Chapter 6
Implementation, Deployment and Governance of SOA Adaptive Systems

R. Brzoza-Woch, Ł. Czekierda, J. Długopolski, P. Nawrocki, M. Psiuk, T. Szydło, W. Zaborowski, K. Zieliński and D. Żmuda

Abstract This chapter introduces a pragmatic methodology for adding and managing adaptability in multiple layers of the SOA application execution infrastructure. Adaptability mechanisms and techniques are investigated by referring to the MAPE-K pattern, which is viewed as the most representative solution for adaptive and autonomous systems. The SOA solution stack developed by IBM is selected as the basis for the application execution infrastructure model. This makes the proposed concepts easier to understand, while not detracting from their general nature. The adaptability aspect is considered in a broad context, with attempts to address, in a uniform way, all SOA applications composed of software services (Virtual Services) and hardware components (Real World Services). The proposed methology is supported by the AS3 Studio package which is a complete suite of tools providing extensions of SOA systems with adaptability features. This methodology is presented as a crucial part of the governance process of SOA applications. Finally, a case study which illustrates the proposed approach is described.

1 Introduction

Service-oriented applications operate in dynamic business environments. These applications should therefore become highly flexible and adaptive, as they need to adequately identify and react to various changes observed in the execution environment [78]. Adaptation is the relation between a system and its environ-

R. Brzoza-Woch · Ł. Czekierda · J. Długopolski · P. Nawrocki · M. Psiuk · T. Szydło · W. Zaborowski · K. Zieliński(✉) · D. Żmuda
Faculty of Computer Science, Electronics and Telecommunications,
Department of Computer Science, AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: kz@agh.edu.pl

ment where change is provoked to facilitate proper operation of the system in the environment.

The adaptation process of SOA applications should be investigated in the context of the widely accepted characteristics of service orientation proposed by Thomas Erl [20] which refer to the following eight principles:

- Standardized Service Contract,
- Service Loose Coupling,
- Service Abstraction,
- Service Reusability,
- Service Autonomy,
- Service Statelessness,
- Service Discoverability,
- Service Composability.

It is evident that, taken together, these principles allow for easier adaptation of SOA applications by system integrators during the (re-)development process and at runtime. Standardized Service Contract principles mean that services within the same service inventory remain in compliance with the same contract design standards which simplifies applications integration. Moreover, the service contract is usually developed separately from the service logic and provides the sole means of accessing service functionality and resources. This allows for independent creation of a contract and service implementation, reducing unintentional dependencies between services. This feature is referred to as Service Loose Coupling and supports adaptability, since each service can be controlled separately. Service contracts contain only essential information about services, thus enabling their categorization. This principle, known as Service Abstraction, makes searching for suitable services more efficient. The Service Autonomy principle means that services should exercise a higher level of control over their underlying runtime execution environment, which implies more predictive behavior. The ability of services to function autonomously is achieved by reducing shared access to service resources and increasing their physical isolation. It simplifies adaptation performed on the resource allocation level. The autonomy of individual services is also especially important for adaptation of SOA applications performed at the service composition or integration level. This feature is complemented by the Statelessness principle, which implies that services should minimize resource consumption by deferring the management of state information when necessary. Additionally, services are supplemented with communication-oriented metadata with which they can be effectively discovered and interpreted. This is an important prerequisite of dynamic structural adaptation.

Service orientation principles allow for effective development of adaptive SOA applications. Unfortunately, SOA is mainly focused on design techniques which support developers in constructing services. It does not encompass runtime aspects of service operation, i.e. how to manage and maintain services. Without proper management business objectives cannot be met as it is impossible to specify goals and determine whether they are, in fact, reached.

SOA applications and services should satisfy Service Level Agreements despite changing execution conditions such as system load, number of users, etc. This is why loose coupling and composition features inherent in SOA should be extended to cover adaptive behavior [10], ensuring self-adaptation of the system at the runtime. This aspect is often referred to as compositional adaptation [77], enabling software to modify its structure and behavior dynamically in response to changes in its execution environment. Adaptive systems are a remedy for the complexity of computing environments, 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 by middleware technologies and virtualized computational infrastructures. Better exploitation of modern IT infrastructures is a prerequisite of achieving the required QoS (Quality of Service) and end-user satisfaction, characterized by QoE (Quality of Experience).

Implementing an adaptive SOA system remains a challenging issue: the adaptation process requires suitable mechanisms to be built into applications or the execution environment itself. Satisfying service orientation principles means that self-adaption of SOA systems can be considered a self-contained aspect, introduced during development or at runtime. Adaptive applications can be developed in the process of transforming a preexisting application and hardware components (which does not involve changes in the application's business services). This is consistent with the fact that the investigated adaptive systems remain business-agnostic and instead focus on ensuring the required nonfunctional parameters.

SOA application development and deployment should be considered in the context of the SOA Solution Stack (S3) proposed by IBM [3], which provides a detailed architectural definition of SOA. This allows for clear separation of different adaptability mechanisms referencing particular layers of the S3 model and assigning the requested adaptability extensions.

The goal of this chapter is to propose a pragmatic methodology for adding and managing adaptability aspects in multiple layers of the S3 stack, as well as to present the Adaptive S3 (AS3) Studio package which is a complete suite of tools supporting extensions of SOA systems with adaptability features. This methodology is considered part of the governance process of SOA applications. The adaptability mechanisms and techniques are investigated by referring to the MAPE-K pattern [1] as the most representative for adaptive and autonomous systems.

The adaptability aspect is considered in a broad context, with attempts to address, in a uniform way, all SOA applications composed of software services (also called Virtual Services) and hardware components (Real World Services) [35]. Such an approach is justified by the increasing importance of pervasive systems [67], bringing interaction from enterprise systems back to the real world. In this context the adaptive behavior of Real Word Services is considered a critical element, combining adaptive interaction, adaptive composition and task automation by involving knowledge of user profiles, intentions and previous usage patterns.

2 Development of Adaptive Systems: Motivation

This section describes the fundamentals of adaptive system development, referring to the state of the art in this domain. The presented analysis is performed in the context of the SOA Governance process definition and requirements to precisely identify the place and role of adaptive systems in SOA applications. Adaptability mechanisms are evaluated by taking into account service orientation principles to better illustrate the challenges involved in SOA adaptive system implementation.

According to [51] the goal of SOA Governance is to ensure reliable long-term operation of a SOA. More specifically, it provides the ability to guarantee SOA adaptability and integrity as well as check services for capability, security and strategic business alignment. Its overall goal is SOA Compliance, i.e. compliance with legal, normative and intra-company regulations respectively. SOA Governance includes the identification of a decision-making authority for the definition and modification of business processes that are supported by SOA and the requirements for service levels and performance including access rights to the services. It also defines the way how reusable services are defined, designed, accessed, executed, and maintained as well as the determination of service ownership and cost-allocation in a shared-service organization. This definition includes all crucial tasks and activities of SOA Governance and structures them into organizations, processes, policies and metrics. In addition, this definition covers all important aspects of IT Governance and specializes them in the context of SOA. SOA Governance defines the organizational structure of SOA and provides a way to implement it within an existing corporate structure.

Numerous models for SOA Governance have been proposed so far. Ten of them are investigated and compared in [51]. Each emphasizes different aspects, including service lifecycle management and organizational change [8]. The result of this study is the TEXO Governance Framework [32], compiled on the basis of the existing frameworks. The processes provided by this framework (depicted in Fig. 1) have been grouped into five phases: design, deployment, delivery, monitoring, and change.

The design phase covers all strategic aspects related to the operation of a service marketplace, enabling services to be purchased and traded. The development and deployment of services as well as selection of third-party services belong to the deployment phase. The delivery phase addresses all aspects of service and infrastructure operations. It is closely coupled with the monitoring phase as both phases occur concurrently. The monitoring phase covers all aspects of service and infrastructure monitoring. The change phase contains all processes and tasks required to adjust and change the infrastructure and services traded in the marketplace.

The presented division of governance phases leads to the conclusion that adaptation processes primarily concern the monitoring and change phases of the Governance Framework. The activities most relevant to the adaption process are highlighted in Fig. 1. Adaptation can be performed manually (by a system administrator) or automatically. System administrators may specify high-level policies which determine how the system should adjust its behavior at runtime in order to meet the specified requirements. Administrators are consequently relieved from dealing with low-level

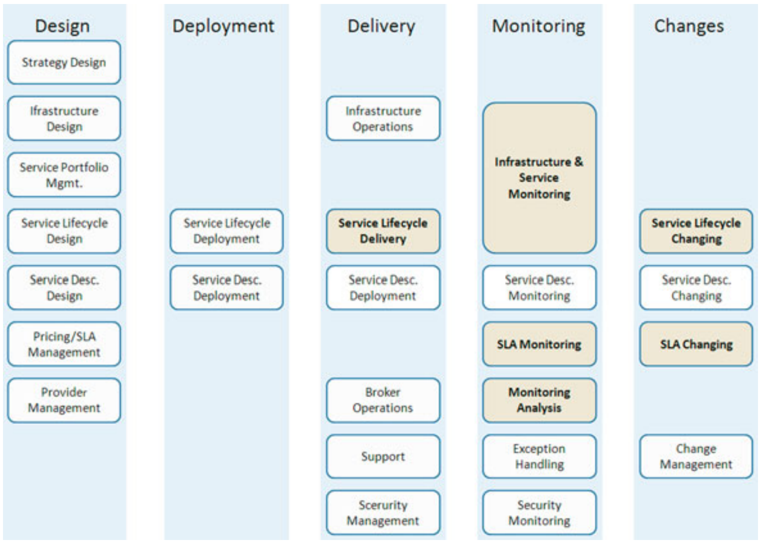


Fig. 1 SOA governance processes [32]

aspects of system operation. This goal is absolutely desirable, leading to adaptive systems which operate without (or with limited) human intervention. One of the emerging paradigms aimed at reducing effort involved in deploying and maintaining complex computer systems is called Autonomous Computing (AC) [37]. This paradigm might be applied in a very natural way to the development of adaptive SOA systems.

Autonomous Computing applications share some common properties [39] enabling them to properly apply this paradigm. Such properties aim to clarify the relation between adaptability and AC, and can be summarized as follows:

- **Adaptability**—the core concept behind adaptability is the general ability to change a system's observable behavior, structure or realization. This requirement is amplified by automatic adaptation that enables a system to decide about adaptation by itself, in contrast to ordinary adaptation, which is decided upon and triggered by the system's environment (e.g. users or administrators).
- **Awareness**—closely related to the adaptation and the execution context. It is a prerequisite of automatic adaptation. The term "context" is defined as sufficiently exact characterization of the situations in which a system might find itself by means of perceivable information relevant for the adaptation of the system. Awareness has two aspects: self-awareness (enabling a system to observe its own system model, state, etc.) and awareness of the environment.
- **Monitoring**—since monitoring is often regarded as a prerequisite of discovery and response to emerging events, it constitutes a system awareness. Monitoring indicates the system's state and thus characterizes a situation in which adaptation is necessary.

- **Dynamicity**—encompasses the system’s ability to change during runtime. In contrast to adaptability this only constitutes the technical facility of change. While adaptability refers to the conceptual change of certain system aspects, which does not necessarily imply the change of components or services, dynamicity is about the technical ability to remove, add or exchange services and components.
- **Autonomy**—as the term Autonomous Computing already suggests, autonomy is one of the essential characteristics of such systems. AC aims at unburdening human administrators from complex tasks, which typically require a lot of decision making and problem solving without human intervention.
- **Mobility**—mobility enables dynamical discovery and usage of new resources, recovery of crucial features etc.
- **Traceability**—traceability enables the unambiguous mapping of the logical architecture onto the physical system architecture which facilitates easy deployment of necessary measures. Autonomous Computing may help reduce this effort by allowing administrators to define abstract policies and then enable systems to configure, optimize and maintain themselves according to the specified policies. The notion of traceability is again closely related to that of adaptation: adaptation decisions are also based on an abstract system model.

The relation between AC properties and the service orientation principles is presented in Table 1. The AC adaptability is supported by all principles, but most significantly by P2 and P4. AC monitoring is strongly related to P7—Service Discoverability, as it is a prerequisite of monitoring activity. AC dynamicity is very much related to P8 and P2. Service composability represents more flexible relation than integration and allows for services to be interconnected at runtime. The next AC property, mobility, is supported by P4 and partially by several other principles. This property is very much in line with service orientation. The same concerns AC Traceability which is supported to some extend by P3. Services have abstract descriptions which allows for abstract model construction and mapping of the logical system architecture onto the physical one. Unfortunately, support for specification of abstract policies which

Table 1 Autonomic Computing Properties vs. Service Orientation Principles

Autonomic Computing	Service Orientation Principles							
Properties	P1	P2	P3	P4	P5	P6	P7	P8
Adaptability	+	++	+	++	+	+	+	+
Awareness								
Monitoring	+						++	
Dynamicity		+			+	+	+	++
Autonomy								
Mobility	+	+		++		+	+	
Traceability		++						

P1-Standardized Service Contract, *P2*-Service Loose Coupling, *P3*-Service Abstraction, *P4*-Service Reusability, *P5*-Service Autonomy, *P6*-Service Statelessness, *P7*-Service Discoverability, *P8*-Service Composability

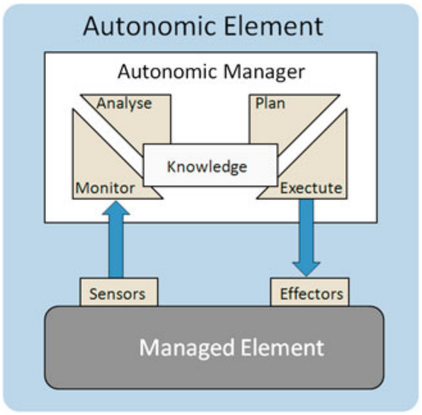


Fig. 2 IBM's MAPE-K reference model for autonomous control loops [1]

would enable systems to configure, optimize and maintain themselves according to the specified policies, is not offered directly by service orientation principles.

Two AC properties—Awareness and Autonomy—have no direct service orientation counterparts. Awareness is related to the ability of a system to observe its own system model, state, etc. This feature is not required by any service orientation principles. The AC Autonomy property has a different meaning than P5—Service Autonomy. While the former concept concerns the ability to make decisions without human intervention, P5 instead refers to isolation of the execution environment of a service. This consideration leads to an important observation, namely that implementation of SOA self-adaptive systems requires both AC Awareness and AC Autonomy. Support for AC Traceability should also be provided where possible.

To achieve autonomous computing IBM has suggested a reference model for autonomous control loops [1], which is sometimes called the MAPE-K (Monitor, Analyse, Plan, Execute, Knowledge) loop and is depicted in Fig. 2. This model is used to express the architectural aspects of autonomous systems.

In the MAPE-K autonomous loop the managed element represents any software or hardware resource that is given autonomous behaviour by coupling it with an autonomous manager. This element is equipped with: (i) Sensors, often called probes or gauges, which collect information about the managed element, and (ii) Effectors, which carry out changes in the managed element.

The data collected by the sensors allows the autonomous manager to monitor the managed element and execute changes through effectors. The autonomous manager is a software component that can be configured by human administrators using high-level goals and uses the monitored data from sensors and internal knowledge of the system to plan and execute the low-level actions deemed necessary to achieve high-level goals. The internal knowledge of the system is often an architectural model of the managed element. The goals are usually expressed using event-condition-action (ECA) policies, goal policies or utility function policies [78].

There are many diverse implementations of the MAPE-K loop:

- Autonomous Toolkit [31]—developed by IBM. It provides a practical framework and reference implementation for incorporating autonomous capabilities into software systems.
- ABLE [9]—another toolkit proposed by IBM which provides autonomous management in the form of a multiagent architecture: each autonomous manager is implemented as an agent or set of agents,
- Kinesthetics Extreme [33]—this work was driven by the problem of adding autonomous properties to legacy systems, i.e. existing systems that were not designed with autonomous operation in mind.
- 2K [40]—represents an autonomous middleware framework and offers self-management features for applications built on top of this framework.

It is necessary to point out that none of these systems addresses or takes advantage of the SOA paradigm. To better explain the technical aspects of the MAPE-K loop activities, we will consider them in more detail.

Monitoring

Monitoring involves capturing the managed element state or properties of the environment. Two types of monitoring can be distinguished: (i) Active Monitoring which requires instrumentation of software or hardware at some level, for example by modifying and adding code to the implementation of the application or the operating system in order to capture function or system calls; (ii) Passive Monitoring which relies on already built-in system capabilities for presenting information about their operation, e.g. system logs, load monitors, etc.

Analysis

Analysis is rather straightforward and depends on the planning activity specified below. It may concern data filtering or recognition of the managed element state.

