Adaptive SOA Solution Stack

Krzysztof Zieliński, *Member*, *IEEE*, Tomasz Szydło, Robert Szymacha, Jacek Kosiński, Joanna Kosińska, and Marcin Jarzab

Abstract—This paper presents the concept of an Adaptive SOA Solution Stack (AS3). It is an extension of the S3 model, implemented via uniform application of the AS3 element pattern across different layers of the model. The pattern consists of components constituting an adaptation loop. The functionality of each component is specified in a generic way. Aspects of these patterns are analyzed in relation to individual S3 layers. The ability to achieve multilayer adaptation, provided by several cooperating AS3 elements is also discussed. Practical usage of the proposed concepts for Adaptive Operational Systems, Integration, and Service Component layers are presented in the form of three case studies. Each study describes the architecture of the proposed system extensions, selected software technologies, implementation details, and sample applications. Related work is discussed in order to provide a background for the reported research. This paper ends with conclusions and an outline of future work.

Index Terms—Services architectures, services management, operational model, quality of services.

1 Introduction

Over the last several years adaptive systems have attracted significant attention. Development of these systems is driven by the emergence of new software technologies enabling dynamic and reconfigurable software development, such as Dynamic AOP [1], OSGi [2], and virtualization technologies, e.g., Xen [3], VMware [4], and Solaris Containers [5], offering runtime control of computer resources.

Adaptive systems focus on the development of systems that modify their structure and behavior in response to changes in the execution environment [6]. These systems monitor state, correlate information and perform actions according to user-defined policies. They are considered a simpler form of Autonomic Computing [7], [8], [9] systems, capable of performing self-management on their own, based on knowledge collected during operation.

Adaptive systems ameliorate the complexity of computing systems, expressed not only by the number of connected hardware and software components but also by the growing space of configuration parameters and management strategies offered through middleware and virtualized computational platforms. Better utilization of modern IT infrastructures is a prerequisite for achieving the required Quality of Service (QoS) and end-user satisfaction, characterized by Quality of Experience (QoE).

All these aspects can be considered in the context of Service Oriented Architecture (SOA) [10], [11]—the most popular paradigm for implementing enterprise software systems. SOA enables the development of applications by combining loosely coupled and interoperable services. The complexity of this process and the need to guarantee QoE and QoS make adaptive systems especially suitable for SOA deployment.

1.1 Adaptive SOA Research Issue

SOA application development, deployment and execution requirements may be analyzed in the context of the SOA Solution Stack (S3) [11] proposed by IBM. It is a highly generic model, taking into account all presented aspects, with particular attention to SOA policy-driven governance. This justifies its selection as a starting point for research on integrated SOA adaptive systems.

Research addressing adaptive SOA applications [12], [13], [14] concentrates mainly on the application level. Adaptability aspects cover SOA application deployment [15], dynamic selection of services [13], and dynamic service composition [16], [14]. The key observation is that S3 layers are not independent of one another and decisions made on a given layer may influence the behavior of other layers. Any parameter set by lower layers can be considered as a constraint for the adaptability strategies implemented on higher layers. This is a well-known problem in hierarchical system optimization. Its practical solution in the context of SOA systems requires not only adaptation strategies capable of being applied across several layers of the system, but also a uniform approach to extending each layer with adaptability mechanisms. This distinguishes the proposed approach from other studies focusing on selected layers, such as adaptive ESB [17] or Middleware [18].

In the context of existing work on adaptive SOA, there are still some critical issues that have not been well explored.

- Construction of a uniform adaptive system model which could be applied to any layer of SOA-based computing systems including virtualized computational resources. Such a model should include basic components with well-defined interfaces, suitable for constructing an adaptation loop.
- Selection of a software implementation technology that can be applied across S3 layers, providing interoperability of adaptability extensions. The same category of adaptability components from different

- *The authors are with the Department of Computer Science, AGH—University of Science and Technology, Al. Mickiewicza 30, Kraków 30-059, Poland. E-mail: {kz, tszydlo, szymacha, jgk, kosinska, mj}@agh.edu.pl.*

Manuscript received 5 Apr. 2010; revised 29 July 2010; accepted 28 Jan. 2011; published online 7 Feb. 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org and reference IEEECS Log Number TSCSI-2010-04-0038.
Digital Object Identifier no. 10.1109/TSC.2011.8.

layers should be easy to integrate and use in adaptation loops spanning more than one layer.

- The adaptation strategy must process large amounts of data collected during SOA application runtime. Thus, efficiency and scalability issues require particular attention. This calls for research on selective and dynamic instrumentation of the system to add monitoring and management functionality. The collected information (facts) should be processed efficiently by decision engines implementing the adaptability strategy.
- SOA systems are characterized by high levels of elasticity and dynamicity as runtime composition is one of the most important features of these systems. The same aspect refers to adaptability extensions which may be reconfigured on demand, without restarting or suspending the system. This requires further research on dynamic and reconfigurable software development.
- The proposed framework is characterized by agnostic adaptation strategy selection. In spite of this, adoption of probability and statistical theories for predicting the QoS of computing resources and services could play an important practical role. Recently, runtime utilization models [15] for SOA systems have emerged as a focus of intensive research.

1.2 Main Contribution

The main research contribution of this study is an integrated approach to adaptive SOA system development. The integrated approach means that adaptability aspects are introduced in a uniform way to each layer of the S3 model. This leads to a multilayer adaptive system which offers full control over QoS and QoE parameters, taking advantage of modern software and hardware technologies. The proposed approach has the following specific features:

- A pattern-based approach to adaptive extensions. The extension of each S3 layer follows the same proposed pattern consisting of typical components which, together, create an adaptation loop. This pattern focuses on policy-driven management systems and is similar to the Autonomic Computing model [19].
- A flexible approach to adaptation loop component selection. Components can be implemented using existing software packages such as Event Processors or Rule Engines, wrapped with suitable interfaces ensuring interoperability. The OSGi [2] technology is used for this purpose—therefore, components can be deployed as bundles.
- A practical presentation of the proposed concepts, applied to several layers of the S3 model. It provides guidance for system developers on how to construct adaptive loops for different layers. The key point is that the same pattern is used in a uniform way for various layers, such as virtualized computer resource management and adaptive ESB deployment.
- Assessments and evaluation of the proposed solutions. The presented concept is verified by implementation of the Adaptive Operational System layer, Adaptive Integration layer, and Adaptive Service Components layer of the S3 model.

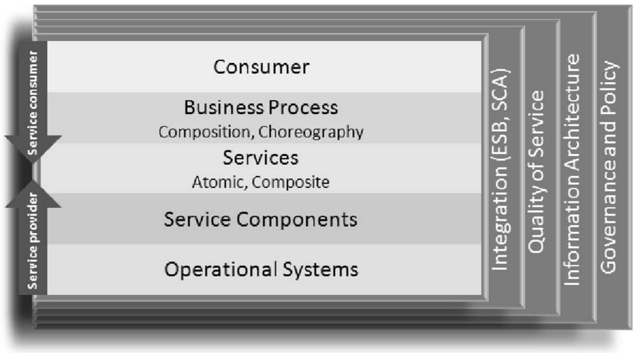

Fig. 1. SOA solution stack model.

This paper is organized as follows. Section 2 introduces the work by presenting and analyzing the S3 model and stating the requirements of adaptive extensions for each layer. Section 3 presents the concept of Adaptive S3 (AS3) elements—a compound pattern for adaptive loop construction. In Section 4, the Adaptive S3 model is described. Section 5 presents the Adaptive Operational Systems layer built with Xen Virtual Machines and Solaris Containers as an initial case study of the proposed model. Adaptive ESB and Adaptive SCA are presented in Sections 6 and 7, respectively. Section 8 provides an overview of related work. Finally, Section 9 concludes this paper and outlines future work.

2 Motivation

SOA application development and deployment should be considered in the context of the SOA Solution Stack proposed by IBM [11], which provides a detailed architectural definition of SOA split into nine layers. This model is depicted in Fig. 1. Each layer has a logical and physical aspect. The logical aspect includes all the architectural building blocks, design decisions, options, key performance indicators, and so on. The physical aspect covers the applicability of each logical aspect in reference to specific technologies and products and is out of scope of our analysis. The S3 model is based on two general assumptions.

1. The existence of a set of service requirements that are both functional and nonfunctional and collectively establish the SOA objective. Nonfunctional service aspects include security, availability, reliability, manageability, scalability, latency, and the like.
2. A single layer or some combination of layers can fulfill specific service requirements and, for each layer, service requirements are satisfied by a specific mechanism.

Adaptability can be described as an activity whose goal is to guarantee the required level of nonfunctional parameters specified by QoE or QoS for each layer.

The nine layers of the S3 stack are as follows: Operational Systems, Service Components, Services, Business Process, Consumer, Integration, QoS, Information Architecture, and Governance and Policy. A brief description of each layer is presented below as a background for further discussions in this section. Only the Consumer layer is not described, as it constitutes a topmost, nontechnical layer wherein the global effects of system activity are observed and evaluated by end users.

Operational systems. This layer includes all application and hardware assets running in an IT operating environment that supports business activities (whether custom, semicustom or off-the-shelf). As this layer consists of existing application software systems, SOA solutions may leverage existing IT assets. Currently, this layer typically includes a virtualized IT infrastructure that results in improved resource manageability and utilization. This property could be effectively exploited in the development of an adaptive virtualized infrastructure, guaranteeing the required level of computational or communication resources accessibility.

