture for SLM on the workflow layer.

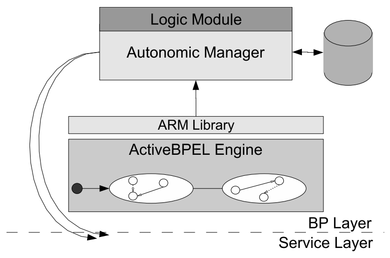

Figure 3: The principal SLM architecture on the business process layer

3.2 SLM on the service layer

In a sample scenario a cluster of JBoss [10] application servers hosts several web services which are part of a SOA scenario. Internally, these web services are implemented by Enterprise JavaBeans (EJB). We assume that the response times of the JBoss instances are independent from each other because they do not share any resources (each instance runs on a different host). We used an autonomic manager to control the number of JBoss server instances in the cluster.

[*]See http://www.activebpel.org

In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

The application server instances are performance instrumented using the Open Group *Application Response Measurement (ARM)* [13, 14] API. The resulting performance measurements are represented as CIM Units of Work in a *CIM Object Manager (CIMOM)* Details concerning the performance instrumentation are described in [5].

Figure 4 depicts the principal management architecture for the service layer. The system under management sends performance measurements to the ARM library, which is connected with a CIM provider. Configuration and performance information is represented in the CIM information model of the CIMOM. The autonomic manager accesses the CIMOM through its CIM/WBEM extension module. Management operations are performed on the CIM model and then carried out by the CIM provider.

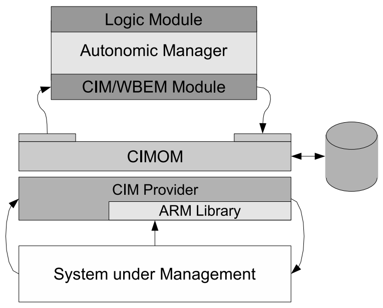

Figure 4: The principal SLM architecture

Figure 5 displays a simplified version of the control algorithm involved. This control algorithm is a formal representation of the management knowledge (originally developed by a human administrator) that is necessary for controlling the system.

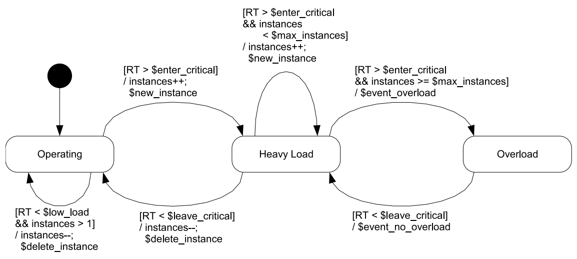
[RT > $enter_critical && instances < $max_instances] / instances++; $new_instance
[RT > $enter_critical] / instances++; $new_instance
[RT > $enter_critical && instances >= $max_instances] / $event_overload
Operating
Heavy Load
Overload
[RT < $low_load && instances > 1] / instances--; $delete_instance
[RT < $leave_critical] / instances--; $delete_instance
[RT < $leave_critical] / $event_no_overload

Figure 5: Control cycle for number of application server instances

The algorithm operates on the average response time RT_C of a service class C. For the example we assume that a server instance provides the services of only one class at a given time t. Initially the algorithm is in state `Operating`. If RT exceeds $\$enter\_critical$ an additional instance is created and the state machine changes to the `Heavy Load` state. While $RT \geq \$enter\_critical$, additional instances are created as long as the total number of instances $instances \leq max\_instances$. If the maximum number of instances

In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

is reached and RT remains greater than $max\_instances$, the system changes to the `Overload` state and an event of type $\$event\_overload$ is generated. If RT decreases below $leave\_critical$ an event of type $\$event\_no\_overload$ is generated and the system changes to state `Heavy Load`. If RT decreases below low_load, the system changes to `Operating`, where unused instances are removed when the load stays below low_load and the number of instances is greater than one.