Planning

Planning takes into account the monitoring data from sensors to produce a series of changes to be effected on the managed element. ECA rules, already mentioned in this section, that directly produce adaptation plans from specific event combinations, could be used in the simplest case. Rule-based planning determines the actions to take when an event occurs and certain conditions are met. This type of planning (referred to as Policy-Based Adaptation Planning) usually does not take into account system history and is therefore stateless. More advanced planning known as Architectural Models acknowledges a model of the system in the form of a connected component network. This allows users to ascribe constraints and properties to individual components and connectors. Violation of these constrains triggers adaptation actions. The third type of planning involves the Process-Coordination Approach where adaptation tasks result from defining the coordination of processes executed in the managed elements.

Execution

This activity does not merit special attention as it concerns technical issues related to execution of adaptation tasks. The managed element effectors are used for this purpose.

The presented analysis of self-adaptive systems design with special attention to AC Systems led to the definition of an adaptive systems space. This space covers three directions: (i) how the adaptation process is executed, (ii) where the adaptation mechanisms are located, (iii) when the adaptability mechanisms are added to the system. Definition of the adaptive systems space (depicted in Fig. 3) enables us to show where the adaptive SOA system investigated in this chapter can be located. The properties of such a space could be summarized as follows: the adaptation process is executed automatically; adaptation mechanisms are located in the middleware or infrastructure layer; adaptation mechanisms are added during deployment or at runtime.

The reference model which explains the role of Infrastructure and Middleware Layers in Adaptive SOA is shown in Fig. 4. The Infrastructure Layer is interpreted in a rather broad sense as it combines not only physical resources such as typical servers connected to computer networks, but also small physical devices (Real-World Devices) such as mobile phones, intelligent sensor networks, etc. with pre-installed

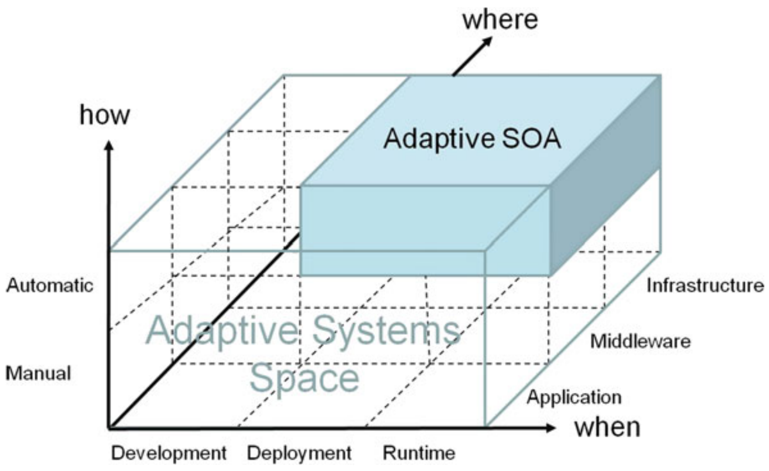


Fig. 3 Adaptive SOA vs Adaptive Systems Space

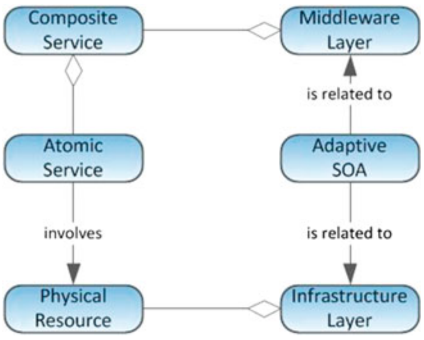


Fig. 4 Concept map showing Adaptive SOA

systems or embedded software/firmware. The Infrastructure Layer could be exposed as a set of virtual resources for higher layers, e.g. Middleware.

The Middleware Layer provides all of the system services that are relevant to the service orientation principles and their implementation. From an abstract point of view the Middleware Layer serves as a container for Composite and Atomic Services. These services require physical resources to perform their function.

3 Real-World Service

As shown in Fig. 5, a physical resource can be used by a computer or a Real-World Device. In the first case, through a computer, Virtual Services can be provided. In the second case, through appropriate instrumentation of Real-World Device it is possible to create and provision a variety of Real-World Services. However in both cases these are atomic services which are created composite service.

3.1 Concept

The Real-World Service [26] is a feature of the physical world object (Real-World Device), which provides an embedded logic module and a communication module, and is exposed to external systems according to service orientation principles. Such a service allows other components to interact with it dynamically. In contrast to virtual services (such as enterprise services) real-world services provide data about physical things/devices (World of Things) in real time. The Real-World Device is equipped, by means of hardware extensions, with logic and monitoring/management features.

The use of real-world services is associated with the concept of real-world awareness which is defined as follows [29]:

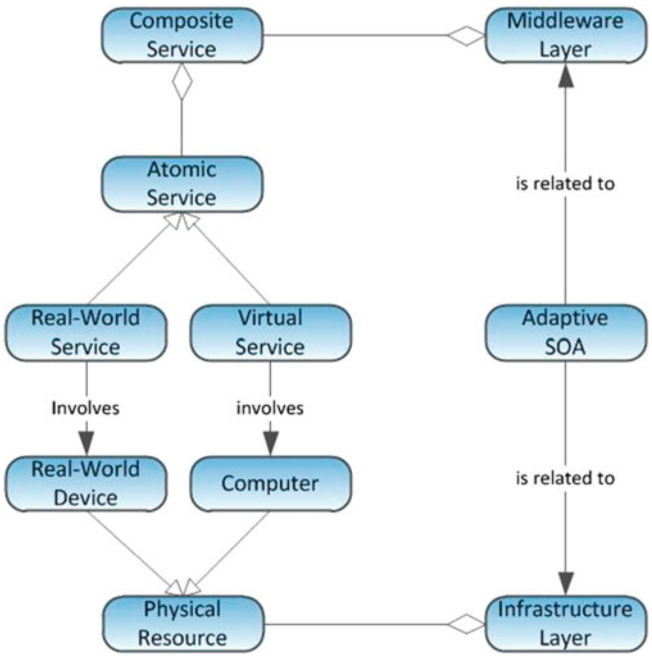


Fig. 5 Concept map introducing Real-World and Virtual Services

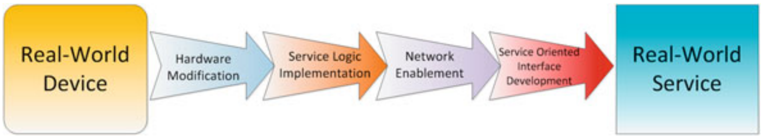


Fig. 6 Process of transforming a Real-World Device into a Real-World Service

> Real World Awareness is the ability to sense information in real-time from people, IT sources, and physical objects—by using technologies like RFID and sensors—and then to respond quickly and effectively.

The Real-World Service is created by adding logic to a physical object (Real-World Device). Network communication and adaptability allow users to obtain information about such an object in real time, and perform suitable management. It seems that the idea of Real-World Services complements the concept of real-world awareness in terms of the ability to sense information in real time, from a variety of physical objects.

Before any Real-World Devices can be used by enterprise systems, they must be modified into Real-World Services. This process adds aspects of service orientation to the Real-World Device and exposes the functionality of the device in the form of a service. In a general case the modification process can be divided into the following stages (Fig. 6):

1. Hardware modification—this calls for addition of the necessary mechanical parts and actuators to the device, as well as low-level protocol extensions by adding various electronic circuits so as to enable digital control.
2. Service logic implementation—the goal in this step is to build a dedicated server embedded in the augmented device created in step 1. The added logic allows exposure of Real-World Device features.
3. Network enablement—this step involves augmenting the device with a communication module, typically implementing the TCP/IP protocol stack.
4. Service oriented interface development—aimed at implementation of the service-oriented protocol stack (for example, SOAP/REST-based Web Services) to expose the functionality of the server to individual clients or enterprise applications.

Depending on the nature of the Real-World Device, some of the above steps might be simplified or skipped.

3.2 Realization

The first stage of the modification process mostly depends on the nature and properties of a Real-World Device. There are many devices in our environment that are intended to be used and controlled manually. These devices usually require mechanical and electronic modification to be connectable to external computer systems. First of all, such modifications may require adding extra components and mechanical parts (such as servomotors) to the device if automatic operation is required. Another typical augmentation is a computing chip that may need to be added to enable automatic changes of the device's mechanical or logical state. This could be a processor-based microcontroller or an FPGA programmable logic matrix. It should also be noted that some devices may already possess inbuilt features which allow a local embedded system to control the device in question. In such cases the device may only require some electronic and electrical modifications. The goal of all these modifications is to expose the full functionality of the Real-World Device on the level of digital logic signals. In some situations there is no need for any modification. In such cases the Real-World Device is already equipped with the proper (simple or advanced) digital interface and is ready to be controlled by external systems. For this type of devices the first stage can be completely skipped.

The second stage of the modification process depends strongly on the selected techniques and hardware used for implementation of the Real-World Device processing logic, such as:

- full hardware implementation (e.g. inside an ASIC or FPGA chip),
- IP core processor (with or without hardware accelerated parts) inside an FPGA chip,
- general-purpose microcontroller (with or without abstraction layers),
- mobile device (e.g. smartphone).

The first approach assumes that service logic and high-level communications required for service exposure will be implemented as "pure hardware". This implies that the service operation algorithm must be converted into a set of properly connected logic gates and flip switches, and then embedded in a programmable logic chip (such as FPGA or ASIC). The use of this technique is very difficult and requires substantial experience (especially in the area of digital circuit design), but in return offers efficiency and performance which is unreachable via any other technique. A description of a sample service implementation exploiting this approach can be found in [59]. The authors present the design of a "system on chip" (SoC) which operates as a Web service (WS). Their proposed system is entirely devoid of software and conceived as a hardware pattern for trouble-free design of network services offered as WS in a service oriented architecture (SOA).

The second approach is very similar to the first one as it also assumes the use of an FPGA device in the implementation of service logic and high-level communication, albeit in a completely different way. The FPGA device delivers a hardware platform for a soft-core (virtual) processor with capabilities adjusted to the requirements of the implemented service. The algorithm of operation is implemented as a program that runs on that processor. In the course of service implementation it is also possible to adjust the configuration of the soft-core processor for specific requirements of the implemented service. One example of such an approach is the hardware-software integrated development platform presented in [65]. The platform is based on the ALTERA Stratix II EP2S60 FPGA chip and dedicated to create SOA-compatible image processing services. The core features of such services are implemented as sequential C++ programs executed on the Nios II soft-core processor, while the most computationally expensive image processing operations are offloaded to a hardware accelerator. An additional advantage of the first two approaches is the ability to use the same flexible hardware equipment to implement completely different services.

The third approach assumes the use of a general-purpose microcontroller. In this case, the hardware architecture is delivered by the microcontroller manufacturer. Service logic and exposure are handled by software that runs on the microcontroller. Many modern consumer electronics (called smart or intelligent devices) are now equipped with capabilities used to share their functionality with PDAs, smartphones or other mobile devices (e.g. an electronic scale equipped with a Bluetooth interface for collecting and exchanging weight data). Unfortunately, the communication protocols used in this scope are usually incompatible with the protocols used for service exposure, and it is often impossible to introduce any modification inside the device itself. Fortunately, modern mobile devices (PDAs, smartphones) usually provide wireless Internet access (through Wi-Fi, GPRS, HSPA or LTE protocols)—hence the fourth scenario for modification and conversion of Real-World Devices into Real-World Services is to use the mobile device as a kind of proxy between the smart device and enterprise systems. In this scenario the service logic and high-level communication protocols are implemented as software running on a mobile device.

To enable the Real-World Device to be controlled over the Internet by enterprise systems (the 3rd stage of the modification process), the device must be equipped with a communication module. There are many such modules on the market—examples include the DigiConnect network module [19] and Tibbo programmable embedded modules [74]. Both solutions come fitted with hardware- or software-based TCP/IP stacks and Wi-Fi communications. While the DigiConnect module only offers one socket connection session at the time, the Tibbo modules are much more robust in that they allow up to 16 concurrent socket communication sessions. Having more than one socket session allows us to add extra features to the Real-World Service, such as service discovery.

An important concept for enabling service orientation on Real-World Devices is contained in the Device Profiles for Web Service specification (DWPS) [52] which is the successor of Universal Plug and Play (UPnP). This technology was developed to enable secure Web Service capabilities on resource-constrained devices. DPWS was mainly developed by Microsoft and some printer device manufacturers. DPWS allows secure messages to be sent to and from Web Services. It also supports dynamic discovery of Web Services, Web Service descriptions, as well as subscribing to and receiving events from a Web Service.

Web Services for Devices (WS4D) is an initiative which brings Service-Oriented Architecture (SOA) and Web Services technologies to the application domains of industrial automation, home entertainment, automotive systems and telecommunication systems. WS4D advances results from the ITEA SIRENA project [84]. The WS4D toolkits available on the project's website complies with DPWS. The toolkit is based on gSOAP and is targeted for small resource-constrained devices and can be used to implement DPWS-compliant devices using the C programming language. Another toolkit, based on J2ME, is available for small and resource-constrained devices, enabling implementation of DPWS-compliant devices in Java. Yet another toolkit, based on Apache Axis2, is targeted for resource-rich implementations to connect DPWS-compliant devices with the Web Services world.

In the SOCRADES (Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices) project [34] physical legacy devices are grouped into three categories: non-electronic devices that are not WS-capable, electronic devices which do not support WS due to their limited resources, and WS-capable devices. To expose the features of WS-capable devices DPWS profiles are used. For devices which are not WS-capable, this can be done in two ways: by using the Gateway (dedicated for non-WS-capable electronic devices) or the Service Mediator (originally designed for collecting data from non-electronic devices).

Recently significant effort has been invested in enabling the convergence of sensor networks with the IP world and providing Internet connectivity for "smart objects". The IETF Working Group IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) proposed an RFC [47] to enable IPv6 packets to be carried over IEEE 802.15.4. In addition, the IETF Working Group Routing over Low power and Lossy networks (ROLL) designed a routing protocol named IPv6 Routing Protocol for Low power and Lossy Networks (RPL). RPL was proposed because none of the existing known protocols such as Ad hoc On-Demand Distance Vector (AODV), Optimized Link State Routing (OLSR) or Open Shortest Path First (OSPF) meet the specific requirements of Low power and Lossy Networks (LLN), see [70]. The RPL protocol targets large-scale wireless sensor networks (WSN) and supports a variety of applications e.g. industrial, urban, home and building automation or smart grids.

The Constrained Application Protocol (CoAP) [68] is a specialized web transfer protocol for use with constrained networks and nodes for machine-to-machine applications such as smart energy and building automation. These constrained nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while networks such as 6LoWPAN often have high packet error rates and a typical throughput of several dozen kbit/s. CoAP provides a method/response interaction model between application endpoints, supports built-in resource discovery, and includes key web concepts such as URIs and content-types. CoAP easily translates to Hypertext Transfer Protocol (HTTP) for integration with the web, while also meeting the specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

Another approach to combining Real-World Services with Virtual Services is to generalize the Open Services Gateway initiative (OSGi) model into a collection of loosely coupled software modules interacting through service interfaces. While OSGi is a Java based solution, in [63] authors discuss how to turn non-Java-capable devices and platforms into OSGi-like services. They propose an extension to Remote Services for OSGi (R-OSGi) which makes communications to and from services independent of the transport protocol, and implement an OSGi-like interface that does not require standard Java (or even any Java at all). As an example, implementations for Connected Limited Device Configuration (CLDC), embedded Linux, and TinyOS are presented.

The idea behind modern network systems is to use the Internet as a connection space for as many Real-World Devices as possible. Each device, modified to become a Real-World Service, has its own reference address on the Internet and can be contacted by others systems. However, the large number of Real-World Services makes it difficult for users to locate any specific service. To make it easier, a Real-World Service registration and discovery features are required. Each of the available Real-World Services regularly sends information about its reference and description to one or more discovery servers which maintain a database of active Real-World Services on the network (registration process). The information needed to find and use the various Real-World Services is available through the discovery server (discovery process). Among the various technologies that provide service discovery we can mention SLP (Service Location Protocol), Jini, Apple Bonjour and WS-Discovery in DWPS [27].

4 Realization of the Adaptation Loop

This section describes the current realization status of the Adaptation Loop and highlights solutions capable of converting Managed Resources into Virtual Services (as well as Real World Services).

The MAPE-K model introduces five elements of an autonomous loop which, taken together, support development of completely autonomous systems. In order to describe an adaptation loop it is enough to use only four elements: Monitor, Analyse, Plan and Execute (later referred to as MAPE). Figure 7 presents techniques involved in realization of the adaptation loop. They are divided into those related to Monitoring and Execution, and those applicable to Analysis and Planning. The former techniques are used for instrumentation of Managed Resources for the purpose of adding Sensors and Effectors, while the latter are used for interpretation, analysis of data provided by Sensors and planning actions executed by means of Effectors.

The first subsection will present a review of techniques used for Monitoring and Execution, while the second subsection describes techniques of Adaptation and Planning. The final subsection contains a survey of existing work presented in the context of identified techniques.