Service components. This layer contains software components, each of which is an incarnation of a service or service operation. Service components reflect both the functionality and QoS for each service they represent. Each service component

- provides an enforcement point for ensuring QoS and service-level agreements,
- flexibly supports the composition and layering of IT services, and
- conceals low-level implementation details from consumers.

In effect, the service component layer ensures proper alignment of IT implementations with service descriptions. Service QoS depends on the efficiency of internal components used for service provisioning. It opens a space for adaptability within the Service Component layer. The observed service QoS is not only the result of Service Component activity but also depends on computational resources used in execution. This behavior illustrates the role of the Operational Systems layer and facilitates multi-layer adaptability.

Services. This layer consists of all services defined within SOA. In the broadest sense, services are what providers offer and what consumers or service requesters use. In S3, however, a service is defined as an abstract specification of one or more business-aligned IT functions. This specification provides consumers with sufficient information to be able to invoke the business functions exposed by a service provider. It is necessary to point out that services are implemented by assembling components exposed by the Service Component layer and that this assembly process might be performed dynamically with the support of adaptability mechanisms.

Business process. In this layer, the organization assembles the services exposed in the Services layer into composite services that are analogous to key business processes. In the non-SOA world, business processes exist as custom applications. In contrast, SOA supports application construction by introducing a composite service which orchestrates information flow among a set of services and human actors. Again, these composite services can be constructed dynamically, according to a specific adaptation policy.

Integration. This layer integrates layers 2 through 4. Its integration capabilities, supported by ESB, enable mediation, routing and transporting service requests from the client to the correct service provider. This layer is particularly well

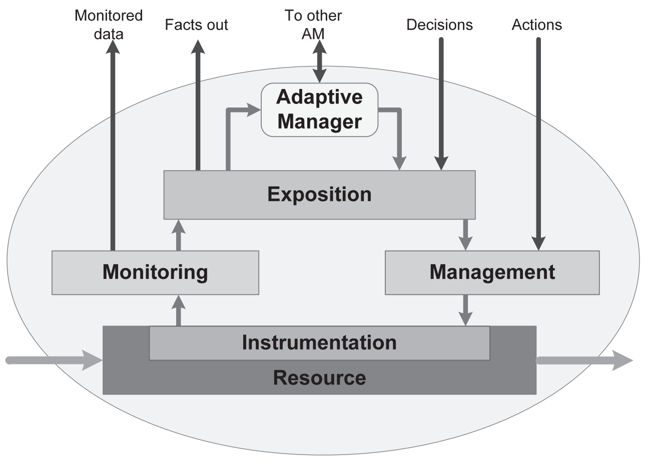

Fig. 2. AS3 element model.

suited for adaptability mechanisms, which is further illustrated by the discussion of Adaptive ESB.

Quality of service. Certain characteristics of SOA may exacerbate well-known IT QoS concerns: increased virtualization, loose coupling, composition of federated services, heterogeneous computing infrastructures, decentralized service-level agreements, the need to aggregate IT QoS metrics to produce business metrics and so on. As a result, SOA clearly requires suitable QoS governance mechanisms.

Information architecture. This layer covers key data and information-related issues involved in developing business intelligence with the use of data marts and warehouses. It includes stored metadata, required for correct interpretation of actual business information.

Governance and policy. This layer covers all aspects of managing the business operations' lifecycle. This layer includes all policies, from manual governance to autonomous policy enforcement. It provides guidance and policies for managing service-level agreements, including capacity, performance, security, and monitoring. As such, the Governance and Policy layer can be superimposed onto all other S3 layers. From a QoS and performance standpoint it is tightly connected to the QoS layer. The layer's governance framework includes service-level agreements based on QoS and key process indicators, a set of capacity planning and performance management policies to design and fine-tune SOA solutions as well as specific security-enabling guidelines for composite applications.

It is evident that the final three layers are directly related to adaptability. They provide key mechanisms required by the adaptation loop, thereby affecting the first five layers listed above.

This observation constitutes our motivation for the presented work on Adaptive S3.

3 Concept of Adaptive S3 Element

The concept of the proposed Adaptive S3 element is presented in Fig. 2 and refers to a single S3 layer. S3 layer is defined as a set of components, such as architectural building blocks, architectural decisions and interactions among components and layers. This definition emphasizes the existence of many options that can be subjected to architectural decisions taken during the SOA application design phase or postponed until runtime. The structure of the AS3 element follows the typical control loop construction patterns and consists of the components listed below.

The **Managed Resource** is instrumented with sensors and effectors. The term Resource is used here in an abstract way, meaning anything (e.g., virtual machine, application server or ESB) that can be monitored and managed. The instrumentation process equips the Resource with the ability to expose its state and data required to characterize its activity. The same process also installs mechanisms for changing Resource parameters or configuration, according to decisions imposed by the control loop. The key point is that instrumentation should be nonintrusive, selective, and dynamic (performed during runtime).

Data communicated by sensors is collected by the **Monitoring Component**. This component is responsible for calculating selected metrics and processing events. Simple events may be assembled into complex ones, important for the adaptation strategy. Complex Event Processors such as Esper [20], or Drools [21] can be used to perform this activity in a scalable and efficient way.

The aggregated data—the output of the Monitoring Component—is forwarded to the **Exposition Component**. The exposition process is related to the Adaptive Manager (AM) used for selection of control actions. It transforms monitored data into a format used by given manager representations. Therefore, the Exposition Component plays the role of a harmonization layer.

To enact the adaptation strategy, an **Adaptive Manager** is used. It may be based on the standard theory of regulation, fuzzy logic or neural networks. In the proposed study, policy-based management has been chosen. Such a solution is suitable when a decision has to be based on many different parameters and a precise mathematical model of the managed system is not available. The adaptation actions are selected by the Policy Engine (PE). Rule Engines such as Jess [22] or Drools [21] can be used in scalable implementations.

The action selected by the Adaptive Manager is converted by the Exposition Component to a format acceptable by the **Management Component**. This component enforces management actions using effectors which instrument the Managed Resource.

Adaptive Service layer of composite services can be constructed dynamically, akin to adaptive services, which are built from components in the adaptive Service Components layer. The same applies to the adaptive Business Process layer where processes are constructed from services. These three layers are conceptually very similar. The abstraction level (granularity) is, however, significantly greater in the Service and Business Process layer, as explained by ongoing studies [13], [14], [16]. Thus, the pattern specified by the proposed AS3 element could be applied here. The Business Process layer activity concerns mainly the choreography and orchestration processes which may be performed with the aid of the adaptive Integration layer, investigated in more detail in Section 6.

The presented AS3 element constitutes a standard adaptation loop. The activity of this loop could be event-driven or performed at regular intervals. It could act autonomously or in cooperation with other similar elements. For this purpose the AS3 element sends out monitored data and facts, and is able to receive decisions and actions taken by other AS3 elements. Adaptive Managers from different AS3 elements are also able to cooperate with one another. This feature makes AS3 suitable not only for single-layer adaptation but also for making global decisions which affect many layers. This issue will be further explained in Section 4.

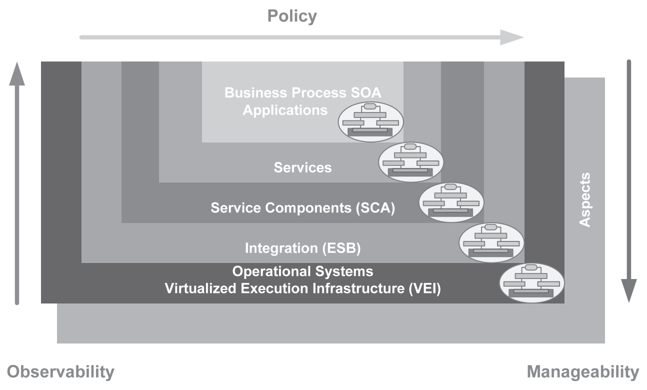

Fig. 3. Adaptive SOA solution stack.

4 Adaptive S3 Stack

The proposed AS3 element can be applied in a uniform way across the S3 stack layers described in Section 2. This leads to the Adaptive SOA Solution Stack, depicted in Fig. 3. Every layer of this stack is marked with a symbol of the AS3 element. This stack is presented in a slightly modified way comparing to the original S3 model as the Integration layer is placed directly above the Operational Systems layer. In the standard S3 model the Integration layer is vertical, to signify its importance for each horizontal layer. Fig. 3 assumes that every higher layer relies on the functionality of the lower layers. This is why the Integration layer is placed just above the Operational Systems layer.

The Operational Systems layer is usually represented as a Virtualized Execution Infrastructure (VEI). Computational and communication resources are virtualized, which results not only in better hardware utilization but also higher manageability. For instance, a Xen Virtual Machine or a Solaris Container with CPU and memory usage guarantees could be allocated to process a single service instance with a required QoS level. Computational resource allocation might be dynamically changed according to the observed load and decisions performed by the adaptation strategy. This leads us directly to the AS3 element concept.

The same considerations apply, in a natural way, to other layers of the AS3 stack, giving rise to an adaptive Integration layer, Service Components layer, etc.