The maximum allowed response time RT_{max} is a parameter that is totally independent from the service implementation and thus may be easily controlled externally (e.g., by an SLM system on the business layer). The system sets $\$enter\_critical = S * RT_{max}$ to establish a safety buffer (again $0 < S < 1$ specifies a safety overhead to be defined by the provider) and adjusts $leave\_critical$ and low_load accordingly.

3.3 Autonomic manager

The Distributed Systems Lab developed a modular autonomic manager that provides a customisable basis for the managers on the workflow and the service layer. The modular autonomic manager is based on a management framework [3] which had originally been designed for the management of CORBA-based applications.

Internally, the autonomic manager consists of four main components: The central element of the autonomic manager is the `module manager`, which provides `adapters` to connect extension modules. These modules implement sensors, actuators, and the internal logic of the autonomic controller. A `messaging subsystem` that is part of the module manager is responsible for the message handling within the system.

The autonomic manager supports three kinds of extension modules: `event modules`, `action modules`, and `logic modules`. `Event modules` are modules that possess their own threads and thus are able to react actively to changes within the environment by creating internal messages. `Action modules` are passive; they act - triggered by internal messages - by analysing application-specific sensors, or performing management tasks.

Sensors may be realised using either `event modules` (push model) or `action modules` (pull model). Application-specific actuators are realised through `action modules`.

The modules implement an application-class specific interface, e.g. for Web-based Enterprise Management (WBEM), or Simple Network Management Protocol (SNMP) instrumented applications.

`Logic modules` form the “brain” of the autonomic manager. Currently, finite state machines and an XML-based policy language are supported, however an extension to neural networks and other control-mechanisms is possible because of the modular structure of the autonomic manager. `Logic modules` act periodically or are triggered by incoming messages. Management decisions are communicated to other modules using the internal messaging capabilities.

Several `action` and `event` modules have been implemented:

- A log-analysis module is able to parse system log files.
- Shell-scripting supports the execution of UNIX shell scripts.
- A generic CORBA management interface handles generic CORBA invocations.
- A Web Services module handles generic Web Services invocations.

In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

- *Simple Network Management Protocol (SNMP)* modules can talk to SNMP agents using `GetRequest` and `SetRequest`. In addition SNMP traps are handled by an `event` module.
- The *Web based Enterprise Management (WBEM)*/*Common Information Model (CIM)* [7] module allows the autonomic manager to act as a CIM client.

Inter-manager communication is realised by web services or CORBA modules.

Details concerning a method for formalising management knowledge for the autonomic manager can be found in [4].

4. Current Challenges

The presented design of our SLM architecture comprises two coupled control cycles for defining and enforcing response times in different layers of a SOA. The architecture is not yet capable to handle concurrent workflows with different priorities and to enforce or negotiate QoS parameters with services that are located in different administration domains. In general, these are challenges regarding interaction of entities and overall management organisation.

4.1 Self-Organisation - a possible solution?

Self-organisation in physical, chemical and biological systems may be described as a spontaneous process that leads to the formation of diverse kinds of structures on a macroscopic level. Self-organising systems regularly show emergent properties that characteristically are unpredictable and irreducible. They are not limited to physical, chemical and biological systems, but self-organisation may also be discovered when looking at e.g. social and economical systems.

Self-organising systems are classified as autonomous regarding certain criteria and as non-linear in terms of system-behaviour. We often find a high degree of redundancy in these kinds of systems.

When aiming at utilising self-organisation principles in the area of technical systems, unpredictability is a major challenge. In addition, emerging behaviour often solely occurs when reaching a “critical mass” of system elements, which complicates testing of these kinds of systems. In technical systems, it is generally desirable to implement the concept of “guided emergence”, where system elements show a certain degree of self-organisation and therefore emerging behaviour, but are still controllable on a macroscopic level. An important design issue is also the fact that self-organising systems usually possess a great redundancy, whereas technical systems usually are built of a set of individually specialised components.

Organisational theory concentrates on the structuring of organisations and therefore also deals with self-organisation between individuals. Therefore similar challenges can be discovered in this field as organisational theory also tries to guide emergent behaviour into predefined structures. The structure of an organisation is complemented by the definition of ways of communication between individuals. Permanent communication is necessary because knowledge is essentially distributed within an organisation.