4.1 Monitoring and Execution

As presented in Fig. 7, the Managed Resource can be either Virtual (related to a Virtual Service) or Real-World (related to a Real-World Service); however in both cases it can be composed of both Software and Hardware elements. Instrumentation techniques are therefore divided into two categories. Software instrumentation is mostly related to the Middleware Layer while hardware instrumentation ties in with the Infrastructure Layer. As presented in Fig. 5 a Real-World Service always involves a Real-World Device, which makes hardware instrumentation especially important for adding adaptability to resources related to the real world. Software instrumentation is commonly used in the case of both Virtual and Real-World Resources.

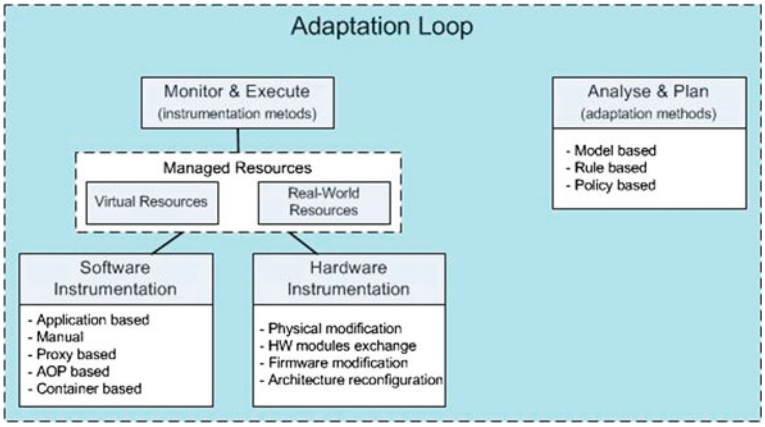


Fig. 7 Adaptation loop realization techniques

4.1.1 Software Instrumentation

One of the key approaches applied to software instrumentation is the Interceptor Design Pattern [4]: a design pattern used when software systems or frameworks need to offer a way to change, or augment, their usual processing cycle. The Interceptor Design Pattern may solve several problems related to software development. For example, it is commonly used for monitoring the internal execution of an application. Another area of usage is the ability to change or extend application behaviour. New features are implemented as interceptors and invoked by the working application. Such extensions do not need to be aware of other parts of the application, nor change existing parts. They also do not affect the design of the system.

The Interceptor Design Pattern may be realized with the use of various approaches, for instance with the Proxy Pattern where the original object is replaced with a proxy with the same interface (or contract) as the original. Usage of this pattern provides several possibilities, such as controlling access to the proxied object or lazily loading such objects. The Proxy Pattern also provides the possibility to monitor original object invocations or even alter them. It is also possible to create a proxy-based solution which enables plugging in of sensors and effectors, facilitating transparent software instrumentation.

Another approach for software instrumentation compliant with the Interceptor Design Pattern is called container-based instrumentation. In contrast to previously described solutions, in this case the subject of instrumentation is not an application per se, but rather a container or execution environment in which the application is being executed. The lowest level of such instrumentation is enrichment of the virtual machine in which the process is executing. Examples of such systems are presented in [24, 82] where Java Virtual Machine (JVM) is instrumented in order to access runtime monitoring data. However, this approach is somewhat dated: nowadays similar techniques are mapped into the domains of services and components. In this case the subject of instrumentation is the container in which services are connected, exposed and able to communicate—such as Service Component Architecture (SCA) or Enterprise Service Bus (ESB). In the case of SCA, instrumentation may be applied to enable exchanging composites from which the service is created. In this way it is possible to manage the Quality of Service (QoS) of services which are executed in the container. In the case of the ESB container, instrumentation may be used to relay service calls to one of several instances of a particular service in order to provide the desired QoS.

One of the main programmatic mechanisms used to realize software instrumentation is the concept of Aspect-Oriented Programming (AOP), first introduced in [38]. In this approach existing software can be interwoven with aspects which change or extend its behaviour in a fully transparent way. This approach is especially useful when the source code does not admit modifications. Initially, aspects were woven into the software during compile or load time and any changes required the software to be stopped. To solve this issue the concept of dynamic aspect weaving was introduced. In [5, 56, 57] the authors presented several approaches to weaving and unveawing aspects at runtime, providing a basis for using aspects in order to realize the adaptation control loop.

It is also possible for the software itself to be written in such a way as to enable instrumentation. However, this solution is strongly limited to mechanisms exposed directly by the software, i.e. retrieving monitoring data via dynamically added sensors. As such, if some features or mechanisms are not enabled, it becomes necessary to use one of the techniques described above.

4.1.2 Hardware Instrumentation

Instrumentation and introduction of new features, including sensors and effectors, to hardware resources may require modifications in the physical design of electronic modules. Modifying and reconfiguring embedded systems in order to add new features might be applied on different levels of design, starting from firmware reconfiguration, through modification of external peripherals and microprocessor design, all the way to selection of hardware modules that the system is composed of. In general, these possibilities can be classified as:

1. Physical modifications—modifications that require additional elements to be installed or modules to be exchanged.
2. Embedded software modifications—modifications performed by software reconfiguration, e.g. changes to firmware or configuration elements.

Some embedded systems can be designed in a modular way, enabling new features (e.g. measurement and debugging elements) to be added by plugging additional modules into the existing platform. In other solutions it might be necessary to solder additional elements manually. Such modifications cannot be performed automatically as they require physical intervention. As programmable logic devices become more and more robust, software modifications might be perceived, to some extent, as physical modifications since they involve modifications in the embedded system's internal architecture. Upon installation effectors may use the same hardware reconfiguration techniques to affect hardware resources during the adaptation process.

In the simplest case, embedded systems might be reconfigured by replacing the firmware of the internal processor causing a change in its functionality and enabling additional features to be used in the adaptation loop. This kind of reconfigurability mirrors typical standalone systems. A more promising approach is to reconfigure the hardware of the embedded system. The concept of reconfigurable computing that combines some of the flexibility of software with the high performance of hardware processing using high-speed reconfigurable computing fabrics, has existed since the 1960s. Gerald Estrin's landmark paper proposed the concept of a computer composed of a standard processor and an array of "reconfigurable" hardware components [21]. The main processor would control the behavior of the reconfigurable hardware which would, in turn, be tailored to perform a specific task, such as image processing or pattern matching, with performance similar to a dedicated hardware platform. Once the given task was completed, the hardware could be adjusted to perform some other task. This results in a hybrid computer architecture, combining the flexibility of software with the speed of hardware. Unfortunately, Estrin's idea was far ahead of its time given the sophistication of electronic devices. In the 1980s and 1990s there was a renaissance in this area of research, with many proposed reconfigurable architectures developed in the industry and academia [11], such as COPACOBANA, Matrix, Garp, Morphosys and PiCoGA [44]. Such designs became feasible due to the relentless progress in silicon-based technologies which finally allowed complex designs to be implemented on a single chip. The world's first commercial reconfigurable computer, Algotronix CHS2X4 [36], was completed in 1991. Algotronix was designed as a low-cost add-on card for the PC. It contained 9 programmable CAL1024 logic chips. The computer found application in a number of areas, including self-reconfiguration, in which the board reconfigured itself from a design stored in RAM as a result of computation. Ultimately the CHS2X4 would not achieve commercial success, but proved promising enough that its core technologies were later bought by Xilinx—inventor of the Field-Programmable Gate Arrays.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—akin to changeable logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. The main difference compared with custom hardware, i.e. application-specific integrated circuits, is the ability to adapt the hardware at runtime by loading a new circuit on the reconfigurable fabric. This can be achieved through partial reconfiguration, by configuring a portion of a field programmable gate array while another part is still running or operating. Much like software, hardware can be designed modularly, by first creating subcomponents and then higher-level components. In many cases it is useful to swap out one or several subcomponents while the FPGA is operating.

Normally, reconfiguring a FPGA requires it to be held in reset mode. While in that mode an external controller reloads the new design into the FPGA. Partial reconfiguration allows for critical parts of the design to continue operating while a controller (either inbuilt or external) loads a partial design into the reconfigurable module. Partial reconfiguration also can be used to save space for multiple designs by only storing the partial designs that change between sessions. A common situation in which partial reconfiguration might be useful is the case of a communication device. If the device controls multiple connections, some of which require encryption, it would be beneficial to load different encryption cores without bringing the whole controller down. In the scope of design functionality, partial reconfiguration can be divided into two categories:

- dynamic partial reconfiguration, also known as active partial reconfiguration—allows changing parts of the device while the rest of the FPGA is still running;

- static partial reconfiguration—the device is not active during the reconfiguration process. While partial data is sent to the FPGA, the rest of the device is stopped (in shutdown mode) and brought up once the configuration is completed.

There are two styles of partial reconfiguration of an FPGA:

- Module-based partial reconfiguration, which enables reconfiguring distinct modular parts of the design. To ensure communication across reconfigurable module boundaries, special bus macros ought to be prepared. A macro works as a fixed routing bridge that connects the reconfigurable module with the remainder of the platform. Module-based partial reconfiguration requires a set of specific guidelines to be followed at the design stage.
- Difference-based partial reconfiguration, which can be used when a small change is introduced in the design. It is especially useful when exchanging small routines or dedicated memory blocks. The partial bit-stream contains only information about differences between the current design structure (which resides in the FPGA) and the new content of an FPGA.

Maturation of Field Programmable Gate Arrays has led to the concept of open-source hardware, where physical artifacts are designed and offered in the same manner as free and open-source software. Hardware design (i.e. mechanical blueprints, schematics, materials, printed circuit layout data, hardware description language source code and integrated circuit layout data), in addition to the software that drives the hardware, are all released using the open source approach. Since the rise of reconfigurable programmable logic devices, sharing of logic designs has adopted the concept of open-source hardware. Instead of raw schematics, hardware description language (HDL) code is shared. HDL descriptions are commonly used to set up system-on-a-chip platforms using either field-programmable gate arrays (FPGAs) or application-specific integrated circuit (ASIC) designs. HDL modules, when distributed, are called semiconductor intellectual property cores, or IP cores. This allows developers to exploit general purpose electronic modules along with specific hardware descriptions [53] to build custom hardware devices.

4.2 Analysing and Planning

Several approaches to constructing autonomous managers are currently under investigation. One of these is to apply control theory for performance control in complex applications such as real-time scheduling, web servers, multimedia, and power control in CPUs [42]. This methodology is viewed as a promising foundation but its main drawbacks include the need to identify and model managed systems. A different approach is to use policies to guide decisions based on the observed system state and its behaviour. Several languages and specifications have been designed for this purpose. The WS-Policy [75] specification represents a set of specifications that describe the capabilities and constraints of security (and other business) policies on intermediaries and endpoints (this includes e.g. the required security tokens, supported encryption algorithms, and privacy rules) and how to associate policies with services and endpoints. The Ponder [18] language provides common means of specifying security policies that map onto various access control implementation mechanisms for firewalls, operating systems and databases. Understanding of the policies varies between researchers, but usually the term "policy" is used to represent a set of considerations which guide decisions. The policy is provided to the system as a set of rules. A policy information model provides an abstract representation of the information required to define a policy:

- Condition-Action Information Model policies consist of several policy rules that have two elements: conditions and actions. Conditions define when the policy should be applied while Actions define what needs to be done when a particular policy rule is applied. Condition-action policy rules assume the following form: if [list of conditions] then [list of actions]. Such policies are evaluated at regular intervals. Even though this information model is very simple, it has some drawbacks. The frequency of policy evaluation has to be defined, which may influence system reaction time.
- Event-Condition-Action Information Model treat events as conditions. This type of policy model is useful for asynchronous policy evaluation as a response to events generated by the system (e.g. state changes or parameters exceeding threshold values). An event-condition-action policy is denoted as when [list of events and conditions] then [list of actions]. This type of policy becomes particularly useful when determining policy evaluation frequency becomes unfeasible or when event-driven policies are called for.

Policy rules are executed by a rule engine which uses algorithms for efficient pattern matching. Usage of rule engines brings a number of advantages, including separation of business logic from application implementation, the ability to change policies without software recompilation and extending the set of policies at runtime.

Another promising approach for runtime adaptation is to apply mechanisms which leverage software models and the applicability of model-driven engineering techniques to the runtime environment. A runtime model is a causally-connected self-representation of the associated system that emphasizes the structure, behaviour, or goals from a problem space perspective [10]. Models may express several different aspects of the running system. They might express the structure of the underlying system or its behavioural aspects. Structural models show how the system is constructed in terms of objects and invocations. In contrast, behavioral models emphasize how the system executes in terms of flows or traces and events occurring in the working system.

4.3 Survey on Existing Solutions

This section presents a comparison of existing solutions in the context of adaptation loop techniques described previously. Such comparison is performed in order to show the current state of the art concerning instrumentation and adaptation methods in different projects, and to evaluate the prospects of applying them to the SOA domain. The results of this research are presented in Table 2.

As described in the previous sections, several software instrumentation techniques exist. The following paragraphs present selected solutions and their usage in the context of adaptation loops.

In [23] the authors present a framework called Rainbow which can be used for adaptation of software systems. Such adaptation concerns the use of mechanisms located outside of the analyzed system, which enable adaptation strategies to be specified. The Rainbow framework enforces the implementation of its sensors and actuators, thereby enabling adaptation. This solution can be used for systems built in accordance with the SOA paradigm thanks to its modular and flexible architecture. In order to realize an adaptation loop an architectural model of the system has to be prepared. On the basis of this model Rainbow enables adaptation invariants and strategies to be specified. Together, these determine the system's adaptation style and can be evaluated by the adaptation engine. Rainbow is aimed at virtual services, however any set of adaptation strategies created for a hardware adaptation can be used after reorganizing process for the purpose of real-world services.

In [48] the authors present results of their work in the DiVA project which focuses on dynamic variability in complex adaptive systems. DiVA introduces a methodology and tools for runtime QoS management of adaptive systems. It also proposes an approach for specifying and executing dynamically adaptive software systems which combines model-driven and aspect-oriented techniques in order to tame the complexity of such systems. This approach depends on the model of the managed system. In order to apply the proposed approach engineers need to design models independently of the running system and leverage them at runtime to drive the dynamic adaptation process. The DiVA solution does not directly address the SOA domain, but—owing to its flexible architecture—could be easily incorporated into SOA systems. The proposed techniques can be applied to Virtual as well as Real-World Services; however no hardware adaptation metodologies have been published so far.

In [25] the authors describe the Adaptive Server Framework (ASF)—an architectural concept which facilitates the development of adaptive behavior for legacy server applications. ASF provides clear separation between the implementation of adaptive behavior and the business logic of the server application. ASF incurs low CPU overhead and memory usage. It is portable across different J2EE applications servers. The goal of ASF is to provide infrastructure components and services to facilitate the construction of behavioral adaptation. ASF components interact with the application server, monitor the runtime environment, analyse collected data and change the application's behavior by adapting its responses or setting the server's configuration to fulfill business goals. The authors identify two monitoring techniques: adding

Table 2 Instrumentation and adaptation methods used by existing solutions

Solution	Software instrumentation						Hardware instrumentation			Analyse and Plan		
	App. based	Manual	Proxy based	AOP based	Cont. based	Physical mod.	Modules exchange	Firmware mod.	Arch. reconf.	Model based	Rule based	Policy based
[23]			X								X	X
[48]				X						X		
[25]		X	X							X		X
[28]	X									X		
[81]		X								X	X	
[14]		X			X					X		
[43]					X					X		
[79]		X			X					X		
[83]	X	X										
[7]			X									
[6]		X	X		X							
[49]				X	X							
[80]					X							
[60]				X	X							
[55]				X								
[71]				X								
[13]				X								
[41]							X					
[46]									X			
[16]									X			
[12]								X	X			
[50]							X	X	X			
[69]							X	X				
[2]								X				

interceptors by means specific for a given service container (which can be qualified as container-based instrumentation) and wrapping components in JMX MBeans and redirecting client invocations (a combination of proxy and manual instrumentation techniques). In ASF adaptation analysis and planning involve a component model of the system, which can be changed during adaptation, and rely on policies which drive the adaptation process. The proposed approach focuses on components (i.e. the components layer of S3) and can therefore be perceived as partly related to SOA. ASF does not deal with Real-World Services or hardware instrumentation aspects.