Adaptation abilities and metrics are discussed extensively in the literature [23]. The definition of an adaptation strategy as well as its associated parameters, e.g., threshold points, etc., is always very much domain-specific and must be analyzed in the context of a given application domain, along with a preliminary experimental study. This is why the AS3 model is presented as a framework which needs further refinement. In the presented case studies, data are

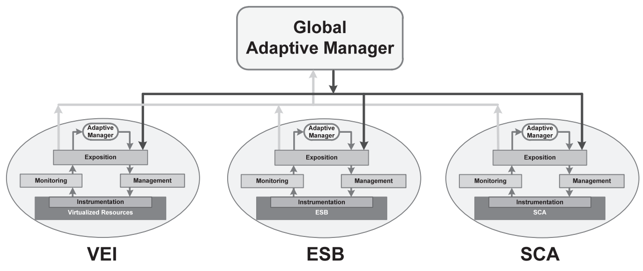

Fig. 4. AS3 model case study.

collected from the experimental implementation and not from simulation experiments. With large data sets, the efficiency of the AS3 stack could be a significant issue. This is why the following measures have been taken.

- Each AS3 layer is controlled independently, with dedicated AS3 elements which may be connected in a hierarchical structure. This concept is similar to the one used by IBM [19] and Motorola [24] in their Autonomic Elements cooperation scheme to control complex systems.
- Each AS3 element is able to communicate monitored data and facts to other elements. This communication can be performed very selectively to reduce the amount of data which needs to be processed. Event processors can be used for this purpose [20]—a standard solution used by modern event-driven systems.
- Adaptive Managers activity could be also organized into hierarchical systems. Local decisions could be coordinated by higher level policies. If Rule Engines [21], [22] are used for the implementation of the Adaptive Manager module, this approach is fully supported. Additionally, the RETE algorithm used by Rule Engines offers high scalability even with a single Rule Engine.

The full potential of the proposed approach is manifested in the cooperation abilities of AS3 components representing different layers. For instance, it is possible to monitor the whole AS3 stack by collecting information from the Monitoring Component of every deployed AS3 element. This reasoning may also refer to other types of AS3 components—thus, we can identify the following important aspects:

1. *Observability*. The monitoring data and facts are propagated from lower layers to higher ones. This could be achieved by connecting Monitoring and Exposition Components from different AS3 elements.
2. *Manageability*. Management actions are sent between Management Components and can be transformed by Exposition Components.
3. *Policy*. This aspect refers to cooperation between PE components. A hierarchical connection scheme seems to be the most natural solution.

Together, these three aspects are illustrated as a vertical layer which cuts across all layers of the proposed AS3 model.

A key implementation issue concerns interoperability of components belonging to different AS3 elements. This issue can be resolved by applying a common software technology for all AS3 elements—something that will be discussed in more detail in subsequent sections.

TABLE 1
AS3 Layers Implementation Examples

AS3 layer	Implementation	Managed Resource	Adaptability Decision
Service Components	Adaptive SCA	Composite	Changing of Composition
Integration	Adaptive ESB	Complex Service Choreography	Modification of Message Routing
Operational Systems	Virtual Execution Infrastructure	Virtual Appliances, VM, Containers	Changing of OS Resources Allocation

The AS3 model presents a reference architecture which is technology-agnostic. While full implementation of this architecture would require significant effort, a good starting point is the construction of adaptive layers. The next step would be to consider the cross-layer adaptation processes. An example of an AS3 model implementation is presented in Fig. 4. In contrast to the AS3 model itself, the implementation depends on a specific technology—JMX [25], Xen, Solaris Containers, ESB, SCA, etc. This is summarized in Table 1. The presented case studies should be considered as proofs of concept. Software technology coherency ensures natural interoperability of individual components. In Fig. 4, this is illustrated by connections between Exposition Components and the Global Adaptive Manager. Any matching AS3 input and output elements could be connected in a similar way.

5 Adaptive Operational Systems Layer

Modern execution environments for SOA operate over virtualized execution resources deployed in a physical infrastructure. This leads to the Operational Systems layer with three basic sublayers: physical, virtualization, and infrastructure, which together constitute the execution infrastructure for SOA applications.

Each of these sublayers can be further structured with service orientation principles in mind, resulting in a so-called Service Oriented Infrastructure (SOI) [26]. SOI enables moving from dedicated infrastructures for specific applications to an architecture in which IT resources and infrastructures' system tools are exposed as services, being defined in resource pools allocated on demand using virtualization techniques.

Adaptive management of SOI involves open mechanisms for monitoring and management of infrastructure resources and services. In general, we can distinguish at least two types of resources whose parameters must be subject to monitoring: 1) Middleware, which is an application container for given SOA services, and 2) Virtual Execution Infrastructure, in which application containers operate. Monitoring processes apply to such parameters as actual resource usage, QoS attributes (if SLA contracts are maintained) and information about service availability. Their principal task is to collect comprehensive information about the history and state of services necessary during planning and realization of specific adaptation strategies.

When analyzing the relationship between SOI and the S3 Model Operational Systems layer, which includes all application assets running in a virtualized IT infrastructure, is the main point of interest. SOI tools should enable effective provisioning of databases, transaction processing systems, middleware (e.g., J2EE or .NET) and SOA services running in a virtualized infrastructure. In fact, the SOI architecture tends to deliver services that can be described as a Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

Currently many virtualization technologies exist based on domains [27], virtual machines [28], and OS containers [29]. When comparing domains to virtual machines, they can be classified as hardware or firmware functions that provide stronger isolation compared to virtual machines. Both enable server virtualization and subdivide physical resources (CPU, RAM, network), but domains typically have lower overhead. A disadvantage of domains is the fixed number of hosted OS instances. Virtual machines are also managed by a hypervisor which itself is an operating system implemented in software, without fixed limits on the number of running OS instances. OS containers can run in VMs, but have minimal overhead and their isolation is accomplished by restricting the scope of system calls to the container from which they are invoked. Thus, there is no need for CPU-intensive emulation performed in hypervisors.

Domains and virtual machines host OS instances that can also be virtualized through OS virtualization techniques and so-called multilevel virtualization. This approach is very flexible in terms of computational isolation and level of resource management, but it also introduces complexity in terms of monitoring and management.

Despite the fact that virtualization is a very effective mechanism for consolidation of workloads and provides means for assigning resource consumption boundaries, finding appropriate resource attributes values that guarantee a given service level is not a trivial task. Considering such factors, one comes to the conclusion that there is a need for a workload monitoring and modeling tool that enables monitoring and planning system performance in virtualized environments. This type of software platform should automate the management of virtualized resources and provision of information about running workloads in a given service center.

5.1 Architecture

The architecture based on the SOI concept assumes a mapping between a typical IT infrastructure and a set of services. Such services are usually based on shared virtualized resources (servers, communication infrastructure, data storage, and applications). To ensure flexible mechanisms for creation of SOA environments that exploit the SOI architecture concept, suitable SOA Virtualized Infrastructure mechanisms for regulating resource access should exist. These mechanisms are called Resource Management Systems (RMSs) and are particularly important in situations where resources are shared.

The usage of virtualization techniques is a natural evolution of existing RMS in distributed environments. Hence, the solution presented in this section can be treated as the answer to the question whether the virtualization offered by available technologies is mature enough to develop virtual computers and virtual communication infrastructures to effectively organize processing in distributed environments. A key point to address is whether efficient management of on demand resources with virtualization techniques is possible using such mechanisms as dynamic infrastructure instantiation and resource allocation according to the requirements of applications or changing environmental conditions. These mechanisms should be able to fully utilize the available infrastructure and decrease the operational costs through resource usage reduction, with no impact on the service QoS level.

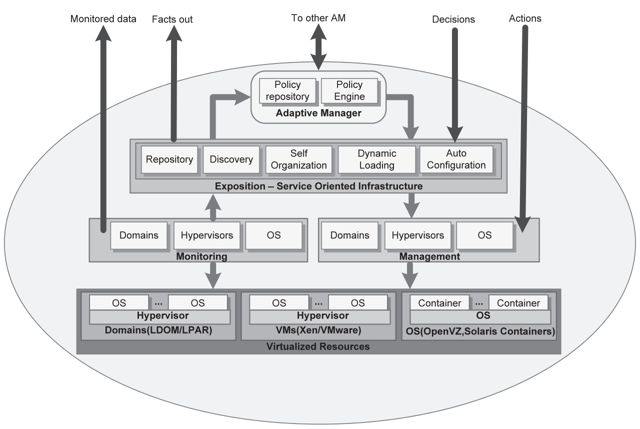

Fig. 5. Architecture of adaptive operational system layer.

The Operational Systems layer that includes the virtualized IT infrastructure and hardware assets is used for adaptive virtualized infrastructure development. A Virtual Execution Infrastructure model that takes this approach into account has been proposed and analyzed in the context of SOA-related technologies. The novelty of the proposed approach can be expressed as several important aspects [30].

- The VEI environment enables creating a container for resource allocation (grouping virtual resources) in the case of synchronized usage.
- Once deployed, the VEI container can be modified during runtime so the running application can obtain or release access to physical resources during execution.
- The VEI runtime management can be performed manually or by a policy engine, executing policy rules defined by the system administrator.
- The VEI container deployment and runtime management should be neutral from the perspective of SOA middleware usage.