Essential elements of organisational structuring are the principles of delegation and participation. Many theories rely on market mechanisms (rewards, egoism) or hierarchies

In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

as the driving principles for motivating individuals to participate and to support the organisational aims.

4.2 Application of self-organisation principles to SLM

When adapting principles observed in the real world to the design of technical systems, we try to reduce the complexity of the original system to the concepts which are considered important for the target system. From our point of view the main challenge is to cope with the paradox that on the one hand we want a certain emergent system behaviour while on the other hand the definition of emergence says that it arises somewhat unexpectedly and thus cannot be engineered into a system in a systematic way. Our research is looking for an answer to the question whether it is possible to create emergent system behaviour while embedding self-organising sub-structures into a global management architecture thus creating a system which still behaves deterministic on the macroscopic level. Our experiences with the autonomic manager framework look promising for the simple scenario implemented so far. Future work will concentrate on inter-manager communication in order to meet the challenges described above.

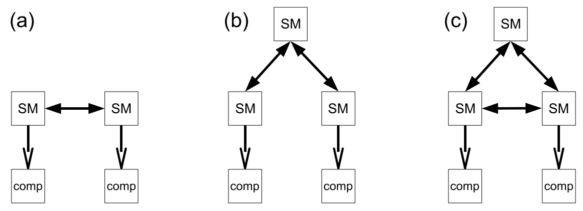

Figure 6: Possible communication schemes between managers

Figure 6 depicts possible communication schemes for managers (SM) while controlling application components (comp). Type (a) depicts a horizontal communication between managers. Type (b) depicts vertical communication between managers. Type (c) depicts a possible form of mixture between (a) and (b).

Vertical (or hierarchical) approaches are easier to implement, but they do not necessarily scale as good as horizontal approaches. In addition, it is a mandatory condition for a hierarchical management approach that it is possible to concentrate the required management knowledge and decision-making authority in one place - a condition which e.g. cannot be fulfilled in an environment that spans across organisational boundaries.

Horizontal approaches could use group-communication and voting mechanisms to form a common strategy. They are more complicated to realise, but do not require a single node to be able to access all relevant management knowledge.

The definition of fundamentals for a flexible, at least partially self-organising management cooperation architecture is ongoing work.

We consider market mechanisms and egoism as principles which might be used to motivate interaction in a self-organising management environment. Managers could be forced to ”pay” a certain price in order to motivate others to support their individual aims. At the same time managers would be interested to increase their wealth while still performing the tasks they are assigned to. This would motivate them to support the manager with the highest bid. Managers could also make use of Time/Utility functions [9] to remunerate other managers which behave in a helpful way. A precondition for this is however that managers must be able to estimate the overall performance of the management system (e.g., by comparing costs and benefit) and to optimise their behaviour accordingly.

Currently, our goal is to design a flexible SLM communication architecture that meets the coordination challenges presented by steadily reorganising and optimising itself, following the approaches we described. Interactions between managers are not predefined, in fact services communicate horizontally as well as vertically to fulfil the QoS requirements defined. The emerging communication infrastructure constantly changes while trying to optimise itself. This clearly distincts the approach from existing management architectures and results in increased flexibility regarding changes in application business logic.

5. Summary

In this paper, we presented an autonomic SLM architecture for SOAs. The central component of the architecture is an autonomic manager, which may be easily customised using extension modules that implement sensors, actuators and management logic. Autonomic managers may interact remotely using special communication modules.

The SLM architecture consists of an autonomic manager on the service layer, which is responsible for service reconfiguration in order to meet requirements that are set externally. The autonomic manager accesses the system under management with the help of a CIMOM, which provides a CIM representation of relevant configuration information within the system as well as performance measurements.

On the business process layer, an autonomic manager controls the overall business process response time by analysing performance measurements of relevant steps within the workflow. The autonomic manager communicates the resulting performance goals to the SLM on the service layer, and controls their compliance.

We showed that the current interaction paradigms are not flexible enough to apply this approach to more complex scenarios with concurrent business processes and complicated service provisioning scenarios.