In [28] the authors propose a middleware-centric approach to building applications capable of adapting to dynamically changing requirements, pursued in the FAMOUS (Framework for Adaptive Mobile and Ubiquitous Service) project. The authors focus on handheld devices where communication bandwidth or UI preferences change dynamically, depending on the ambient light and noise. The adaptation loop is realized in the following way: when a context change occurs, it is detected by the context monitor which then notifies the adaptation manager. The adaptation manager searches for a configuration which best fits the current context and resource utilization by the application. The search uses a planner component to iterate through plans for all possible application variants. A plan is generated by selecting a specific component for each component role of the application. As some of the selected components can be composites, the planning continues recursively until all leaf nodes are selected. The best configuration is selected by computing a utility value for each plan with respect to the user preferences and properties of the execution environment. This value is returned by the utility function, defined by the developer. The utility funciton is typically a weighted mean of the differences between the offered and required properties. Individual weights in the utility function represent changing user priorities and may be adjusted at runtime. The variant with the highest utility value is chosen. In order to avoid constant changes, the adaptation manager also needs to evaluate whether the perceived improvement is high enough to justify an adaptation. Such evaluation is based on a user-adjustable utility improvement threshold and adaptation delay.

In [81] the authors describe three important steps towards adaptive online systems. First is the assumption that online hardware reconfiguration due to workload changes has the potential to improve performance. Second step assumes that by using uninstrumented middleware and given only raw, low-level system statistics it is possible to predict which of the two configurations will outperform the other at any given time. The last one imposes extending the prediction capability to make precise numerical estimations (i.e. quantitative changes in performance when the system is switched to each of the possible configurations). By fulfilling all three criteria performance gains can be traded off against inevitable reconfiguration costs. The authors start off with a set of experiments using the TPC-W benchmark which shows that given configurations might prove better in different circumstances Subsequently the authors claim that they can infer the optimal configuration on the basis of low-level operating system statistics (with no customized instrumentation). They create a model mapping the current system state (represented by output from the vmstat tool) to the optimal configuration. Using results from previous experiments as training data they apply the WEKA package as an implementation of standard machine learning methods.

In [14] the concept of Adaptable ESB is introduced. It consists of an operations support system that is compliant with NGOSS (Next Generation Operations System and Software) and implements a service-oriented architecture (SOA) that relies on an enhanced enterprise service bus (ESB). This enhanced ESB, referred to as an adaptable service bus (ASB), enables runtime changes to business rules, thus avoiding costly application shutdowns. An implementation of this system has been used by the ChungHwa Telecom Company, Taiwan, since January 2008 and provides complete support for its billing application. As a result the billing process cycle has been reduced from 10–16 days to 3–4 days, paving the way for further business growth.

The paper [43] propose the usage of ESB to build a dependable SOA middleware. Its authors exploit state-of-the-art solutions (i.e. Bayesian networks and fault detection algorithms) to propose a service-based architecture ensuring high dependability of business processes and services. The architecture comprehensively addresses the following challenges: discovering causal relationships, providing high scalability and preventing excessive overhead. The deficiency of the designed middleware is its reliance on the monitoring API of a particular ESB, which is not standardized among different vendors.

Morin et al. have studied the role of runtime models in managing runtime or dynamic variability [22]. Their research focuses on reducing the number of configurations and reconfigurations that need to be considered when planning adaptations of the application. The authors illustrate their approach with a customer relationship management application. Fleurey et al. present preliminary work on modeling and validation of dynamic adaptation [48]. Their proposed approach envisions runtime use of Aspect-Oriented Modelling (AOM). First, the application base and variant architecture models are designed and the adaptation model is built. At runtime the adaptation model is processed to produce the system configuration to be used during execution. Although adaptation to context changes is precisely described, the methods of adapting to changing QoS requirements and capabilities are only mentioned.

There are many different ways to extract monitoring data. Some studies rely on the data provided by the application layer [7, 83], while others focus on the mechanisms of monitored containers [43, 80] or turn to instrumentation of monitored systems [6, 49, 79]. Identifies two main kinds of monitoring data extraction: instrumentation and interception. Interception assumes that the monitored system enables some proprietary way of installing interceptors. Many of the frameworks relies on the AOP instrumentation which has been the subject of numerous studies (cf. [13, 55, 60, 71]).

In order to enforce adaptation in Real-World Services it is necessary to enable hardware instrumentation. Several interesting solutions related to this concept are currently available, as highlighted in the following paragraphs.

The concept of reconfigurable general-purpose hardware can be extended to cover replaceable hardware modules that can be selected for particular usage scenarios. One of the relevant open-source hardware startups is Bug Labs [41]. The company develops a Lego-like hardware platform which tinkerers and engineers can use to create their own digital devices. Development starts with BUGBase, which is a general-purpose Linux computer about the size of a PlayStation Portable, encased in white plastic. It provides four connectors that plug right into the motherboard. The company also manufactures a variety of modules that can plug into the computer—including an LCD screen, a digital camera, a GPS unit, a motion sensor, a keyboard, an EVDO modem, and a 3G GSM modem (There are also extensions for USB, Ethernet, WiFi, and serial ports). Bug Labs intends to produce approximately 80 different modules and hopes that external companies and developers will create their own modules.

A representative general-purpose hardware solution is marketed by the Milkymist [46] project. It is a comprehensive open-source platform for live synthesis of interactive visual effects. The project goes to great lengths to apply the open source principles at every level possible, and is best known for the Milkymist system-on-chip (SoC) platform, which is among the first commercialized system-on-chip designs with free HDL source code. As a result, several Milkymist technologies have been reused in applications unrelated to video synthesis. For example, NASA's Communication Navigation and Networking Reconfigurable Testbed (CoNNeCT) experiment uses the memory controller that was originally developed for the Milkymist system-on-chip in the development of an experimental software-defined radio prototype.

A polar opposite to full reconfiguration of the embedded system architecture is the concept known as Programmable System on Chip (PSoC) [16]. A PSoC integrated circuit is composed of a core, configurable analog and digital blocks and programmable routing and interconnect. Flexible mixed-signal arrays enable signals to be routed to and from I/O pins. This architecture allows designers to create customized peripheral configurations to match the requirements of each individual application, making PSoC substantially different from other microcontroller designs [15].

Reconfigurable hardware platforms are often used as tools for acquisition and processing of data from scientific experiments. Some general (basic) information about the concept and classification of such platforms can be found in [76]. Two specific examples are briefly described below.

The platform described in [12] was designed to acquire and process (in real-time) data from nuclear spectrometry experiments. It uses Xilinx FPGA chips and Texas Instruments Digital Signal Processors. Among the tasks handled by DSP processors is real-time parametrization of the hardware processing algorithms stored in the FPGA. The design also allows the available hardware to be used as a general-purpose acquisition and processing platform.

Another example of a reconfigurable hardware platform for scientific experiments aimed at cellular architectures is called CONFETTI and introduced in [50]. This is a modular, hierarchical platform composed of a number of simple FPGA-based computing units called ECells. ECells can communicate with each other at speeds up to 500 Mbits/s and can be configured independely from many different sources (local FLASH memory, Ethernet, Wi-Fi, etc.) Owing to its modular construction, the user has the ability to replace any of ECell unit with another, compatible unit. Communication between the ECells is also widely configurable.

Another interesting class of devices is represented by Sun SPOTs—Small Programmable Object Technology, developed by Sun Microsystems (now part of Oracle) [69]. SPOTs are designed as an experimental platform for prototyping applications which might be strongly integrated with the environment. Each device comes with a general-purpose sensor board which can—at the developer's discretion—be swapped for a different hardware module such as a flash card reader, a more robust analog input/output extension, or an FPGA board. Another adaptation option is to use the built-in management feature which allows monitoring of working applications, changing device properties and even installing new software remotely.

Address trace analysis is one of the available techniques used to evaluate cache and memory efficiency of computer systems. Address traces are streams of addresses generated during the execution of programs. They can be aggregated using various methods. An interesting way to collect traces—named ATUM—was proposed by Agarwai et al. [2]. Their concept is to modify the microcode of each processor instruction that requires access to memory. In this way the address referenced by that instruction is also stored in a special protected place in memory. This method of collecting address traces can be used in any microprocessor which admits microcode modifications and can be treated as an instrumentation of hardware through firmware modifications (microcode can be treated as processor firmware).

As presented in Table 2 many existing solutions are—in one way or another—related to adaptation loops. While some of them are clearly more mature than others, there is no single technique which tackles all the issues connected with implementing adaptation loops in both the software and the hardware domain. Another significant conclusion is that the hardware domain lacks the requisite Analyze and Plan components. Only a handful of projects address SOA systems, whether directly or indirectly. In order to remedy this issue we have decided to introduce the concept of an Adaptive SOA Solution Stack (AS3) Pattern. It can be applied to heterogeneous environments comprised of both Real-World and Virtual Services in order to enforce the adaptation loop in a seamless and standardized way across all layers of the system which is the subject of adaptation.

5 Adaptive SOA Solution Stack

One of the main models of SOA application development and deployment is the SOA Solution Stack (S3) proposed by IBM [3]. Its core concept is depicted in Fig. 8. The S3 model provides a detailed architectural definition of SOA split into nine layers. Each layer comes with its own logical aspects (including all the architectural building blocks, design decisions, options, key performance indicators, etc.) and physical aspects (which cover the applicability of each logical aspect in reference to specific technologies and products). The S3 model is based on two assumptions:

- The existence of a set of service requirements (functional and nonfunctional) which collectively establish the SOA objective;

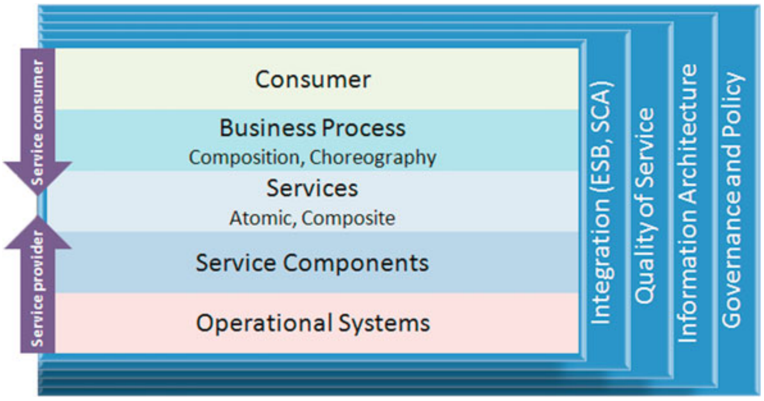


Fig. 8 SOA Solution Stack [3]

- The notion that specific service requirements can be fulfilled by a single layer or some combination of layers. Each layer can satisfy service requirements by way of a layer-specific mechanism.

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 broad (non-technical) description of each S3 layer (except the Consumer layer), is provided in the following paragraphs.

- **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 accessibility of computational or communication resources.
- **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;
  – 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 provides 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 during execution. This behaviour illustrates the role of the Operational Systems layer and facilitates multilayer 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 requestors 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 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 support from 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 2–4. 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 suited for adaptability mechanisms.
- **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, which is needed to correctly interpret actual business information.
- **Governance and Policy**—This layer covers all aspects of managing the business operations' lifecycle. It includes all policies, from manual governance to autonomous policy enforcement. It also 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-specific 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.

The decomposition of SOA Systems proposed by S3 can be used for more precise partitioning of the Adaptive Systems Space introduced in Sect. 2. The "where" axis (showing where adaptation mechanisms can be located) may now be split into sections referring to the S3 layers, as presented in Fig. 9. The Operational Systems

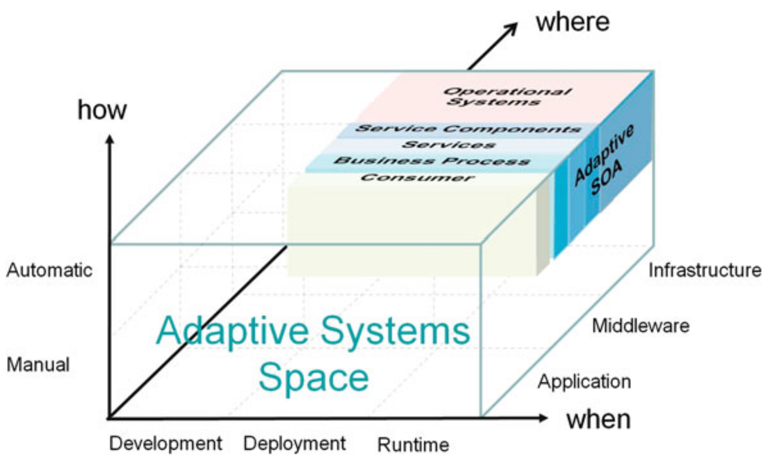


Fig. 9 Mapping of S3 onto the Adaptive Systems Space

layer is assigned to the Infrastructure layer while Service Components, Services, and Business Processes are aggregated by the Middleware Layer. Since the Consumer Layer often contains application-specific mechanisms, it is mapped to the Application Layer. Vertical S3 layers crosscut the entire "where" axis, making them a perfect place for deployment of mechanisms required by the adaptation loop. This is consistent with the fact that the vertical layers (QoS, Information Architecture, Governance and Policy) are directly related to non-functional parameters involved in the adaptation process. The presented mapping clearly highlights the structure of Adaptive SOA in the context of S3.

The S3 Model which coincides with the Adaptive SOA Space is named the Adaptive SOA Solution Stack (AS3) [85]. It could be constructed via uniform introduction of adaptability aspects to each layer of the S3 Model, yielding a multilayer adaptive system which takes advantage of modern software and hardware technologies and offers full control over QoS and QoE parameters. The concept of AS3 has two important constituents:

- AS3 Element Pattern—an architectural pattern used for modelling adaptability aspects in each S3 layer. It contains several components used in the adaptation process.
- AS3 Process—an abstract process defining the transformation of the non-adaptive layer of the S3 stack into its adaptive equivalent.

Each constituent will be described in more detail in the following sections.

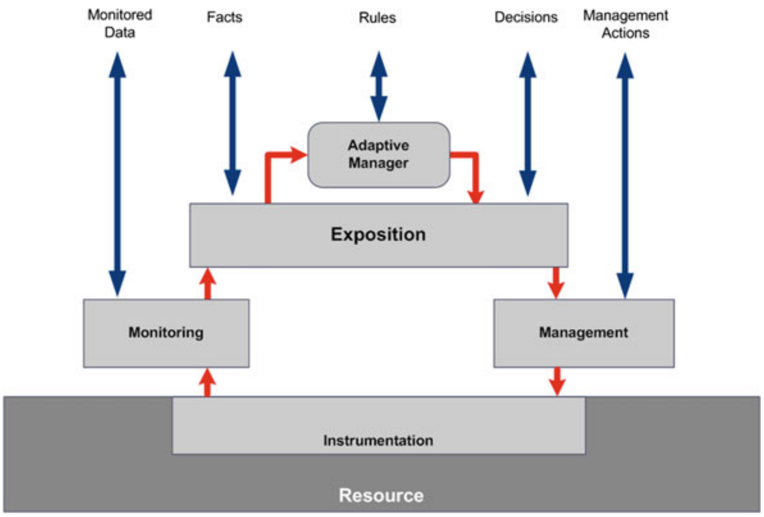


Fig. 10 AS3 Element Pattern

5.1 The AS3 Pattern

In general, the AS3 Pattern depicted in Fig. 10 follows the concept of the MAPE-K control loop and refines it in the context of adaptability-related S3 layers. Throughout the remainder of this chapter we will refer to the AS3 Element Pattern simply as the AS3 Pattern, while the S3 layer to which adaptability is added in accordance with the AS3 Pattern will be referred to as the AS3 Layer. The elements of this pattern could be described as follows.

The Resource is an abstract entity (S3 Layer—e.g. Integration, Service Components or Services). It is transformed into the Managed Resource through instrumentation with sensors and effectors. Sensors expose the state and configuration of the Resource and enable monitoring of its activity. Effectors provide mechanisms for changing Resource parameters or configuration according to actions enforced by the Management Component. Data gathered by sensors is passed to the Monitoring Component which is responsible for calculating selected metrics and processing events. The aggregated data—the output of the Monitoring Component—is forwarded to the Exposition Component.

The Exposition Component cooperates with the Adaptive Manager which is used to select control actions. It transforms monitored data into the format used by the given Manager instance. The Exposition Component therefore acts as a harmonization layer. It is also possible for the Exposition Component to expose some facts to other AS3 Elements and receive high-level decisions which should be enforced in the control loop. The Adaptive Manager Component is used to enact the adaptation loop. Actions selected by the Adaptive Manager are converted by the Exposition Component to a format acceptable by the Management Component. The responsibility of the Management Component is to enforce management actions using effectors of the Instrumentation Component.

As the AS3 Pattern follows the MAPE-K concepts, its elements can be classified in this context. The Resource referred to by the AS3 Pattern (which, following Instrumentation, is transformed into a Managed Resource) represents the same abstraction as the MAPE-K Managed Element. Sensors and effectors perform similar roles in both approaches. The Monitoring Component realizes the Monitor phase while the Management Component handles aspects of the Execution phase of MAPE-K. The Analysis and Planning phases are realized by the Adaptive Manager of the AS3 Pattern. To manage both phases the Adaptive Manager uses a declarative strategy named the Adaptation Strategy. It is assumed that the Adaptation Strategy can be represented with the use of rules which currently do not have any semantic context for realization of the Knowledge concept of MAPE-K. One component of the AS3 Pattern which does not have a direct MAPE-K counterpart is Exposition. It acts as a data harmonization layer and enables facts and high-level decisions to be obtained and exposed by AS3 Patterns located in different S3 Layers.