The VEI system architecture presented in Fig. 5 leverages the pattern introduced in Section 3. More precisely, its components are specified as follows:

- *Virtualized resources.* This sublayer consists of the heterogeneous resources comprising the infrastructure. It also creates an abstraction of physical resources through proper instrumentation mechanisms. Instrumentation relies on provision of dynamic resource grouping and is used by the management

subsystem. Resource grouping relies on VM virtualization or container creation and dynamic modification of its configuration accomplished via virtual networks (as well as via modification of its dynamic parameters). This subsystem is also responsible for provisioning components enabling distribution of operating system images over the virtualized resource instances.
- *Monitoring.* This sublayer collects and aggregates information about the virtualized resources. Its operation relies on unification of information coming from different components (such as CPU, RAM) of virtual resources. Subsequently, this information is delivered to components (in the Exposition layer) as rule engine facts describing the system state.
- *Exposition layer.* Components added to adjust and unify virtualization mechanisms to simplify the interfaces realizing basic VEI system functions related to resource management. These components are exposed as SOI services and provide the following functionality:
  - Self-configuration and reconfiguration (sometimes also called autoconfiguration) supports the diversity of managed virtualization technologies. The system must be able to adapt to specific virtualization platforms and automatically install appropriate versions of components that are dynamically loaded and provide sensors and effectors to be used in a given hypervisor and OS. This ensures that the system is easy to deploy. Self-organization would enable dynamic creation of lists that contain all available physical nodes (which are further virtualized) and domains with hosted OS instances. Such functionality covers many requirements related to managed elements that should be automatically discovered by the Adaptive Manager during system startup or whenever a new element appears [8].
  - Representing monitoring data and system configuration using facts.
  - Provisioning services for searching and localization of resources (specifically, VEI system components realizing resource management); registration and retrieval of other components.
  - Resource management policy enforced via exposed effectors.
- *Management.* Components of this sublayer execute proper actions that are determined by the Adaptive Manager subsystem. The main operations that are executed by these components implement the controlling properties of the application's execution environment (VEI). These operations include resource optimization decisions. Actions that result from and are connected with changes in system components introduce adaptability into the Operational Systems layer.
- *Adaptive manager.* Implemented as a Policy Engine (Drools Rule Engine), which performs management actions referring to resource allocation for VEI.

5.2 Implementation Details

Adaptive Operational System layer is based on virtualization technologies. Feature analysis and practical verification of functionality underpin our selection of virtualization technologies in different types as hardware virtualization (Oracle LDOM), paravirtualization (Xen), and operating system containers (Solaris Containers).

The need for resource management based on complex heterogeneous mechanisms (mentioned above) has led us to choose Java Management Extensions (JMX) technology for developing the prototype of the Adaptive Operational Systems layer (more specifically, its resources and instrumentation layer). By using this technology, we can create and unify diverse methods of distributed object management in the form of a uniform and clear programming interface. A more extended study on implementing such complex systems is presented in [31].

5.3 Case Study

The prototype VEI implementation was tested on dedicated hardware and a dedicated network. This infrastructure consisted of a set of connected servers. The scenario assumed testing the proposed solution in conditions similar to a distributed environment. Therefore, resources were divided into three independent computing clusters and connected with network devices whose configuration corresponded to communication parameters available in WAN networks.

The system was tested through measurement of parameters that describe the resource level usage (e.g., average CPU usage). The sample application was a concurrent implementation of particle interactions computing service for computers with distributed memory, running in an MPI environment.

The experiment ran two instances of the test application in two groups of distributed resources connected with a WAN network with limited bandwidth. Optimization consisted in correcting virtual machine distribution (via VM migration) to eliminate the WAN bottleneck.

The purpose of the experiment was to show the ability to improve application operation through autonomic modifications of VM deployment according to information derived from monitoring network communication.[1] This optimization would be essential for distributed applications which need to exchange large amounts of data.

To conduct measurements, two VEI instances were created and the test application executed within them. The initial virtual machine deployment was random. The system collected information from network communication monitoring modules and enforced its optimization decisions by changing VM deployment. The optimization algorithm was implemented as a rule system.

Fig. 6 shows the load carried by the test application depended on the actions taken by the system. In the initial phase, when migration is taking place, access to resources is limited—hence, the application achieves lower performance compared with optimization-free execution. Once the migration process concludes, a significant performance

1. Executing two tightly bound VMs within one physical node results in the communication having practically unlimited bandwidth.

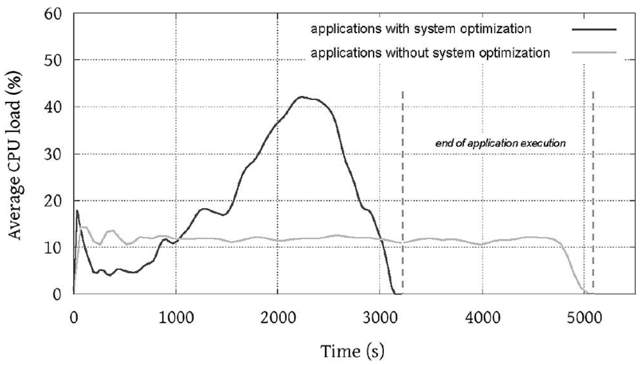

Fig. 6. The load carried by two VEI instances and the whole system.

increase is observed. This is because the bottleneck (in the form of the WAN network communication) has disappeared. The next result is a reduction in application response time.[2]

Another test was performed on a Solaris 10 instance with a number of running containers. Solaris Resource Manager enables specifying resource consumption limits for given container instances through the use of Fair Share Scheduler (FSS) [32], allowing optimal use of virtualized system resources. The system administrator must specify the importance of each workload running in a given container by assigning to it a number of shares according to the FSS model. In Solaris, these limits are expressed via resource control entities that can be set statically or changed dynamically during runtime. The latter case occurs in dynamic environments when new SOA services are provisioned over virtualized resources and new limits on resource consumption must be assigned. Another scenario involves migration of the VM which hosts the Solaris instance—in this case, operating conditions change as the target node might contain more resources and the internal configuration of FSS should be adjusted. The adaptation strategies can be expressed with control algorithms based on the control theory [33] and structured as open or closed-loop controllers. Given 1) S_w: shares assigned to workload W, 2) N: number of active workloads, and 3) S_i: shares assigned to active workload $i = 1, \dots, N$, the relative entitlement E_w of workload W can be expressed with the following equation [34], [35]:

$$E_w = S_w \Big/ \sum_{1}^{N} S_i. \tag{1}$$

Transformation of (1) yields the following formula, to be used by the open-loop controller, where 1) N_w is the number of workloads; 2) number of active workloads changes at time t according to activity state vector $A^t = [A_1^t, \dots, A_{N_w}^t]$, where $A_i^t = 0$ if W_i is not active and $A_i^t = 1$ if W_i is active, $i = 1, \dots, N_w$,

$$S_w^t = \left(U_w \sum_{i \neq s}^{N_w} S_i * A_i^t \right) \Big/ (1 - U_w). \tag{2}$$

Practical exploitation of such a controller is depicted in Fig. 7. The goal of the management policy expressed in

2. The profit would be more significant for applications with long execution times.

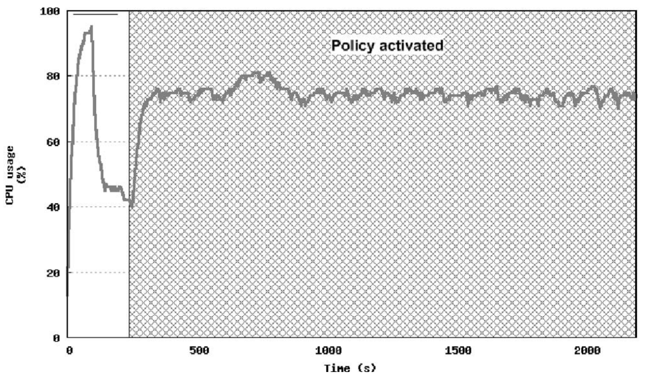

Fig. 7. The open loop controller for adaptive management of CPU consumption of Solaris container.

Drools was to guarantee CPU consumption on the level of 70 percent. It can be observed that following the activation of the policy, the settling time is approximately 2 minutes and the target workload is able to consume enough CPU resources. The presented scenarios describe one of the many practical aspects of system operation and prove the concept of applying virtualization techniques as an effective way of regulating access to resources and binding services to resources through a dedicated execution environment is useful in practice.