Thus, we presented ongoing work concerning inter-manager communication and cooperation, which might lead to a solution for the problem. This cooperation approach is partially based on ideas from organisational theory.

References

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, S. Thatte, I. Trickovic, S. Weerawarana, K. Liu, D. Roller, and D. Smith. *Business Process Execution Language for Web Services, Version 1.1*. IBM Developer Works, May 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

[2] T. Bellwood. *UDDI Version 2.04 API Specification*. UDDI Commitee, 2002. http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm.

[3] M. Debusmann and R. Kroeger. Widening Traditional Management Platforms for Managing CORBA Applications. In *Proceedings of the third IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS)*, Sep 2001.

In: Proceedings of the 13th Annual Workshop of HP OpenView University Association, Hosted by the University of Nice at Cote d'Azur May 21 - 24, 2006

[4] M. Debusmann, M. Schmid, and R. Kroeger. Model-Driven Self-Management of Legacy Applications. In *Proceedings of the fifth IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS)*, Jun 2005.

[5] Markus Debusmann, Marc Schmidt, Markus Schmid, and Reinhold Kroeger. Unified Service Level Monitoring using CIM. In *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, Sep 2003.

[6] Y. Diao, F. Eskesen, S. Froehlich, J. L. Hellerstein, L. F. Spainhower, and M. Surendra. Generic Online Optimization of Multiple configuration Parameters With Application to a Database Server. In *Proceedings of the fourteenth IFIP/IEEE Workshop on Distributed systems: Operations and Management (DSOM 2003)*, Oct 2003.

[7] Distributed Management Task Force, Inc. *COMMON INFORMATION MODEL (CIM) SPECIFICATION*, 2.2 edition, 1999. http://www.dmtf.org/standards/documents/CIM/DSP0004.pdf.

[8] Lundy Lewis. *Managing Business and Service Networks*. Kluwer Academic / Plenum Publishers, 2001.

[9] Peng Li, Binoy Ravindran, and E. Douglas Jensen. Adaptive Time-Critical Resource Management Using Time/Utility Functions: Past, Present, and Future. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, 2004.

[10] Miko Matsumura. JBoss Application Server – Standards based Infrastructure for the Enterprise. White Paper, 2005. http://www.jboss.com/pdf/JBossAS-EnterpriseInfrastructure.pdf.

[11] D. A. Menasce, D. Barbara, and R. Dodge. Preserving QoS of e-commerce sites through self-tuning: A performance model approach. In *Proceedings of the 3rd ACM Conference on Electronic Commerce (EC'01)*, pages 224–234. ACM, ACM Press, Oct 2001.

[12] Y. Diao nad N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury. 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)*, pages 219–234, 2002.

[13] The Open Group. *Systems Management: Application Response Measurement (ARM)*, 1998. Open Group Technical Standard, Document Number: C807.

[14] The OpenGroup. *Application Response Measurement (ARM) Issue 4.0, V2 - C Binding*, 2004. http://www.opengroup.org/management/arm/.

[15] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. QoS-driven Server Migration for Internet Data Centers. In *Proceedings of the 10th International Workshop on Quality of Service (IWQoS 2002)*, pages 3–12, May 2002.

[16] A. W. Scheer, O. Thomas, C. Seel, G. Martin, and B. Kaffai. Geschaeftsprozessorientierte softwarearchitekturen: Revolution auf dem software-markt? In P. Dadam and Manfred Reichert, editors, *Proceedings of 34. Jahrestagung der Gesellschaft fuer Informatik*, volume 1 of *Lecture Notes in Informatics*, pages 2 – 13, 2004. (in German).

[17] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic Provisioning of Multi-tier Internet Applications. In *Proceedings of the 2nd International Conference on Autonomic Computing (ICAC 2005)*, Jun 2005.

[18] World Wide Web Consortium. *Web Services Description Language (WSDL) 1.1*, Mar 2001. W3C Working Group Note, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[19] World Wide Web Consortium. *SOAP Version 1.2*, Jun 2003. http://www.w3.org/TR/soap12.