The full potential of the AS3 Stack is manifested in the cooperation abilities of different AS3 Layers. For instance, it is possible to monitor the whole system by collecting information from Monitoring Components present in each AS3 Layer. Such reasoning may also refer to other types of AS3 components. Thus, the following aspects are inherent in a complete AS3 stack: observability (Monitoring component), manageability (Management component) and policy (Adaptive Manager component) [85]. A key challenge related to leveraging the AS3 Stack is to propose a means of introducing adaptability to layers of the SOA system in accordance with the AS3 Pattern, as well as managing adaptability in an effective way.

The AS3 Pattern can be uniformly applied to systems consisting of both Real-World and Virtual Services. Figure 11 depicts the concept map which reflects the application of the AS3 Pattern in such heterogeneous systems. Blue elements have already been introduced in Sect. 3—they are related to SOA systems. The remaining (green) elements are directly related to the concept of AS3. The AS3 Pattern draws upon Adaptive SOA as one of the possible approaches to modelling adaptation in SOA systems. As presented in the concept map, the AS3 Pattern models enrichment of Middleware and Infrastructure, placed on the "where" axis in Fig. 9, which presents the Adaptive Systems Space. The purpose of enrichment is to enable adaptation of Composite Services and reconfiguration of Physical Resources, with particular focus on Real-World Devices. Enrichment of Middleware concerns all components of the AS3 Pattern, while enrichment of Infrastructure involves only a restricted subset of AS3 Pattern components, i.e. Instrumentation (sensors and effectors), Monitoring and Management. Since hardware is less flexible than software, full implementation of the adaptation loop is rather difficult, as highlighted in Sect. 4. Infrastructure sensors and effectors are enabled by reconfiguration of Physical Resources which could be either Computers or Real-World Devices. Middleware is enriched with all AS3 Pattern and supports the complete adaptation loop. An important point is that instrumented infrastructure enables multi-level adaptation of both Real-World and Virtual Services, performed on the Middleware level.

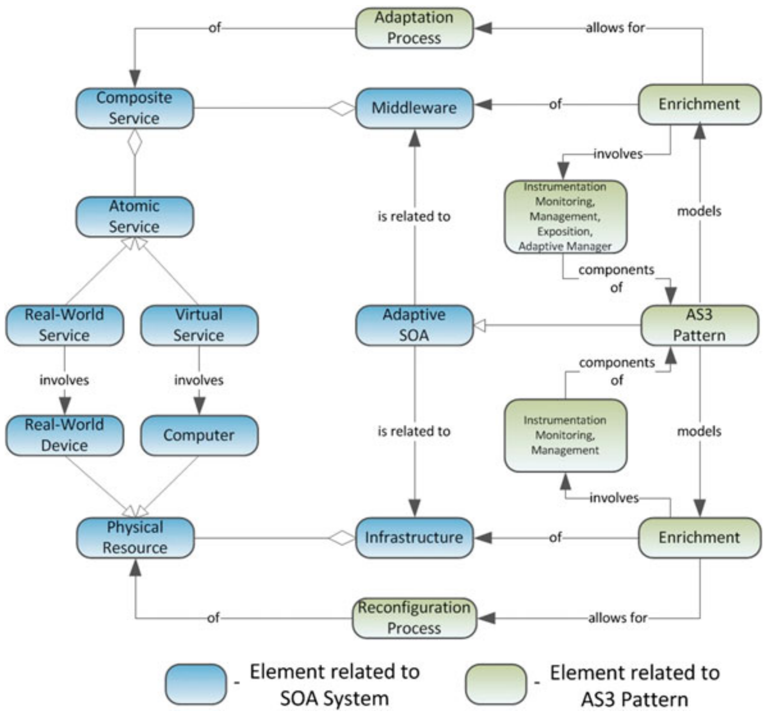


Fig. 11 Application of the AS3 Pattern in heterogeneous systems

The following parts of this section explain some of the concepts introduced in Fig. 11. The first part focuses on the Middleware and explains the enrichment and adaptation enabled by the AS3 Pattern, while the second part is devoted to Infrastructure and its enrichment, highlighting the reconfiguration of Real-World Devices.

Enrichment Supporting the Adaptation Process in the Middleware Layer

Figure 12 presents Middleware enrichment with the use of the AS3 Pattern. First of all, it is assumed that Middleware has a layered structure and that each layer can be divided into a part which provides the Runtime Environment and a part containing the layer's Logic. Logic is delivered by some Artifacts which are specific to the given layer. The Runtime Environment for Artifacts is assumed to be provided in the form of a Container. Layer-specific Artifacts are simply deployed to the Container, which provides them with the Communication feature. As mentioned before, the Middleware enrichment involves all components introduced by the AS3 Pattern. Realization of such enrichment assumes that Monitoring, Management, Exposition and Adaptive Manager components are simply deployed to the Layer's Container in the form of Layer-specific Artifacts. The Instrumentation component is handled

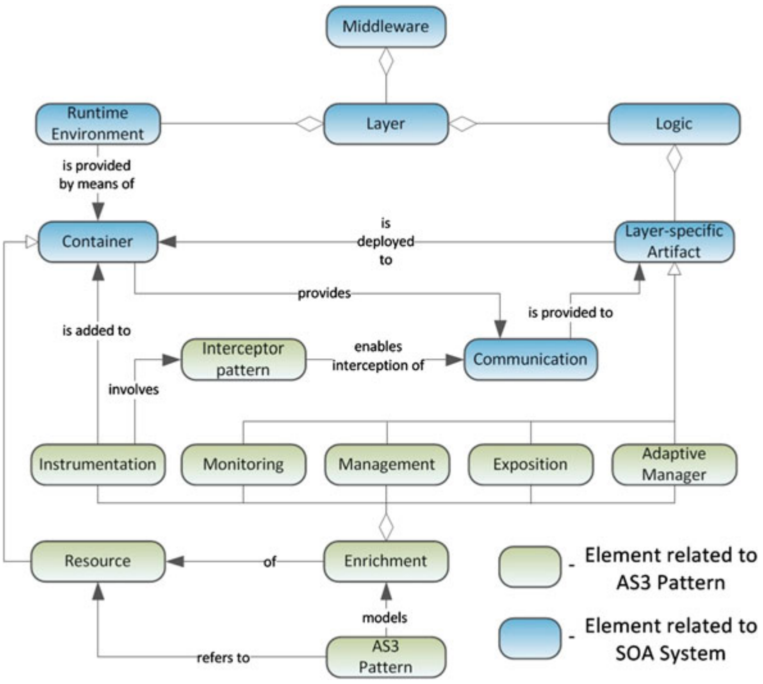


Fig. 12 Enrichment of Middleware using the AS3 Pattern

differently. Since the AS3 Pattern models the enrichment of some Abstract Resource, Middleware Layer Containers are treated as Resources to which Instrumentation is added. It is therefore assumed that Instrumentation is introduced in the Container (by whatever means) and that it applies the interceptor design pattern. The use of this pattern allows for interception of communication performed by Artifacts and leverages this for implementation of Sensors and Effectors which provide monitoring and management features for respective components of the AS3 Pattern. The presented design of Middleware enrichment carries several important advantages:

- Communication between Layer-specific Artifacts can be monitored and managed in an non-intrusive way, which is transparent for applications.
- Monitoring, Management, Exposition and Adaptive Manager components can be easily deployed and managed owing to management features provided by the Runtime Environment of a given Layer.
- Communication between AS3 Pattern components can be easily performed with the use of the Communication feature provided by the Container.

The final purpose of Middleware enrichment is providing mechanisms required by adaptation of Composite Services and used to implement the application's logic. In order to realize this goal, AS3 assumes a certain structure of Composite Services. Specifically, it is assumed, that on a higher level of abstraction each Composite

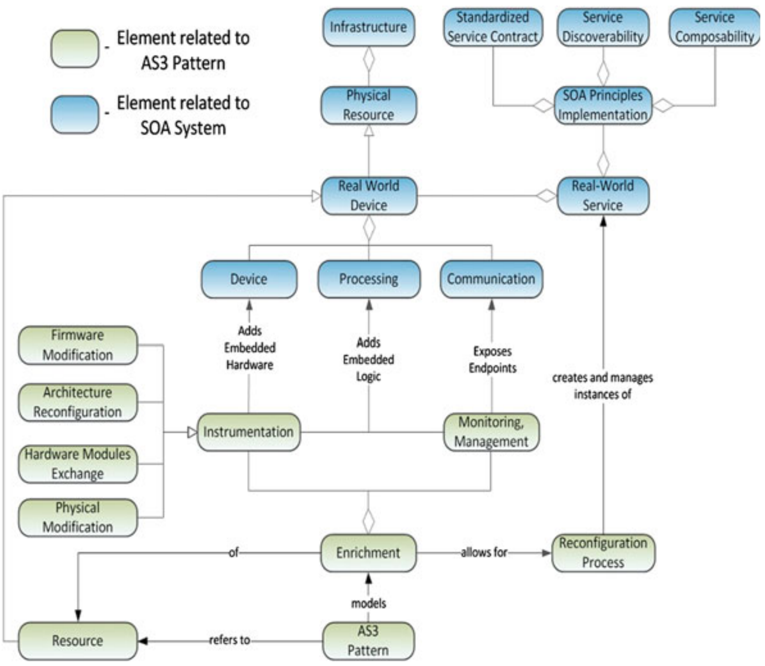


Fig. 13 Enrichment and Reconfiguration of the Infrastructure using the AS3 Pattern

Service can be described as an abstract composition in which abstract services are, in turn, described by their features without referring to a particular instance. For each abstract service several different instances can be deployed and used during execution of an application. Middleware enrichment allows the decision subsystem to dynamically recompose the application by selecting a set of service instances for each service that belongs to the Composite Service. When decisions are made locally, i.e. without regard to how a particular service instance may influence the overall application, the results often lead to unsatisfactory solutions. To improve the outcome of adaptation, approaches such as usage of stochastic models may be used to estimate global system behaviour.

Enrichment Supporting the Reconfiguration Process of the Infrastructure Layer

Figure 13 presents the concept of the Infrastructure enrichment using the AS3 Pattern. The figure focuses on the aspect of Real-World Services and Real-World Devices. Mirroring the relations depicted in Fig. 11, the Infrastructure aggregates Physical Resources, some of which may be Real World Devices. Figure 13 shows that the Real-World Service always involves some Real World Device. Figure 13 makes this involvement more specific and shows that the Real-World Service is an aggregation of a Real World Device and an implementation of SOA Principles. The following SOA Principles are singled out as especially important: Standardized Service Contract, Service Discoverability and Service Composability. Each Real-World Service needs to publish its Contract to some entity in order to be discoverable by other services, and as a result, allow its features to be integrated in Composite Services. Furthermore, it is assumed that each Real-World Device comprises three main elements: a Device (which implements some feature in the real world), Processing Logic (which deals with controlling the Device) and some Communication Mechanisms (which enable the Device to be controlled remotely). The aforementioned implementation of SOA Principles is concerned mostly with Communication mechanisms and uses them to expose Processing in a service-oriented way.

As stated before, Infrastructure enrichment involves three components of the AS3 Pattern: Instrumentation, Monitoring and Management. The resources to which the AS3 Pattern refers are Real-World Devices. Instrumentation of Real-World Devices can be performed with the use of the following approaches: Firmware Modification, Architecture Reconfiguration, Hardware Module Exchange, Physical Modification (all described in detail in Sect. 4.1). As presented in Fig. 13, each of these approaches is a combination of the following actions: adding Embedded Hardware to the Device and/or adding Embedded Logic to Processing. Regardless of the approach used, the result of such actions is always the same: the Real-World Device is equipped with Sensors and Effectors. Monitoring and Management components are handled in a somewhat different manner than Instrumentation. While they can also involve some additional Embedded Logic, they mostly focus on exposure of endpoints by means of Communication mechanisms. The presented realization of Infrastructure enrichment allows for Reconfiguration, which can be used to spawn new Real-World Services and manage them in the context of a specific Real-World Device.

5.2 The AS3 Process

The purpose of the AS3 Process is transforming systems built in accordance with the S3 Model into adaptive environments and managing the adaptation process across different S3 Layers. The assumption of the AS3 Process is that the AS3 Pattern may be applied only to selected layers. Leveraging the potential of specific layers in the context of the adaptation loop may enhance the SOA environment with (among others) the following features:

- Dynamic service sizing [64]: scaling the service to adapt to changing load conditions, either by (i) scaling up (i.e. resizing a running service component, for example by increasing or decreasing its memory allocation), or (ii) scaling out (adding or removing instances of service components).

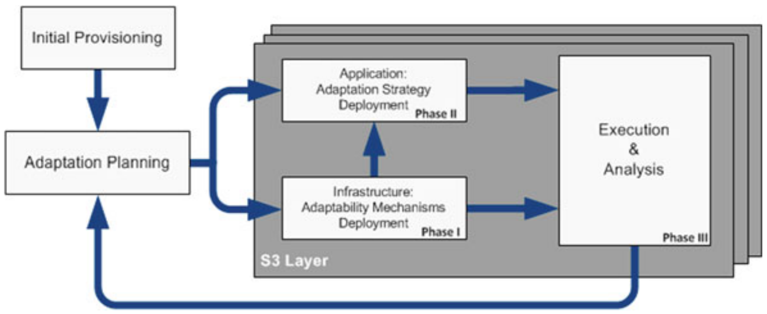

Initial Provisioning
Adaptation Planning
Application: Adaptation Strategy Deployment
Phase II
Infrastructure: Adaptability Mechanisms Deployment
Phase I
Execution & Analysis
Phase III
S3 Layer

Fig. 14 Abstract view of the AS3 Process

- Policy-driven operation optimization [85]—flexibility can be controlled by rules, following an "event-condition-action" approach in which certain conditions trigger automatic actions to alter the service's capacity.
- Cross-business process monitoring and management [61] e.g. to enforce a close feedback loop control paradigm.

The AS3 Process involves the selection of layers that need to be enhanced with adaptability. Such an approach alleviates the overhead incurred by any adaptation-related activities that do not contribute to improving the performance of the application. An abstract view of the AS3 Process in presented in Fig. 14. The Process relies on the following two concepts related to adaptability deployment supported by the AS3 Pattern:

- Adaptability Mechanisms (AM)—these affect all components of the AS3 Pattern, i.e. Instrumentation, Monitoring, Management, Exposition and Adaptive Manager. Deployment of those components into the S3 Layer transforms it into the AS3 Layer, capable of enforcing the Adaptation Strategy.
- Adaptation Strategy (AS)—configuration of Adaptability Mechanisms which drives the adaptation loop. The strategy always refers to sensors and effectors needed to monitor and manage a fragment of the application whose adaptation is defined in the strategy. This strategy is enforced by the Adaptive Manager.

The central activity of the AS3 Process is Adaptation Planning which selects S3 Layers in which adaptation has to be introduced and prescribes an Adaptation Strategy for each layer. A prerequisite for starting the AS3 Process is Initial Provisioning. The purpose of Initial Provisioning is installation of agnostic monitoring and management mechanisms which cut across all S3 Layers. These mechanisms enable discovery of resources belonging to S3 Layers and monitoring of selected QoS parameters. With their help it becomes possible to identify S3 Layers which require Adaptability Mechanisms and design an initial Adaptation Strategy for selected application fragments.

Parts of the AS3 Process related to selected S3 Layers can be divided into three phases. In Phase I, Adaptability Mechanisms are deployed to the infrastructure in accordance with the AS3 Pattern. In Phase II the Adaptation Strategy related to a given application fragment is deployed to the AS3 Layer. In Phase III the adaptation loop of the AS3 Pattern is started and execution of the application fragment is monitored and analyzed. Phase III leverages the AS3 observability aspects by gathering data from the Monitoring Component, as well as its manageability aspect by introducing minor corrections through the Management Component. The output of analysis performed in Phase III is passed to Adaptation Planning, completing the AS3 Process loop.

During subsequent executions of Adaptation Planning, monitoring data obtained from agnostic Initial Provisioning mechanisms is combined with data provided by AS3 Layers. This results in a comprehensive view of the system state and shows how the adaptation process influences the fulfillment of consumer requirements. As a result of Adaptation Planning, some (or all) of the following actions may be executed:

- deploying Adaptability Mechanisms to an S3 Layer which had not been instrumented before;
- modifying the previously-deployed Adaptation Strategy or Adaptability Mechanisms (for instance by adding more mechanisms);
- removing the previously-deployed Adaptation Strategy or Adaptability Mechanisms which are no longer needed.