6 Adaptive ESB

This section presents how the AS3 element concept can be applied to constructing the Adaptive ESB, which is the main building block of the Adaptive Integration layer. As mentioned earlier, the Enterprise Service Bus is an integration technology that allows architects to compose applications with services using various communication protocols. ESB provides mechanisms for message normalization and routing between selected components. A generic structure of an Adaptive ESB functional element is shown in Fig. 8. As it was already described the policy engine perceives the system as an abstraction exposed by the exposition layer, which can be defined as a complex service execution model or a simple set of services involved in execution. In both cases, an adaptation policy has to be defined. This policy is used to represent a set of considerations guiding decisions. Each adaptation policy may express different goals of

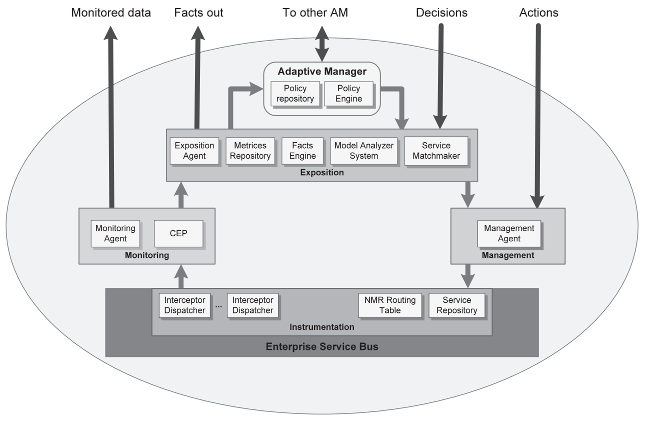

Fig. 8. Adaptive ESB functional elements.

system adaptation, such as minimizing system maintenance costs or ensuring particular QoS parameters. Design-time decisions cannot, however, take into account every context of service execution and lower layer architectural configuration. Hence, runtime architectural decision processing is particularly important for the S3 stack. The ESB layer must, therefore, be equipped with suitable adaptability mechanisms enabling the enforcement of these decisions.

6.1 Architecture

The Adaptive ESB functional element is constructed around the AS3 element pattern defined in Section 3 and depicted in Fig. 8. A number of distributed adaptive elements can be managed by a global Policy Engine in accordance with a high-level strategy. The instrumentation layer enriches the ESB with additional elements, providing adaptability transformations necessary to achieve adaptive ESB (described in the following sections). The monitoring layer is responsible for supplying notifications of events occurring in the execution environment. As the volume of monitoring information gathered from the ESB could overwhelm the Exposition layer, events are correlated with one another and notifications are sent only about complex events. Complex Event Processing can be compared to an inverted database containing stored statements: as data arrives in real time, these statements are executed and notifications are sent to registered listeners. The Adaptive Manager layer analyses facts and infers decisions which are then implemented in the execution environment. Facts representing the state of the system or events occurring in the system are supplied by the Exposition layer.

The approach presented in this section is a model-driven adaptation policy for SOA. The system analyzes composite services deployed in the execution environment and adapts to QoS and QoE changes. The user composes the application in a chosen technology and provides an adaptation policy along with a service execution model. The architecture-specific service composition layer continuously modifies the deployed service in order to enforce the adaptation policy. The abstract plan, providing input for architecture-specific service composition, can be hidden and used only by IT specialists during application development. System behavior is represented by the service execution model.

The composite service execution model is an abstraction of the execution environment. It covers low-level events and relations between services and exposes them as facts in a model domain for the Policy Engine. Decisions taken in the model domain are translated to the execution environment domain and then executed. The Model Analyzer System gathers monitoring data from the execution environment via Monitoring and Management Agents.

This process can be described as architecture-specific service composition adaptation that is performed in order to achieve the required value of QoS or QoE guided by the adaptation policy. The adaptation loop addresses service selection, binding protocol, and interaction policy choices. Thus, the adaptability process mainly concerns integration mechanisms.

6.2 Implementation Details

Monitoring ESB allows management actions to be performed. The presented concepts cover selected issues related to improving communication over ESB by separating traffic into several flows, which can then be independently handled. This leads to increased scalability and is required by many aspects of infrastructural functionality, including monitoring, data collection, management information distribution and security considerations.

The proposed mechanisms are compliant with existing integration technologies. Interceptor mechanisms enable dynamic control of service or component invocation, along with sensors and effectors necessary to close the adaptation loop.

The proposed adaptation mechanism provides elements necessary to close the control loop for ESB. It is used for compositional adaptation in systems which modify service composition while retaining their overall functionality. ESB is suitable for implementation of such adaptation, since one can modify message flows between components in accordance with high-level goals.

6.2.1 Sensors

An adaptive system should react in accordance with various goals, requiring several types of information from ESB. In most cases, this information will be disjunctive, so one would expect to deploy specialized types of sensors rather than generic ones. Interceptor design patterns fulfill these requirements, allowing interceptors to be deployed or undeployed at runtime.

A message is created in a service, is passed through the specialized Service Engine and is sent to the Normalized Message Router (NMR), which reroutes it to a particular destination through the same components. Common usage of this concept includes:

- *QoS measuring.* The functionality of monitoring interceptors is not limited to creating copies of messages sent through them, but may include more complex tasks, providing quantitative information related to service invocation. In the QoS interceptor, a message sent to a service is stored in the internal memory of the interceptor. When a response is generated, the previous message is correlated and response time is evaluated.
- *Tagging/Filtering.* Interceptors are commonly used for message tagging and filtering.

6.2.2 Effectors

Adaptive software has to modify itself to the changes in the execution environment. This adaptation requires performing actions (via effectors) that modify the execution characteristics of a system. For any sort of system, a set of operations has to be defined, along with places where these modifications should be introduced. It has been found that modifying message routes can affect the adaptation of complex services deployed in ESB.

Rerouting messages to other instances is justified only when these instances share the same interface and provide identical functionality. Current implementations of ESB that are compliant with the JBI specification share some common attributes used in the course of message processing. Each invocation of a complex service is described by its Correlation ID (CID) and is constant for one invocation,

TABLE 2
NMR Routing Table

Priority	VESB	CID	Intentional Service Name	EID	Decision
1	n/a	n/a	n/a	+	Service Name
2	n/a	+	+	n/a	Service Name
3	+	n/a	+	n/a	Service Name
4	n/a	n/a	+	n/a	Service Name
5	n/a	+	n/a	n/a	Service Name
6	+	n/a	n/a	n/a	Service Name
7	n/a	n/a	n/a	n/a	Service Name

even when passed among services. A particular message sent between services is described by an Exchange ID (EID). Generally speaking, the invocation of a complex service is described by a CID and consists of several message exchanges described by an EID. Extending the Normalized Message Router with a routing algorithm capable to modify message routing would yield an adaptive ESB.

Once the message reaches the NMR, the routing algorithm relies on matching the message to routing rules in the routing table. If the message matches a routing rule, that rule is fired and the Service Name from the routing rule substitutes the intended destination Service Name. Routing rules are split into groups—Virtual ESB (VESB) with different priorities, analyzed in a particular order. In every message, parameters such as VESB tag, Correlation ID, intended Service Name, and Exchange ID are matched to routing rules. If the message matches several rules, one of them is selected on a round-robin basis to provide load balancing.

Depending on the priority value, different parameters are matched. The presented matching criteria are summarized in Table 2. The lower the priority value the more important the routing rule. Some attributes are omitted when processing rules, though each routing rule returns a Service Name as a result. The decision element which closes the adaptation loop already presented in Fig. 8 gathers information about ESB from sensors and uses effectors to dynamically modify the routing table at runtime.

6.3 Case Study

The purpose of this scenario is to demonstrate how the system responds to disturbances in the execution environment.

Let us assume that one would like to create a *GeoWeather* service providing weather information for a given location. Location data will be provided as a zip code, GPS location, IP address of a computer, or country name. Another assumption is that accuracy can be limited to cities.

Widely available weather services do not provide weather information for GPS coordinates. Instead, they return weather information for a given readable address—thus all possible input formats have to be converted to addresses and then validated against the weather information service.

In order to better present system responses, a fragment of the service is analyzed, i.e., it is assumed that requests only contain geographical locations. The ESB contains several other services that can be used interchangeably. The model of the analyzed service and its projection onto instances found in the ESB is depicted in Fig. 9. There are six possible combinations of service instances that might be executed.

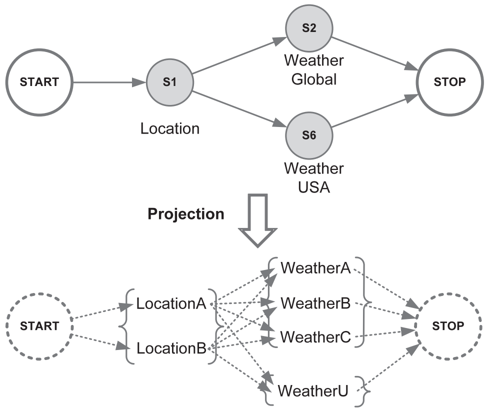

Fig. 9. Projection of an abstract composite service into a composite service.

Without an adaptation policy, user requests are handled by services called *locationB* and *weatherB*. In normal circumstances the agreed-upon response time is in the range of 400-450 ms. Unfortunately, sometimes these instances are overloaded or unloaded, resulting in uneven response times of the whole complex service. Areas marked in Fig. 10 show when disturbances are observed. In the 60-120 s period, the *locationB* service execution time increases from about 200 to 500 ms, inducing an overall execution time of 700 ms which is 300 ms more that it should be. In the 180-240 s period, the *weatherB* service responds about 100 ms faster than normal, resulting in the shortest overall execution time. During the first disturbance phase, the client did not receive a response in the required time, whereas during the second phase the client obtained better-than-agreed upon service quality. With an adaption policy active, the system counteracts disturbances in the execution environment and switches to a different set of service instances whenever it is necessary to maintain the agreed-upon level of QoE.

7 Adaptive Components

Adaptive Components are used in the Adaptive S3 model to provide adaptability for the Service Components layer. The main goal of Adaptive Components is to provide a background for creating atomic services used by the other layers.