In a given S3 Layer, execution of actions enforced by the Adaptation Planning may involve Phase I, Phase II or both phases in such a way that Phase I occurs before Phase II. If Adaptation Planning does not enforce any changes in the infrastructure and on the application level then the Process progresses directly to Phase III where the execution of the adaptation loop is monitored. Phases of the AS3 Process can be performed at different stages of the system's lifecycle. e.g. during provisioning and execution. This distinction is important as some extensions can be introduced either during deployment or at runtime. The AS3 process reflects this fact by introducing three different models (Static, Hybrid and Dynamic) for each phase of the AS3 Process.

The Static Model assumes that a given phase cannot be performed at runtime. In Phase I this means that the infrastructure (or part thereof) has to be shut down and then, once appropriate changes are applied, the infrastructure is again provisioned and returns to the execution phase. In Phase II the same applies to the application. The application has to be stopped and redeployed to support adaptability required by the Adaptation Strategy. The static model can be imposed e.g. by execution containers and applications which do not support runtime modifications. The Dynamic Model assumes that a given phase of the AS3 Process can be performed at runtime. In Phase I the Adaptability Mechanisms are deployed to the infrastructure without the need for a restart. In Phase II deployment of the Adaptation Stategy does not involve halting the application. The Hybrid Model assumes that some modifications of the adaptation process might be performed using the Dynamic Model while others may need to follow the Static procedure. Both Phases (I and II) can be handled in Static, Dynamic or Hybrid Models. Phase III is performed exclusively in the Dynamic Model since it is closely related to execution of the system and oversight of the adaptation process.

Additionally, Phases II and III are executed depending on the implementation of a given S3 Layer as well as the capabilities of Adaptability Mechanisms designed for that layer.

Having presented an abstract view of the AS3 Process and discussed its related aspects, we can summarize the exact steps of the Process in the following list:

1. Performing Initial Provisioning of the whole system;
2. Discovering resources present in all layers;
3. Performing adaption planning which influences steps 5 and 6;
4. Deploying/modifying/undeploying Adaptability Mechanisms in selected S3 Layers according to a suitable execution model;
5. Deploying/modifying/undeploying Adaptation Strategy Agents in selected S3 Layers according to a suitable execution model;
6. Deploying/modifying/undeploying Adaptation Strategy in selected AS3 Layers according to a suitable execution model;
7. Executing the adaptation process and analysing monitoring data provided by AS3 Layers and Initial Provisioning.

The process then continues by jumping to step 2.

The complexity of real-life SOA systems calls for tools which support effective realization of the AS3 Process. The most important core features required from such tools are as follows:

- Selective non-intrusive monitoring installable on demand across different layers of the S3 Model;
- Discovery of services and their interconnection topology during system operation;
- Flexible mechanisms for presenting and managing monitoring data in order to support system response evaluation and adaptation strategy planning;
- Dynamically defined and pluggable adaptability policies for execution of operations;
- On-demand installation of effectors to enforce adaptation decisions.

A toolkit which supports the AS3 Process and meets all the listed requirements—namely, the AS3 Studio—is presented in the next section.

6 AS3 Studio

The AS3 Process is a high-level concept which is platform-independent. However, tools that automate it have to be platform-specific and support a selected set of technologies. Recently, many vendors of SOA-related solutions have begun to focus on supporting dynamic and manageable software environments executed within OSGi [54] containers. More importantly, some of those solutions available as open-source software can be used for implementation of different layers of the S3 Model: examples include Fuse ESB (Services and Integration Layers), Apache Tuscany (Service

Components Layer) and Business Process engines: Apache ODE, JBoss jBPM (Business Process Layer). In light of this, OSGi emerges as the natural choice for the base implementation technology of AS3 Studio.

6.1 OSGi Monitoring and Management Platform

At its core, OSGi [54] is a dynamic component-oriented Java platform for applications developed in accordance with service-oriented design principles [20]. The OSGi framework provides an execution environment for applications, which are called bundles. Bundles expose their features as services according to the "publish, bind, and find" model [30]. Each bundle can be deployed and activated at runtime. Bundles can dynamically select services to be used. Furthermore, the OSGi Framework enforces strict modularization of bundles, which entails that there is no need to shut down the entire JVM when a particular bundle is modified.

The AS3 Studio is a suite of several components deployed over the OSGi Monitoring and Management (OSGiMM) platform:

- AS3 Tools for the Middleware Layer,
- AS3 Tools for Real-World Services,
- AS3 Console.

OSGiMM [62] provides mechanisms for monitoring and managing OSGi containers federated by means of Message Oriented Middleware consisting of a network of message brokers. It enables cooperation of services deployed in OSGi containers distributed across a Federated OSGi system.

OSGiMM consists of core instrumentation and a set of bundles. The core instrumentation is required for dynamic management and monitoring. During installation, the instrumentation has to be added to each OSGi container with the use of provided scripts. Accordingly, OSGiMM bundles have to be deployed to each container of federation. The fundamental feature of OSGiMM is discovering information about all services, bundles and containers of the Federated OSGi as well as their structural relations.[1] Such information is later referred to as a topology. The implementation also provides efficient invocations of service groups, which are used as a foundation for typical management and monitoring patterns [86] in the OSGiMM.

In summary, OSGiMM provides generic features which affect the implementation of the following adaptability aspects:

- Declarativity—the user specifies a monitoring scenario, indicating which parameters need to be monitored and which topology elements are provided. A scenario is specified declaratively and can be exported to a distributed repository for future use.

[1] For example, a container may comprise bundles while a bundle may consist of specific services.

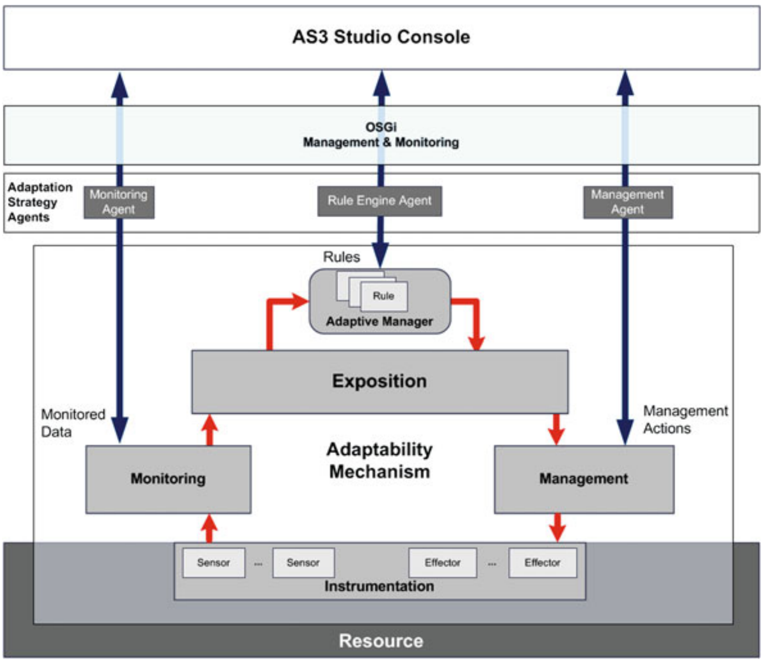


Fig. 15 AS3 toolkit architecture

- Dynamism—monitoring scenarios can be activated at any time and activation does not involve halting the application that is to be monitored. Activation triggers realization of on-demand instrumentation.
- Selectivity—instrumentation is only triggered in locations which are important for a given monitoring scenario; therefore the overhead incurred by monitoring is restricted to a minimum.
- Self-configuration—the federation may change, e.g. when a new container is added or when some services are undeployed. Regardless, the realization of the monitoring scenario is ensured.
- Flow aggregation—when there are multiple users who wish to perform the same monitoring or management activities, the data flow related to the task is aggregated in order to reduce the cost of transmission.

All those features make the OSGiMM a good solution for managing and monitoring OSGi services in a unified manner, which is important in the context of adaptability aspects.

Each of the AS3 tools for the Middleware Layer is a set of bundles that implement Adaptability Mechanisms for a particular S3 Layer, and a set of agents for controlling these mechanisms. The specified architecture of an AS3 Tool is depicted in Fig. 15. There are three different agents: Monitoring, Rule Engine, and Management, all referred to as Adaptation Strategy Agents (ASA). Currently, three AS3 Tools are available. Adaptive SCA (Service Components S3 Layer) is a tool which can be used for building and managing adaptive services compliant with the component-based SCA technology. Adaptive VESB (Services and Integration S3 Layers) can be used to ensure adaptation of services deployed in the Enterprise Service Bus. BPEL Monitoring (Business Process Layer) provides mechanisms for selective BPEL process monitoring and management.

AS3 tools for Real-World Services include three utilities for use with Real-World Services (RWS) in the AS3 Process. The first tool, RWS Builder, handles initialization within the infrastructure layer by creating Real-World Services on top of Real-World Devices. Discovering the resources present in the infrastructure layer is possible thanks to the RWS Discover tool. Performing adaption planning in the context of the infrastructure layer means identifying places where reconfiguration mechanisms should be added to allow creation of many Real-World Services. The final tool, RWS Reconfiguration, can be used to manage the configuration of the Real-World Device and therefore spawn new Real-World Services. All other steps of the AS3 Process are covered by the AS3 tools for the Middleware Layer.

The AS3 Console is based on the Eclipse RCP technology [45] and, as such, enables implementation of Console extensions via plugins. Each of the AS3 tools contributes a different set of plugins to the Console. These plugins provide GUI components which support layer-specific features, e.g. definition of SCA or ESB adaptation rules, creation of facts, adaptive component implementation, complex service modelling, BPEL engine instrumentation and discovery, etc. The AS3 Console is also extended with OSGiMM plugins, which allow it to discover, monitor and manage whole Federated OSGi. In order to connect to the Federated OSGi it is necessary to provide the address of the federation container which will function as the entry point for the console. Many AS3 Consoles can be connected to federation containers simultaneously and it is also possible to connect more than one Console to a single container. In this way it is possible to use the AS3 Studio from many points of the federation at the same time.

As listed in the previous section, the AS3 Process involves execution of several steps. These steps are supported by the AS3 Studio according to the workflow presented in the previous section. Step 1 (Initial Provisioning) has to be performed manually. During this step OSGiMM bundles, along with their core instrumentation, are installed in each of the OSGi containers. Subsequently the configuration of message brokers has to be provided. During this process a logical topology of connections between federation nodes is created which combines the containers into a Federated OSGi. Steps 2 (Discovery) and 3 (Adaptation Planning) are described later on in the section, as they involve the continuous adaptation process. The AS3 Studio automates steps 4 and 5 (Deploying/modifying/undeploying Adaptability Mechanisms and Adaptation Strategy Agents). The user may choose (using the AS3 Console) which mechanisms or agents should be installed in each container. Since OSGiMM discovers the topology of the federation it is possible to transparently transfer specially prepared bundles to remote containers and install them automatically. Once this is done, the user may check whether all operations finished successfully and whether AM/ASA are present in the discovered federation topology. Furthermore, AS3 Studio automates Step 6 (Deploying/modifying/undeploying Adaptation Strategy) by employing the preinstalled Adaptation Strategy Agents. The Adaptation Strategy is specific to a particular layer of the AS3 Model and will therefore be presented along with overall description of the tools.

Steps 2 (Discovery) and 3 (Adaptation Planning) are both performed in a continuous manner during runtime, with full support of the AS3 Studio. This support exploits the concept of the Dynamic Monitoring Framework [86] proposed in our earlier work. The Monitoring Agent in each AS3 Layer, as well as OSGiMM itself, implement an interface which provides topology discovery of resources available in this layer. Additionally it is possible to declaratively specify a Monitoring Scenario which can contain two types of Monitoring Subscriptions:

- Topology Subscription—related to monitoring changes in a topology fragment specified in the subscription,
- Metric Subscription—related to monitoring of topology element metrics (performance, availability, reliability).

The Monitoring Scenario is created with the use of the AS3 Console. When a Scenario is activated, OSGiMM sends subscriptions to appropriate containers where they are passed to the Monitoring Agents. Each agents starts a monitoring process on behalf of a given subscription. Results are communicated to the AS3 Console. The Console provides configurable monitoring panels which can be tailored to a given Scenario, enabling visualization of monitoring data collected from different parts of the federation. The described mechanisms ensure continuous discovery and monitoring, thus allowing the operator to identify elements of the system where adaptation should be applied.[2] When the system changes, the monitoring processes adapt automatically and monitoring panels continue to relay information relevant to Adaptation Planning.

Steps 6 and 7 of the AS3 Process are specific to each AS3 tool and will be described separately for each of the AS3 tools in the following paragraphs. .

6.2 Adaptive VESB

Adaptive VESB is the AS3 tool which introduces adaptability mechanisms within the Integration Layer of the S3 Model. The tool exploits model-driven adaptation [72] 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 selected 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 behaviour is represented by the service execution model. The composite service execution model

[2] Further details can be found on the AS3 Studio website and in our previous paper [86].

is an abstraction of the execution environment. It covers low-level events and relations between services, exposing them as facts in the model domain. Decisions taken in the model domain are translated to the execution environment domain and then executed. An adaptation strategy is specified, according to a user-provided service execution model, taking into account the quality of execution metrics. It relies on configuring ESB [73] elements that are responsible for dynamic selection of service instances for particular use cases in order to provide the desired quality.

Adaptability Mechanisms for ESB are constructed around the AS3 Pattern and consist of the following elements: instrumentation component, monitoring component, management component and adaptive manager. The instrumentation layer enriches ESB with additional elements providing the adaptability transformations necessary to achieve adaptive ESB. These elements are responsible for managing sensors and effectors installed in ESB. Sensors gather information about running applications by intercepting messages, while effectors are used to influence message flow in ESB by modifying the routing table of NMR (which is a core element of ESB involved in message processing). The monitoring layer supplies notifications of events occurring in the execution environment. As the volume of monitoring information gathered from ESB might overwhelm the Exposition layer, events are correlated with one another using Complex Event Processing and notifications pertain only to such complex events. The Exposition Layer is responsible for maintaining and updating the composite service execution model. Facts representing the state of the system or events occurring within are supplied to the Adaptive Manager which analyses them and infers decisions to be implemented in the execution environment by the Management Layer.

Adaptation Strategy Agents for ESB provide interfaces for managing adaptability mechanisms. The Monitoring Agent provides high-level features for management and configuration of sensors deployed in ESB. The Management Agent can be used to influence message routing in ESB, while the Rule Engine Agent manages the composite service execution model and deploys adaptation strategies. The Adaptation Strategy for Adaptive ESB consists of the following elements:

- Definition of a composite service execution model,
- Definition of new facts that represents events occurring in the integration layer,
- Definition of an adaptation policy that uses the previously defined and deployed facts.

All these elements can be configured using the ASA provided by VESB tools. To simplify strategy definition, VESB Tools also provide a GUI for the AS3 Studio Console.

6.3 Adaptive SCA

Adaptive SCA is an AS3 tool designed to enhance the Service Components Layer of the S3 Model with adaptability features. It enables service designers to assemble services from components according to the SCA specification. SCA is a technology-independent solution and therefore supports components created using many different technologies as well as various communication protocols. A set of components connected with one another within a service is also called a composition.

SCA uses the Dependency Injection Pattern [58] to model a composition: adaptation mechanisms need to be introduced on the level of references between components. Adaptive SCA instruments these references and uses OSGiMM to monitor communication between components, providing suitable mechanisms to choose which component should handle a particular reference.

The Adaptation Strategy for Adaptive SCA defines different sets of components which are to be used in case of specific situations discovered by the monitoring system. These sets are also referred to as composition instances. For instance, if a service composed of particular components becomes too slow, it may temporarily switch over to other components which provide better QoS (however at a higher cost).

The Adaptation Strategy definition for Adaptive SCA consists of the following elements:

- Service model definition, i.e. a composition and a set of its instances,
- Definition of adaptation policies that use predefined policy templates and metrics gathered by OSGiMM.

If required, it is possible to create policy templates tailored to a particular service. All these features are supported by the ASA Adaptive SCA tool. To simplify their management, Adaptive SCA also provides a GUI for the AS3 Studio Console.

Deployment of a service with adaptability mechanisms requires additional actions performed automatically by the AS3 Studio. Upon defining the composition and its instances, the Adaptive SCA Tool modifies the SCA deployment descriptor by injecting monitoring and management agents into components' references. Afterwards, service provisioning can be performed and the service executed in accordance with the AS3 Process.

6.4 BPEL Monitoring

The Business Process Layer of the S3 Model supports composition of business processes from the available services describing steps that need to be executed to complete a given process. This allows non-technical users to declaratively describe business process flows either in terms of a document in a dedicated language (such as BPEL) or by using graphical tools. Formally defined business processes can be executed by business process execution engines which interact with services according to the specified flow. Modern business processes are often highly dynamic, which means that proper aggregation and analysis of performance indicators representing their execution is important from the managerial viewpoint.

The AS3 Studio offers a highly configurable monitoring system called the Business Process Monitoring Platform, which support efficient capture, propagation and visualization of the data needed for the business process execution analysis. This tool is implemented within an OSGiMM container and utilizes important system services in order to satisfy the following functional requirements:

- discovery and presentation of the existing business process engines, deployed processes and running process instances,
- on-demand monitoring of the selected discovered elements (mainly business processes),
- presentation and up-to-date view of business process definitions and the execution state of running process instances.

The Business Process Monitoring Platform was developed with the following nonfunctional requirements in mind:

- only the necessary data is transmitted between system elements,
- the monitored business processes are not affected by the monitoring process,
- the architecture remains scalable and extensible for multi-vendor systems,
- a standard data model is in place for all the monitored components.

The system relies heavily on mechanisms provided by the OSGi standard which naturally create a SOA environment in a single Java Virtual Machine. System elements are exposed as OSGi services capable of dynamic discovery and runtime reconfiguration. The core layers in the presented architecture are:

- ESB with OSGiMM—a communications backbone that provides seamless integration of system components, their discovery as well as transport of monitoring data.
- BPEL Monitoring Domain—consists mainly of business process engines along with their respective sensors: BPM Engine Monitors, business processes, independent event processing components and infrastructure nodes where such elements are deployed.
- Monitoring Console—a GUI system element that supports configuration of the monitoring system and exposes key system features to the user.

Each monitored business process engine is associated with a dedicated monitor which generates standardized notifications of changes in the executing process.

The communications layer is responsible for propagation of events, as well as filtering them when no subscriber is interested in a particular kind of event. The presented monitoring platform uses OSGiMM and supports model-based declarative definitions of the monitoring process by means of a monitoring scenario. In light of the above, business process monitoring can be treated as an example of a well-defined scenario.

There are two types of loosely coupled clients connected by the OSGiMM backbone: event sources (mainly business process engine-specific monitors—BPM Engine Monitors) and event consumers such as Monitoring Consoles. The architecture also covers event interceptors, e.g. rule-based event processors that are hybrids of these two client types: they intercept the flow of events from other event sources to the Monitoring Console for the purpose of processing.

In the context of the presented work installing a monitoring scenario is equivalent to creating a monitoring subscription for the events whose flows is enabled by the activation of the monitoring scenario. As the main monitoring goals are twofold (monitoring specific topology elements and the entire topology), topology subscription is also supported.

In comparison with the layered architecture of OSGiMM, we can observe relocation of the business process analysis logic. Whereas in standard OSGiMM metric events are processed by building a CEP processor, in the presented platform analytical logic (rule-based event processors) is moved upstream in the layered architecture, becoming an optional and reconfigurable topology element, to enable better management by business users.

6.5 *Infrastructure Enrichment Tools*

In this subsection two software tools for creating and managing Real-World Services are presented. The first tool (RWS Builder) supports creating RWS by adding service logic, transforming a Real-World Device into a Real-World Service. The second tool (RWS Reconfiguration) enriches the infrastructure with a reconfiguration feature which can be used in the service adaptation process or for incarnation of new Real-World Services. Both tools operate in a specific hardware environment containing FPGA (Field Programmable Gate Array) and CPLD (Complex Programmable Logic Devices) chips from Altera Corp., microcontrollers with the ARM-Cortex core from ST-Microelectronics, Tibbo Ethernet and Wi-Fi modules on custom circuit boards.

6.5.1 RWS Builder

The process of creating a Real-World Service on the basis of a Real-World Device consists of several steps described in Chap. 3. One of those steps involves generating the service logic. Here, the programmer's effort can be greatly reduced by using the tool described in the following section.

Real World Service logic is typically implemented as a hardware description of an FPGA chip set up as a software module for a microcontroller-based device. Developing this logic manually to produce a hardware description is a tedious job which requires knowledge of the underlying hardware architecture and proficiency in using hardware description languages (i.e. VHDL, Verilog).

RWS Builder is an example of a software tool for high-level code synthesis. Its functionality is tailored to two specific types of RWS: motion detection and object classifier services. The generated project files depend on the selected functionality and can be filled with service-specific code by the developer. RWS Builder integrates multiple heterogeneous design tools from different vendors into a single applica-

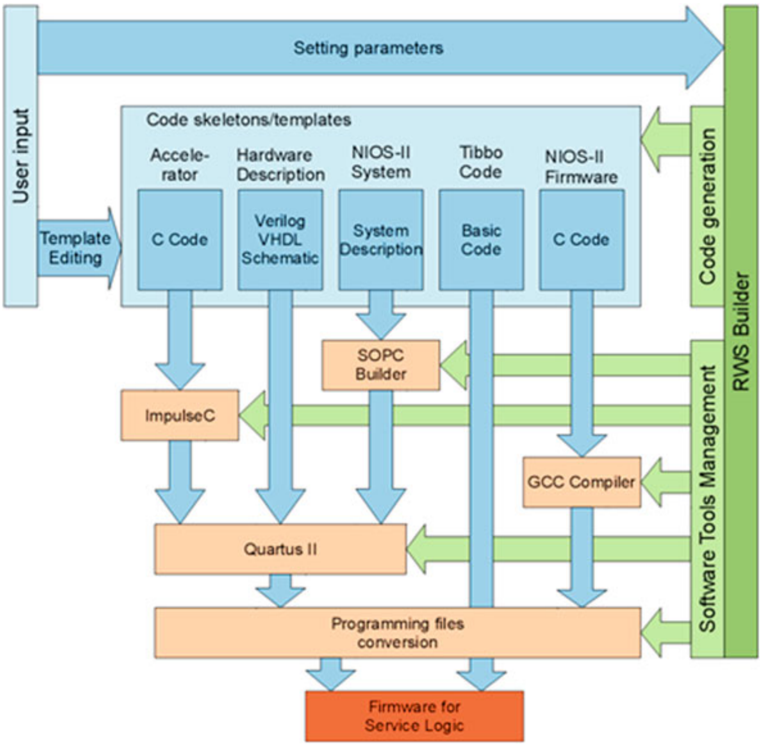


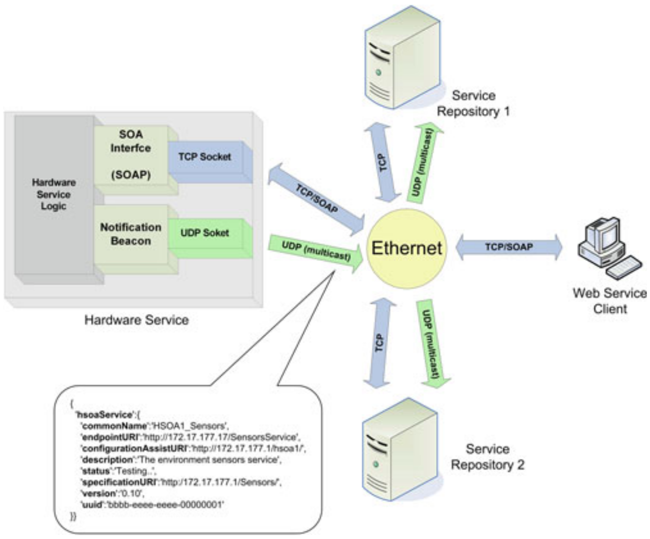
Fig. 16 Service logic generation using RWS Builder

tion that manages all required design, implementation and installation steps. It is responsible for creating an optimized code skeleton to be compiled by the ImpulseC C-to-HDL compiler. It also generates an adequate standard hardware design for the Altera Quartus-II environment and an optimized software template for embedded FPGA microcontrollers and network communication modules. The service logic generation procedure is visually depicted in Fig. 16.

6.5.2 RWS Reconfiguration

An important aspect of providing RWS is introducing solutions which enable discovery of such services along with their reconfiguration. This is the key added value of AS3. Discovery tools are necessary to locate RWS in a network and to recognize their potential capabilities and what kind of logic they can handle.

In order to provide a list of all Real-World Services to users and adaptation tools a live repository has been implemented. The selected approach follows the same paradigm as in Service Location Protocol (SLP)-and Simple Service Discovery Pro-



```
{
 'hsoaService':{
  'commonName':'HSOA1_Sensors',
  'endpointURI':'http://172.17.177.17/SensorsService',
  'configurationAssistURI':'http://172.17.177.1/hsoa1/',
  'description':'The environment sensors service',
  'status':'Testing..',
  'specificationURI':'http:/172.17.177.1/Sensors/',
  'version':'0.10',
  'uuid':'bbbb-eeee-eeee-00000001'
}}
```

Fig. 17 RWS Discovery method

tocol (SSDP)-based discovery schemes. The general architecture of the proposed discovery solution is presented in Fig. 17.

Each of the services available in the network announces its presence using User Datagram Protocol (UDP) advertisements sent out to a well-known multicast address. Such an advertisement, called a notification beacon, carries service description, including metadata (endpoint information, Universal Unique Identifier (UUID), timestamp, lifetime, human-readable description etc.) and a link to a service specification document. Notification beacons are received by dedicated multicast listeners which in turn update the Lightweight Directory Access Protocol (LDAP) service repository. Data stored in different repositories can be synchronized using LDAP servers tools. In this way one repository can collect service announcements from more than one IP network. Service announcements may employ addresses accessible from outside the local network if this type of client is expected. Access to the repository is usually provided using a Transmission Control Protocol (TCP) connection. In the absence of network configuration restrictions computers from all over the Internet can browse the repository and utilize hardware services. From the repository point of view, any external entity which needs to update the repository contents needs to provide security credentials and also pass authorization checks. The repository itself was implemented as an LDAP database, leveraging existing security mechanisms to provide Authentication, Authorization, and Accounting (AAA) services. It can be used by automatic tools as well as end users interested in browsing through the repository contents and able to leverage LDAP query and search mechanisms.

Once the hardware device is detected, it might be necessary for the user to choose and upload firmware and configuration files which provide service-specific logic. The

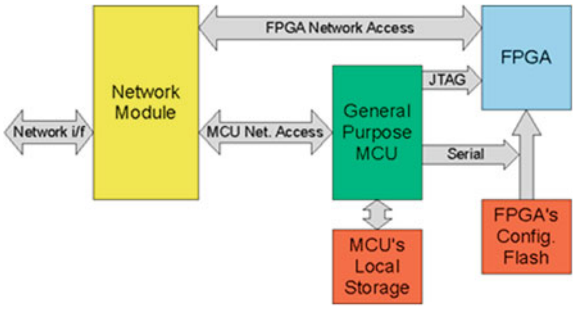


Fig. 18 The RWS Reconfiguration subsystem

reconfiguration process covers two approaches: firmware and architecture modification, both part of RWD instrumentation according to Fig. 13 in Sect. 5. The ability to remotely change hardware configuration parameters is also a vital feature for implementing adaptability in FPGA- or microcontroller-based RWS. The reconfiguration process can be implemented using various hardware and software methods.

In FPGA-based RWS, configuration is stored on add-on Flash memory configuration chips which upload their contents to the FPGA upon power-up or on request. The reconfiguration process itself can proceed in several ways. The remote reconfiguration feature used by the the presented tool (RWS Reconfiguration) relies on a general-purpose microcontroller unit (MCU) locally connected to the FPGA chip and responsible for uploading new configurations (Fig. 18).

In this case the microcontroller is equipped with a wired or wireless network interface and locally accesses FPGA using the Joint Test Action Group (JTAG) interface, the Flash chip using JTAG or dedicated serial interface, or both of these interfaces.

The RWS reconfiguration subsystem can use multiple blocks of local configuration memory in order to store temporary configuration data. As the local FPGA's Flash configuration memory reduces the service reconfiguration downtime, additional local memory in the MCU (Fig. 18) can decrease total reconfiguration time by storing many configuration files. When a new configuration is required, the current one does not need to be overwritten, but can instead be preserved for future reuse. This enables implementation of simple configuration caching, eliminating the need to transfer a new configuration file for each feature. In some implementations the MCU's local storage might be shared between the MCU and an FPGA, and therefore used locally to store data required by the service running on the FPGA.

7 Case Study

The goal of this section is to illustrate the issues discussed earlier on in this chapter. We will show how the AS3 Studio provides support for development of sophisticated applications (see also [17]) and how its adaptability mechanisms can be exploited to transparently maintain the specified quality level of the application's operation.

As already mentioned, advanced service-oriented systems need to provide support for seamless integration of both virtual and real-world services into a single application. Thus, the case study scenario will utilize both types of services and address selected aspects of safety management in the real world (e.g. in an enterprise). Although reliant on specific components, the application presented here can be perceived as a representative of a broader class of solutions related to safety management. We will demonstrate how software services can enhance operation of real-world services, taking over their tasks when necessary—to increase application performance. We will also show that the adaptive infrastructure allows easily introducing various procedures to satisfy application's QoS requirements.

The real-world devices used in this case study have been implemented and instrumented by the authors themselves—their short description is given below.

7.1 Characteristics of the Entrance Protection Application

The general idea behind the application called Entrance Protection is as follows: the face of a person wishing to enter a protected area is captured by a camera. If it is recognized as belonging to an authorized entrant, access is granted and the door lock disengages for several seconds. To ensure sufficient light for the camera a lighting system is installed and connected to a power switch. The switch is activated if a light level detector—enabled just prior to powering up the camera—detects unsatisfactory illumination. The whole process is triggered by a pyrometer which detects rapid, significant temperature changes within the observed area. Figure 19 presents a logical view of this layout with particular elements accessible as services connected to Enterprise Service Bus.

The most interesting part of the system is its pattern recognition feature—so this aspect will be discussed in more detail. A common practice when constructing video surveillance systems is to send the video stream to a central point for processing—in our scenario this involves pattern recognition. In the presented case, however, a much more reasonable approach is to process the data locally at its source avoiding network congestion during data transfer. A digital camera with an embedded pattern recognition module and a library of stored patterns may instead dispatch a simple event, triggering the downstream parts of the business process. Figure 20 presents the Entrance Protection business process in the BPMN notation. The process is triggered by an event generated by the Pyrometer Service when the measured temperature is approximately equal to the normal temperature of a human body. In the "Measure Illuminance" task the Light Detector Service is called to measure illumination. If necessary, the Power Switch service may be used to provide additional lighting. In the next step, the Camera Service is invoked to begin recording and run the recognition algorithm. The service returns its results within a predefined period of time. Upon successful recognition the identifier of the matching pattern is returned and the process progresses to the next step; otherwise it is aborted. In the "Check Access" task an external virtual service is called to check if permission can be granted. If so, the Door Lock service is called to release the door lock and, after 3 more seconds, called again to lock the door.

7.2 Properties of the Real World Devices and Services Used

This section briefly characterizes the Real World Services utilized in our case study. They have been instrumented and are controlled using our universal RWS module providing them with IP communication feature. Consequently, they expose their functionality as SOAP endpoints, which is a common solution applied by third-party devices. Such an approach is fully satisfactory for controlling each device separately; however to be able to easily build sophisticated applications more advanced mechanisms are necessary. Using RWS Builder tool described in the previous section, all Real World Services used in the discussed scenario have been equipped with a dedicated proxy exposing their functionality in the OSGi environment and thus enabling their utilization in the OSGi Monitoring and Management layer of the AS3 Studio. In the presented case study several real-world services are used such as camera, door lock, light detector, power switch and temperature detector.