In SOA systems, atomic services are mainly comprised of components. The need to ensure implementation and

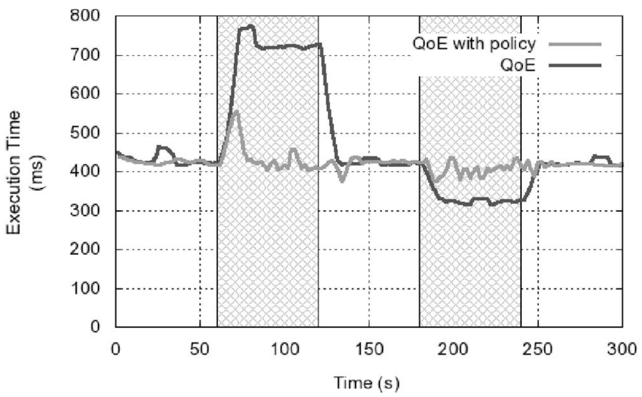

Fig. 10. Evaluation of ESB use case scenario.

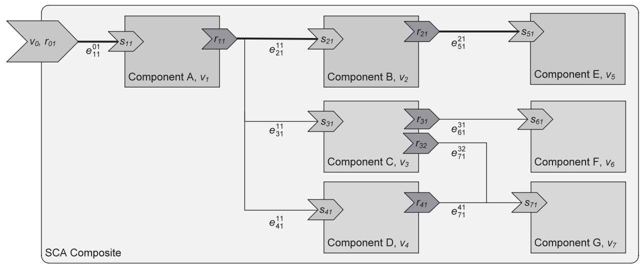

Fig. 11. Sample composition with a selected Composition Instance.

communication protocol independence forces the selection of an independent integration environment. The Service Component Architecture (SCA) specification [36] provides such a solution. It supports several component implementations (such as Java, C++, BPEL, and Spring) as well as various communication protocols (Web Services, RMI, and JMS). Moreover, it is highly extensible, as evidenced by different SCA implementations [37] which introduce additional technologies not covered by the original blueprint.

However it should be noted that the SCA specification lacks of adaptability mechanisms, which play a crucial role in ensuring conformance between the provided services and changing user requirements (i.e., Quality of Service).

The SCA specification involves the concept of a composite which consists of a set of components connected by wires. A wire consists of a service representing the functionality exposed by one component, a reference representing the functionality required by another component (delegate object) and a binding protocol which represents the communication protocol between the reference and the service. Several components in an SCA composite may expose services for external communication. These services are further exposed, e.g., via ESB.

Services created using SCA composites should be able to adapt to changing customer requirements and service provider capabilities (such as different CPU speeds resulting from infrastructure changes, etc.) Nonfunctional customer requirements are recognized as QoS metrics and may be provided by the QoS layer of the S3 model. On the other hand, the measured service QoE, represents actual capabilities of a service. In an ideal case, QoE metrics should be as close to QoS requirements as possible.

An SCA composite may be perceived as a directed graph with nodes representing components and edges representing wires (directed from references to services), hereafter called a *Composition Instance* (CI). A CI is connected with a specific QoE description derived from a composite. A set of *Composition Instances*, which meet the same functional requirements and differ only with respect to nonfunctional ones (QoS) is called a *Composition*. *Compositions* may also be represented as a directed graph, created by joining all CIs which provide a given element of functionality. The rules for joining CIs are as follows: if CI1 and CI2 use the same component as a specific node, these components can be joined; otherwise both components need to be added to the graph. Such a solution reduces the amount of resources required by all CIs (by exploiting shared nodes) and enables more efficient CI processing. A sample *Composition* with a selected *Composition Instance* is depicted in Fig. 11.

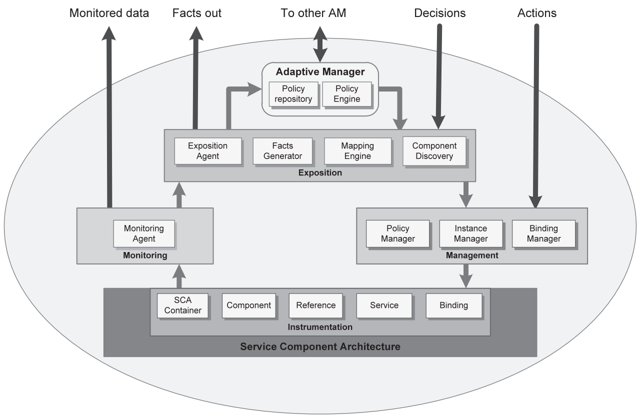

Fig. 12. Adaptive SCA.

7.1 Architecture

The architecture of the Adaptive SCA is depicted in Fig. 12, and it is a realization of the previously described AS3 Element. The Adaptive SCA service is a service created using SCA technology with the proposed enhancement, which could be integrated by the Adaptive ESB Element.

Instrumentation is used to enhance the SCA Container to manage components, their services and references, and binding protocols used for remote communication. The monitoring layer includes the Monitoring Agent, which gathers QoE metrics for the service. The exposition layer is mainly composed of the Fact Generator (used to create specific facts further used by the Policy Engine), and the Mapping Engine, which is responsible for applying mappings between the Composition and Composition Instance into a particular set of components, discovered by the Component Discovery. Policy, Instance, and Binding Managers perform low-level management tasks, such as applying the selection of components taking part in the Composition Instance.

7.2 Implementation Details

Adaptive SCA comes as a set of extensions for the Service Component Architecture [36] (in particular, the Apache Tuscany SCA [37] implementation). These extensions enhance the SCA component model, turning it into an Adaptive SCA component. Low-level extensions provide interceptors injected into SCA references to enable sensor invocations and apply selection of proper services and binding protocols (effectors), as depicted in Fig. 13.

For better understanding of this process, the figure should be compared with the service model (cf. Fig. 11). A reference is connected with several component services in the Composition, but only one should be selected within the current Composition Instance. This selection is performed by Instance and Binding Effectors.

The functionality of sensors and effectors is exposed by the Monitoring and Management Interface (MMI), which is used on the upper level of the AS3 Element.

The Policy Engine is used to process adaptation policies provided with the service description (e.g., by the service customer). Such policies describe QoS requirements using a high-level abstract language and they are further mapped to specific rules used to select proper CIs. CI descriptions (maintained by the Adaptive SCA Element) are used to

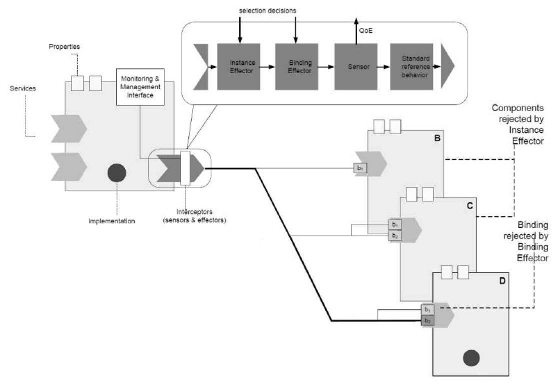
selection decisions
QoE
Properties
Services
Monitoring & Management Interface
Instance Effector
Binding Effector
Sensor
Standard reference behavior
Components rejected by Instance Effector
B
C
D
b1
b2
Binding rejected by Binding Effector
Implementation
Interceptors (sensors & effectors)

Fig. 13. Adaptive SCA component with interceptors and reference selection process.

perform selection decisions using component reference effectors exposed by MMI (cf. Fig. 13).

7.3 Case Study

Our case study presents a possible adaptation of a typical service created using Adaptive SCA. The service provides banknote recognition functionality. The Composition which realises this functionality is depicted in Fig. 14.

The main component, *BanknoteRecognizer*, implements the overall functionality and contains two references. The first reference is used to provide the functionality of loading an image from a remote location and converting it to a specific file format (using specialized libraries, such as *ImageMagick* or *NetPBM*), while the second one is used to scan two images for similarities.

There are four Composition Instances defined in the Composition, which differ only in their nonfunctional parameters. The CIs are as follows:

- CI 1. Banknote Recognizer, Image Loader, File Loader, NetPBM Image Converter, NeuralNetwork Image Recognizer.
- CI 2. Banknote Recognizer, Image Loader, File Loader, NetPBM Image Converter, Edge Recognizer, Edge Detector.
- CI 3. Banknote Recognizer, Image Loader, File Loader, ImageMagick Image Converter, NeuralNetwork Image Recognizer.
- CI 4. Banknote Recognizer, Image Loader, File Loader, ImageMagick Image Converter, Edge Recognizer, Edge Detector.

With the quality metrics presented in Table 3.

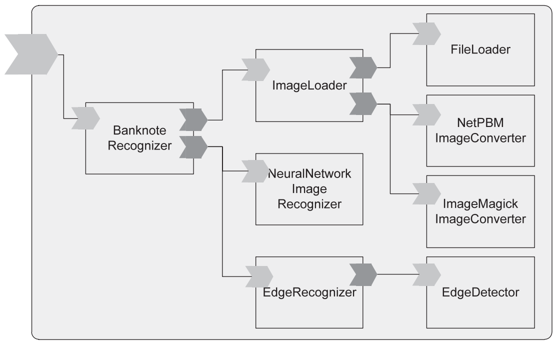

Fig. 14. Case study composition.

TABLE 3
Quality of Service for Case Study CIs

CI name	**invocation time** [s]	**cost** [eurocents]	**memory usage** [MB]
CI 1	5	2	2000
CI 2	8	5	1000
CI 3	2	9	1500
CI 4	5	4	1400

The presented experiment shows a typical situation, when the capabilities of the service provider change temporarily, e.g., due to a brief system overload (which may result from increased service demand, exposure of new services, etc.).

The adaptation policies used in the case study are defined as follows:

- Keep average *invocation time* between 4 and 6 seconds.
- Keep average *cost per invocation* between 3 and 6 eurocents.
- Maximum allowed *invocation time* is 11 seconds.
- Keep average *memory usage* between 1,300 and 1,800 MB.
- If *invocation time* and *cost* requirements are fulfilled, try to find a CI with lower *cost*.

These policies are transformed to low-level rules by the Policy Engine, following which the adaptation is performed. During service execution the system discovers that CI 4 may be used (thus the provided QoE meets requirements described by the policies). However, after 55 seconds of service execution, the invocation time of CI 4 increases to 10 seconds, which is not acceptable. This causes selection of the other CIs, which in turn results in proper average QoE. After 87 seconds, the invocation time of CI 4 reverts to its initial value (5 seconds), hence this CI is again selected. The observed QoE achieved with this adaptability process is depicted in Fig. 15.

8 Related Work

The presented work concerning VEI is very much related to research on Grid resource management. This paper [38] presents how to develop collaborative Grids into stable, flexible, and dynamic resource management environments

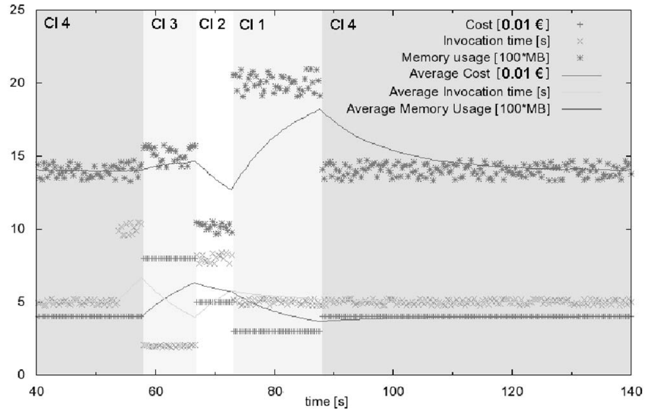

Fig. 15. QoE in the case study experiment.

by means of a Collaborative Awareness Management (CAM) model. CAM optimises resource collaboration, promotes cooperation, and responds to specific circumstances.

Management of the underlying network infrastructure supporting Grid communications is not performed by the same management systems which manage the Grid itself. In the presented scenario, integrated management of VEI and network could simplify the overall maintenance processes. Work [39] proposes a hierarchical policy-based architecture whose goal is to allow such integration where Grid policies are translated to network policies according to rules defined by network administrators. It also describes a prototype implementation of such an architecture.

As it can be observed that most of these efforts address the specific issue of virtualizing computing resources and concentrate on problems resulting from integration with existing environments at the management level. There is an observable lack of complex solutions addressing different resource virtualization mechanics.

Some projects attempt to extend adaptability mechanisms in ESB, but these solutions are focused on selecting the most appropriate service instances without evaluating how it may influence overall composition. Chang et al. propose a Dynamic Composition Handler (DCH) [16] for ESB. Business processes are defined using BPEL and may include several activities. Some of them may have more than one service component which can perform a given activity. DCH determines the most appropriate service component and invokes it according to the requirements and circumstances of various service consumers. Bai et al. propose Dynamic Routing in the Enterprise Service Bus (DRESR) [40] framework to enable dynamic message routing. The authors define an Abstract Routing Table as an execution sequence of services in terms of abstract service specifications. Prior to execution of an abstract service, the framework analyses existing service providers and chooses one that meets nonfunctional requirements such as response time. The services are evaluated based on current testing results as well as historical data. Chen et al. present an Adaptable Service Bus (ASB) [41] that enables, to some extent, dynamic composition of services. In order to modify the behavior of the application at runtime, developers can adjust the parameter values maintained in external storage.

In [15] and [6], Morin et al. present results of their work in the DiVA [42] project, which focuses on dynamic variability in complex, adaptive systems. In several aspects, this work is related to the concepts of Adaptive Components. The aim of the prepared *dynamically adaptive systems* is to provide the functionality described with a specified model, according to specified QoS and current context (e.g., user location). The authors propose several metamodels for architecture, context, and reasoning description. Although adaptation to context changes is precisely described, the methods of adapting to changing QoS requirements and capabilities are only mentioned.

In [14], Vuković presents service composition methods for a context-aware environment, especially a mobile one. Nau et al. investigate planning-based service composition [13], [43]. The framework, called GoalMorph, mainly focuses on resilience to failures arising both at composition and execution time.

S-Cube (Software Services and Systems Network [44]) is a project funded by the European Community's Seventh Framework Programme. One of its activities (WP-JRA-1.2) focuses on "Adaptation and Monitoring Principles, Techniques and Methodologies for Service-based Applications [12]." According to its description, it should propose solutions for different levels of adaptation, in particular for: optimization, recovery, QoS-based adaptation, evolution, mediation. In contrast to the presented work, it uses adaptation mechanisms for composing services into business processes and recomposes them according to adaptation strategies. Unfortunately, this project is scheduled for 2007-2013 and currently there are no detailed results on the investigation of adaptation mechanisms.

9 Conclusions and Future Work

This paper presents the concept of the Adaptive S3 Model. Individual layers of this model are currently in the process of development and subject to intensive research which requires a lot of effort. The major contribution of this paper is the proposition of how the adaptation loop construction—the AS3 element pattern—could be applied in a uniform way across the S3 layers to reduce the complexity of SOA applications QoE and QoS governance. The essential part of this study shows practical usage of the AS3 pattern across different layers. Such an approach, supported by proper selection of available software tools for code instrumentation, monitoring, and policy processing, leads to an efficient and scalable solution. This has been illustrated by the implementation of adaptive Operational Systems, Integration and Service Components layers.

Uniform implementations of the AS3 elements over different layers facilitate their interoperability and provide opportunities for multilayer adaptation of SOA systems. This challenge has not been explored in this paper, but the presented systems are fully prepared for such experiments and we intend to pursue them in the course of our future work. The integrated adaptation of SOA applications and their execution environments should lead to full control over service QoS and QoE. Realization of this vision requires careful analysis of new adaptation strategies, metrics, and models.

The proposed study relies on the adaptive system concept. The complexity of management actions means that autonomic systems, being the most mature and advanced form of adaptation, are an appropriate solution in this regard. More complex adaptation algorithms can take into account additional knowledge gained during system operation, while other parts of the proposed system can be applied without any changes.

This paper focuses on the Adaptive S3 model concept, while software tools for AS3 elements for particular layers are not presented. A collection of these tools, called AS3 Studio [45], is now under development at the Computer Science Department, AGH University of Science and Technology. AS3 Studio supports implementation and deployment of adaptation loop components via a user-friendly GUI. It also facilitates convenient definition of policy rules and retrieval of monitoring data. It is constructed as a set of plug-ins for the Eclipse framework.

ACKNOWLEDGMENTS

The research presented in this paper was partially supported by the European Union in the scope of the European Regional Development Fund program no. POIG.01.03.01-00-008/08.

REFERENCES

[1] P. Bachara, K. Blachnicki, and K. Zielinski, "Framework for Application Management with Dynamic Aspects J-EARS Case Study," *Information and Software Technology,* vol. 52, no. 1, pp. 67-78, 2010.

[2] N. Bartlett, *OSGi In Practice,* http://njbartlett.name/osgibook.html, 2009.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03),* pp. 164-177, 2003.

[4] E.L. Haletky, *VMware ESX Server in the Enterprise: Planning and Securing Virtualization Servers.* Prentice Hall, 2008.

[5] M. Lageman, *Solaris Containers What They Are and How to Use Them,* Sun Microsystems, http://www.sun.com/blueprints/0505/819-2679.pdf, 2005.

[6] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, and J.-M. Jézéquel, "Modeling and Validating Dynamic Adaptation," *MoDELS Workshops,* pp. 97-108, 2009.

[7] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer,* vol. 36, no. 1, pp. 41-50, 2003.

[8] A.G. Ganek and T.A. Corbi, "The Dawning of the Autonomic Computing Era," *IBM Systems J.,* vol. 42, pp. 5-18, 2003.

[9] A. Janik and K. Zielinski, "Adaptability Mechanisms for Autonomic System Implementation with AAOP," *Software—Practice and Experience,* vol. 40, no. 3, pp. 209-223, 2010.

[10] L.-J. Zhang, N. Zhou, Y.-M. Chee, A. Jalaldeen, K. Ponnalagu, R.R. Sindhgatta, A. Arsanjani, and F. Bernardini, "SOMA-ME: A Platform for the Model-Driven Design of SOA Solutions," *IBM Systems J.,* vol. 47, no. 3, pp. 397-413, 2008.

[11] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: A Service-Oriented Reference Architecture," *IT Professional,* vol. 9, pp. 10-17, 2007.

[12] C. Cappiello, K. Kritikos, A. Metzger, M. Parkin, B. Pernici, P. Plebani, and M. Treiber, "A Quality Model for Service Monitoring and Adaptation," *Proc. Workshop Monitoring, Adaptation and Beyond (MONA+) at the ServiceWave Conf.,* Dec. 2008.

[13] D.W. Evren, D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia, "Automatic Web Services Composition Using SHOP2," *Proc. Workshop Planning for Web Services,* 2003.

[14] M. Vuković, "Context Aware Service Composition," PhD dissertation, Univ. of Cambridge, Apr. 2006.

[15] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg, "Models@Run.time to Support Dynamic Adaptation," *Computer,* vol. 42, pp. 44-51, 2009.

[16] S.H. Chang, H.J. La, J.S. Bae, W.Y. Jeon, and S.D. Kim, "Design of a Dynamic Composition Handler for ESB-Based Services," *Proc. IEEE Int'l Conf. e-Business Eng. (ICEBE '07),* pp. 287-294, 2007.

[17] I.Y. Chen, G.-K. Ni, and C.-Y. Lin, "A Runtime-Adaptable Service Bus Design for Telecom Operations Support Systems," *IBM Systems J.,* vol. 47, no. 3, pp. 445-456, 2008.

[18] K.-J. Lin, M. Panahi, Y. Zhang, J. Zhang, and S.-H. Chang, "Building Accountability Middleware to Support Dependable SOA," *IEEE Internet Computing,* vol. 13, no. 2, pp. 16-25, 2009.

[19] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer,* vol. 36, no. 1, pp. 41-50, Jan. 2003.

[20] "Esper—Event Stream and Complex Event Processing for Java," http://www.espertech.com, Dec. 2009.

[21] P. Browne, *JBoss Drools Business Rules.* Packt Publishing Ltd., 2009.

[22] E. Friedman-Hill, *Jess in Action: Java Rule-Based Systems (In Action Series).* Manning Publications, Dec. 2002.

[23] P.K. McKinley, S.M. Sadjadi, E.P. Kasten, and B.H.C. Cheng, "Composing Adaptive Software," *Computer,* vol. 37, no. 7, pp. 56-64, 2004.

[24] J. Strassner and D. Raymer, "Implementing Next Generation Services Using Policy-Based Management and Autonomic Computing Principles," *Proc. IEEE/IFIP 10th Network Operations and Management Symp. (NOMS),* 2006.

[25] *Java Management Extensions (JMX) Specification, Version 1.4, JSR 160,* Sun Microsystems, http://jcp.org/en/jsr/detail?id=160, 2006.

[26] "Service Oriented Infrastructure Reference Framework," draft technical standard, The Open Group, SOA Working Group, https://www.opengroup.org/projects/soa-soi/uploads/40/19218/soi-V1-5-P1.pdf, 2008.

[27] R.W. Doran, "Amdahl Multiple-Domain Architecture," *Computer,* vol. 21, pp. 20-28, 1988.

[28] M. Rosenblum, "The Reincarnation of Virtual Machines," *Queue,* vol. 2, no. 5, pp. 34-40, 2004.

[29] S.J. Vaughan-Nichols, "New Approach to Virtualization is a Lightweight," *Computer,* vol. 39, pp. 12-14, 2006.

[30] J. Kosińska, J. Kosiński, and K. Zieliński, "Virtual Grid Resource Management System with Virtualization Technology," *Proc. Second Conf. High Performance Computers' Users (KU KDM '09),* 2009.

[31] K. Balos, M. Jarzab, D. Wieczorek, and K. Zielinski, "Open Interface for Autonomic Management of Virtualized Resources in Complex Systems Construction Methodology," *Future Generation Computer Systems,* vol. 24, no. 5, pp. 390-401, 2008.

[32] J. Kay and P. Lauder, "A Fair Share Scheduler," *Comm. ACM,* vol. 31, no. 1, pp. 44-55, 1988.

[33] J.L. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury, *Feedback Control of Computing Systems.* Wiley-IEEE Press, 2004.

[34] M. Jarzab and K. Zielinski, "Framework for Consolidated Workload Adaptive Management," *Proc. Software Eng. in Progress—II IFIP Central and East European Conf. Software Eng. Techniques,* pp. 17-29, 2006.

[35] J. Adamczyk, R. Chojnacki, M. Jarzab, and K. Zielinski, "Rule Engine Based Lightweight Framework for Adaptive and Autonomic Computing," *Proc. Eighth Int'l Conf. Computational Science (ICCS),* pp. 355-364, 2008.

[36] *SCA Service Component Architecture, Assembly Model Specification, Version 1.00,* OASIS, May 2007.

[37] Apache Tuscany Home Page, http://tuscany.apache.org, 2012.

[38] P. Herrero, J.L. Bosque, M. Salvadores, and M.S. Perez, "A Rule Based Resources Management for Collaborative Grid Environments," *Int'l J. Internet Protocol Technology,* vol. 3, no. 1, pp. 35-45, 2008.

[39] R. Neisse, E.D.V. Pereira, L.Z. Granville, M.J.B. Almeida, and L.M.R. Tarouco, "An Hierarchical Policy-Based Architecture for Integrated Management of Grids and Networks," *Proc. IEEE Fifth Int'l Workshop Policies for Distributed Systems and Networks,* pp. 103-106, 2004.

[40] X. Bai, J. Xie, B. Chen, and S. Xiao, "DRESR: Dynamic Routing in Enterprise Service Bus," *Proc. IEEE Int'l Conf. e-Business Eng.,* pp. 528-531, 2007.

[41] I.-Y. Chen, G.-K. Ni, and C.-Y. Lin, "A Runtime-Adaptable Service Bus Design for Telecom Operations Support Systems," *IBM Systems J.,* vol. 47, no. 3, pp. 445-456, 2008.

[42] DiVA Home Page, http://www.ict-diva.eu, Jan. 2010.

[43] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice.* Morgan Kaufmann, 2004.

[44] S-Cube, the Software Services and Systems Network Project Home Page, http://www.s-cube-network.eu, Dec. 2009.

[45] IT-SOA Project, http://www.soa.edu.pl, 2012.

Krzysztof Zieliński is working as a full professor and head of the Institute of Computer Science at AGH-UST. His interests focus on networking, mobile and wireless systems, distributed computing, and service-oriented distributed systems engineering. He is the author of more than 200 papers in this field. He has been the project/task leader in numerous EU-funded projects, e.g., PRO-ACCESS, 6WINIT, and Ambient Networks. He served as an expert with the Ministry of Science and Education. Now, he is leading SOA-oriented research performed by the IT-SOA Consortium in Poland. In this area, his research interests concern adaptive SOA solution stack, services composition, service delivery platforms, and methodology. He is a member of the IEEE, ACM, and the Polish Academy of Science, Computer Science Chapter.

Tomasz Szydło is working toward the PhD degree and is also a research assistant in the Department of Computer Science at AGH-UST. Recently, he has been working on the SOA project conducted by several universities in Poland. His interest in this project is concentrated on adaptive ESB. His interests focus on service-oriented architectures as well as mobile systems. He has participated in several EU research projects including CrossGrid and Ambient Networks. He is the author of 10 research papers. He has cooperated with the Machine Learning and Inference Laboratory at George Masson University, working on the conversion of rules into decision trees. He was also an intern at IBM Hursley, United Kingdom, where he worked on integration of the MQTT protocol with service-oriented device architectures. He is involved in the CISCO Regional Academy at AGH-UST as an instructor.

Robert Szymacha is working toward the PhD degree and is also a research assistant in the Department of Computer Science at the AGH-University of Science and Technology, Krakow, Poland. Currently, he is working in the SOA research project performed by the IT-SOA Consortium in Poland. He works on adaptive component integration and QoS-aware services. His interests focus on networking, distributed systems, context-aware systems, and SOA applications. He is author of nine papers in these areas. He has been working on several large projects, like 6FP EU: Ambient Networks, SGI Project: Parallel Isosufraces Visualization and Reconstruction, and Sun Microsystems: Spontaneous Containers Implementation in Java Multitasking Virtual Machine.

Jacek Kosiński has been working as a PhD research assistant in the Department of Computer Science, AGH University of Science Technology, Krakow, since 2000. He is a Cisco CCNP instructor. He has participated in national and international research projects, mainly EU-founded. Currently, he is working on an SOA research project performed by the IT-SOA Consortium in Poland and on a project related to the Polish national grid initiative—PL-Grid. He works on the usage of virtualization techniques in areas related to resources management. His main interests focus on computer networks, operating systems, and virtualization. He is the author of several papers in these areas.

Joanna Kosińska has been working as a research assistant in the Department of Computer Science, AGH University of Science and Technology, Krakow, since 2001. She has participated in several national and international research projects, mainly EU-funded. Currently, she is working on the SOA research project performed by the IT-SOA Consortium in Poland. She works on the usage of virtualization techniques in areas related to resources management. Her main interests focus on service-oriented architecture and distributed environments based on Java.

Marcin Jarzab received the MSc degree in computer science from the University of Science and Technology (AGH-UST) in Krakow, Poland, in 2002. He worked as a software consultant at Consol* Solutions and Software from 2000-2002, participating in many projects for Telco companies. He was an intern at Sun Labs in the latter half of 2003, investigating the application of the Multitasking Java Virtual Machine to the J2EE environment. His research interests include the tuning and performance evaluation of distributed systems, design patterns, frameworks, and architectures of autonomic computing environments.