Camera

The smart camera can be used to implement video surveillance and environment monitoring services. In the presented scenario the smart camera is equipped with an

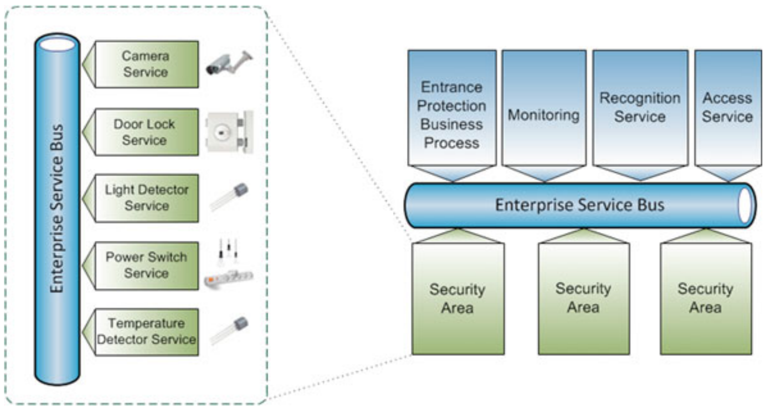


Fig. 19 A single instance of the entrance protection application (*left-hand figure*) and integration of its multiple instances (*right-hand figure*)

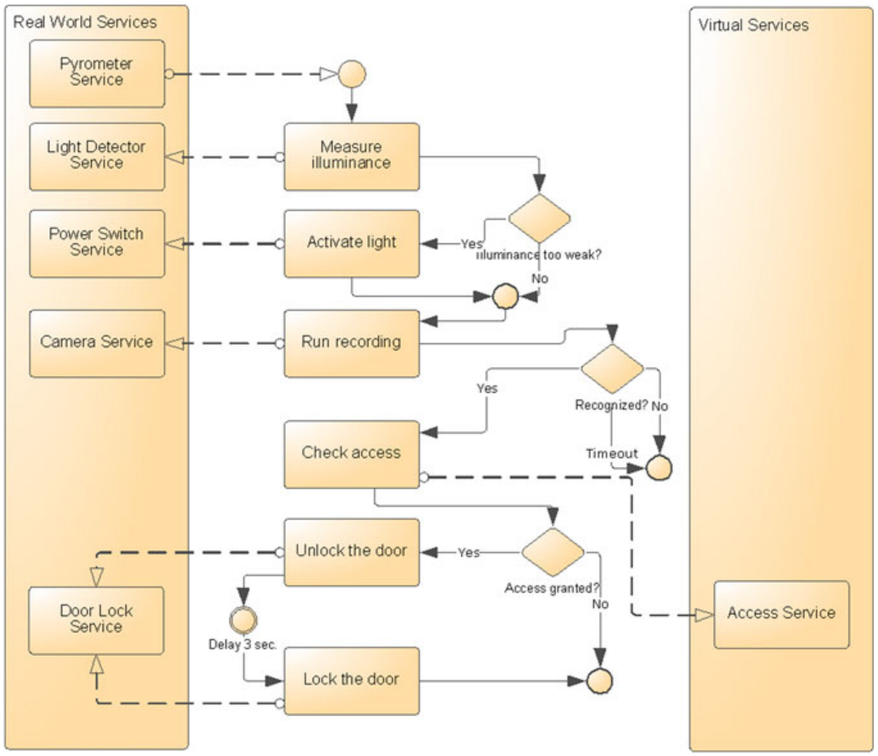


Fig. 20 Entrance Protection business process in the BPMN notation

object classifier functionality and the internal classification algorithm is trained for facial recognition.

The object classifier service is designed to perform classification of similar objects, e.g. faces, car models or road signs. The service applies an algorithm based on Kernel Regression Trees, introduced in [66] and implemented in a fast and massively parallel manner in the FPGA. The classifier's interface consist of methods to begin recognition of an object in the captured scene and retrieve the results of recent recognition runs.

Door Lock

The Door Lock Service is the main actuator unit in the security system described in this section. It is able to remotely lock or unlock the door leading to a restricted area in response to client requests. It assumes the form of an anti-burglary lock service, designed as an add-on module for ordinary door locks available on the market. The system consists of an anti-burglary door lock connected to a servomotor with the help of a custom aluminum hitch. It should be noted that this design does not prevent regular usage of the door lock's mechanism. In the event of a power failure or any other emergency the lock can still can be opened or closed with a key.

Light Detector

Another service used in this scenario is the light level detector. It provides information about light intensity at the monitored area. The result is expressed in lux (lx units). In order to avoid noise and deal with flickering light sources (such as fluorescent bulbs) the average value from several readings is taken. The light detector is designed as an add-on extension for a sensor network node.

Power Switch

The Power Switch Service was designed to allow to remotely control the power supply of four separate mains-powered devices. In the presented case study the service is utilized to turn on or cut off power to additional light sources. The universal RWS module inside the Power Switch uses opto-triacs to supply power.

Temperature Detector

In the presented case study a contactless temperature measurement service is used to measure the temperature of an object in the monitored area. Results reported by this service are used to trigger the whole system's logic. The pyrometric detector used to build the service can measure temperatures between -50 and 350° C. The detector is connected to a Sun SPOT device which is one of the nodes of a sensor network.

7.3 Application Quality Considerations

A system implementing the presented concept may appear quite simple, but to operate correctly in a large enterprise with many entry points and thousands of employers it must properly take into account such aspects as scalability, extendibility and ease of integration with other systems running in the enterprise. Business processes are characterized, among others, by their QoS parameters which are specified in the Service Level Agreement. In the adaptive systems introducing such parameters may trigger automatic adaptation mechanisms whose goal is to meet quality criteria regardless of the complexity of the task or any other external factors. In the discussed case the most critical element of the business process is pattern recognition and thus possibility to introduce adaptation in its operation will be here discussed in more detail. An obvious limitation of the presented approach is camera memory capacity—in large systems storing all relevant patterns in its memory may prove prohibitively expensive, or even impossible. To increase the solution's cost-effectiveness without giving up the advantage of local processing some patterns (for example those with the lowest frequency of matches) may be stored outside of the hardware and processed in software. (In light of this concession, transmission of visual data appears unavoidable, although in order to limit data volume only selected images should be sent—rather than the entire video stream.) Fortunately, the pattern recognition task can easily be distributed among many processing (worker) nodes and performed concurrently. The camera pattern recognition module can either return an identifier associated with the matching pattern or fail to do so if it is unable to find an association. The latter case may occur in one of three situations:

- the person whose face was scanned is not known to the pattern recognition module,
- the quality of the captured image was too low (it was too noisy, etc.),
- the recognition process was not able to complete within the required time limit - what is dependent not only on the number of patterns to be analyzed but also on complexity of the transformations applied to each image.

It is evident that it is possible to influence the percentage of recognition failures mainly in the last case. For a set amount of processing power the relation between processing time and recognition failures is inversely proportional. Two QoS parameters can be introduced to describe this behavior:

- maximum processing time (MPT),
- maximum level of recognition failures (MRF).

To be able to control both parameters' values independently it is necessary to influence accessible processing power. In our case additional virtual services can take some processing from the camera. Our goal is to show how to effectively use adaptability mechanisms provided by AS3 Studio to control the amount of additional processing power automatically and dynamically—satisfying imposed QoS parameters.

7.4 Business Process and Infrastructure Enhancements to Satisfy the Application's Quality Requirements

Taking into consideration the discussed QoS requirements the business process should be enhanced to utilize virtual services which implement pattern recognition algorithms similar to the one provided by the Camera Service - the result of its enhancement is depicted in Fig. 21. If the recognition process performed by the Camera Service fails due to a timeout (see the previous subsection), the Entrance Protection business process may download the image currently being analyzed and distribute it to a number of virtual services along with subsets of pattern identifiers to be examined. Successful recognition by any of the nodes triggers an event which immediately aborts the entire processing task on other nodes. The remainder of the business process is realized without any changes. Operation of this business process

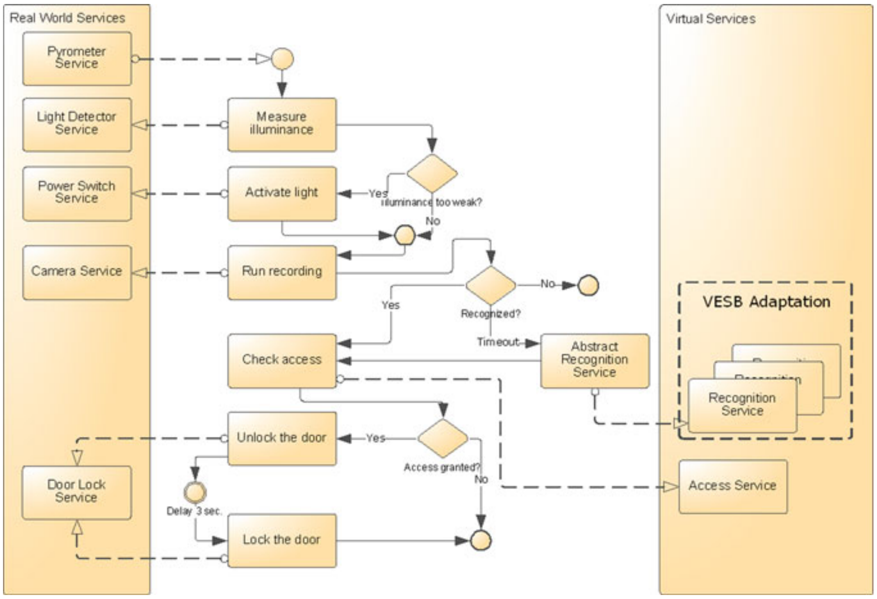


Fig. 21 Enhanced Entrance Protection business process in the BPMN notation

is partially controlled by Adaptation Manager being the part of the Virtual ESB (VESB).

The adaptability mechanisms introduced by VESB enable the business process to be described in an abstract way, without referring to particular service instances. When an abstract composition is instantiated both the number of instances of virtual services and their locations are specified at deployment time. The number of virtual services that perform image recognition may change according to the number of images being processed at the same time [72]. Accordingly, the number of service instances is controlled by the AM deployed in VESB. The Camera Service may contain a database of photos of registered employees, while the photos of trainees might be stored in a database that is accessed by virtual services. When the Camera Service fails to recognize an entrant, that person's picture can instead be processed by a dynamically changing pool of virtual service instances (Recognition Service) implementing the image recognition algorithm. During morning hours, when employees typically arrive at the workplace, the volume of images to be processed is higher than during the rest of the day and the number of virtual service instances should dynamically adapt in order to achieve the desired quality of service. An important issue is how to estimate the number of instances engaged in a particular business process. In this case, a fully sufficient approach is to perform historical analysis of previous executions of the application in the specified timeframe. This data can be gathered transparently by monitoring business process execution and conducting analysis of communication patterns in the Virtual ESB instance dedicated to this application. The presented adaptation strategy may be used to decrease the cost of service maintenance [73]. Additional virtual service instances might be brought online when necessary and then turned off, reducing resource and power consumption while other working instances are capable of providing satisfactory Quality of Experience (QoE).

Monitoring the activity of a VESB instance can lead to enhancements of the recognition process. Many different heuristics can be applied (e.g. noting that people usually enter the building in groups, etc.) The gathered statistics can trigger reconfiguration of the camera service, providing it with a different set of patterns.

7.5 Another Adaptation Procedures

The presented adaptation procedure in not the only one possible; the Entrance Protection application can be extended in many other ways. Table 3 presents some of them, the first one (1) has already been discussed in the previous section and the following ones will be shortly characterized below.

An unusually high number of unsuccessful entry attempts may trigger an alarm or activate additional, more in-depth surveillance mechanisms. Such functionality can be achieved by further enhancing the business process (2a); however a more interesting option is to exploit the monitoring and management features of AS3 Studio (2b). Communication between federated ESB services can be intercepted and analyzed (by a dedicated business interceptor) and sent to the Complex Event Processor which then performs continuous statistical analysis and triggers appropriate actions.

The same approach can be used to determine whether the lighting system works correctly (3a, 3b). If, despite its use, there is still an excessive number of recognition failures, further corrections can be introduced, whether manually or automatically (e.g. turning on additional lamps).

The adaptation process may also affect lower levels of the infrastructure. If software processing is frequently able to cope with cases not correctly handled by hardware, this could be interpreted as an incentive to replace the FPGA-based pattern recognition algorithm or set of stored patterns (4).

8 Conclusions

Adaptive SOA Systems construction is fully justified by complexity of the enterprise class applications deployed nowadays and their changing business and execution requirements. This approach is very much in line with one of an emerging paradigm aiming at simplifying and reducing efforts required to deploy and maintain of complex computer systems such as Autonomic Computing (AC). The detailed analysis shown that exists direct relation between AC properties and the service orientation principles. It is also evident that the MAPE-K pattern could be used as a foundation

Table 3 Various adaptation procedures applied to the Entrance Protection business process

	Observed behavior	Sensor	Effector	Adaptation strategy	Behavior implemented by
1	MRT/MRF exceeded	Monitoring activity in VESB instance	ESB routing mechanism	To keep optimal number of recognition service instances	VESB
2a	Excessive number of unsuccessful entry attempts	Security service (being the part of the business process)	Alarm service	To ensure desired security level	Security service
2b		Business interceptor connected to ESB, OSGiMM			CEP
3a	Excessive number of recognition failures due to noise	Environment service (being the part of the business process)	Light service/manual setting of light	To ensure desired level of MRF	Environment service
3b		Business interceptor connected to ESB, OSGiMM			CEP
4	Considerable number of cases not handled by hardware	Monitoring of the business process, OSGiMM	Trigger replacement of FPGA-based pattern recognition memory	To ensure defined percentage of cases handled in hardware	RWS Builder

of the adaptive SOA system construction. These considerations lead to the adaptive systems space definition and location of Adaptive SOA in this space. This approach is further refinement in the context of S3 layer model of SOA systems.

Pragmatic usage of the MAPE-K Pattern across S3 layer results in AS3 Element definition. This element exposes implementation aspects of the MAPE-K pattern deployment in context of SOA systems. It is used by the AS3 Process which transforms systems built in accordance with the S3 Model into their adaptive versions and managing the adaptation process across different S3 Layers.

The AS3 Pattern may be applied only to selected layers. Leveraging the potential of particular layers adaptation loop execution can enhance the SOA environment with such features as: dynamic service flexibility, policy-driven operation optimization, and cross-business process monitoring and management.

The proposed solution considers adaptability aspects of SOA systems in uniform way referring to SOA applications composed with software services, named also Virtual Services, and hardware components being specified as Real World Services. Such approach is justified by increasing importance of pervasive systems bringing interaction from enterprise systems back to the real world. In this context, adaptive behavior of Real Word Services plays a critical role combining adaptive interaction, adaptive composition and task automation, by involving knowledge regarding user's profile, intentions, and previous use of the system. To clarify the proposed approach the reference model of the Adaptive SOA referring to Real-World and Virtual Services has been proposed and presented in the form of the concept maps.

Taking into account rather complex process of enrichment of the SOA System with adaptability functionality, dedicated software tools which support this transformation are important. The presented approach fully exploits the separation of concerns paradigm which isolates adaptability aspects from application business logic in the development and deployment phases and at the run-time. Such approach is strongly supported by the dynamic and flexible software execution environment offered by OSGi which allows the software modification at the run-time. The similar role plays the FPGA as far as Real-World Services are considered. The properly designed FPGA boards enable possibility for remote firmware modification on demand. Combining these two technologies lead to successful deployment of SOA adaptive systems in practice.

Acknowledgments This work has been performed by many contributors of IT-SOA Projects. The authors wants to thanks especially to: Paweł Bachara, Marcin Jarząb, Robert Szymacha, Dominik Radziszowski, Jacek Kosiński, Kornel Skałkowski, Przemysław Wyszkowski, Sławomir Zieliński. 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. An architectural blueprint for autonomic computing (IBM Corp.), IBM Corp., (2006)
2. Agarwal, A., Sites, R.L., Horowitz, M.: ATUM: a new technique for capturing address traces using microcode. In: ISCA, pp. 119–127 (1986)
3. Arsanjani, A., Zhang, L.J., Ellis, M., Allam, A., Channabasavaiah, K.: S3: A service-oriented reference architecture. IT Prof. **9**, 10–17 (2007)
4. Avgeriou, P., Zdun, U.: Architectural patterns revisited a pattern language. In: 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee, pp. 1–39 (2005)
5. Baker, J., Hsieh, W.: Runtime aspect weaving through metaprogramming. In: Proceedings of the 1st International Conference on Aspect-riented Software Development, AOSD '02, pp. 86–95. ACM, New York (2002)
6. Barbon, F., Traverso, P., Pistore, M., Trainotti, M.: Run-time monitoring of instances and classes of web service compositions. In: Proceedings of the IEEE International Conference on Web Services, ICWS '06, pp. 63–71. IEEE Computer Society, Washington (2006)
7. Baresi, L., Guinea, S.: Towards dynamic monitoring of ws-bpel processes. In: Proceedings of the Third International Conference on Service-Oriented Computing, ICSOC'05, pp. 269–282. Springer, Berlin (2005)
8. Bieberstein, N., Bose, S., Fiammante, M., Jones, K., Shah, R.: Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap. IBM Press, Upper Saddle River (2005)
9. Bigus, J.P., Schlosnagle, D.A., Pilgrim, J.R., Mills, Diao, Y.: Able: a toolkit for building multiagent autonomic systems. IBM Sys. J. **41**(3) (2002). doi:10.1147/sj.413.0350
10. Blair, G.S., Bencomo, N., France, R.B.: Models@run.time. IEEE Comput. **42**(10), 22–27 (2009)
11. Bobda, C.: Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications, 1st edn. Springer Publishing Company, New York (2007)
12. Cardoso, J.M., Simoes, J.B., Correia, C.M.B.A., Combo, A., Pereira, R., Sousa, J., Cruz, N., Carvalho, P,. Varandas, C.A.F.: A high performance reconfigurable hardware platform for digital pulse processing (2004)
13. Chen, C., Li, L., Wei, J.: Aop based trustable sla compliance monitoring for web services. In: Proceedings of the Seventh International Conference on Quality Software, QSIC '07, pp. 225–230. IEEE Computer Society, Washington (2007)
14. Chen, I.Y., Ni, G.K., Lin, C.Y.: A runtime-adaptable service bus design for telecom operations support systems. IBM Syst. J. **47**(3), 445–456 (2008)
15. Jayapandian, J.: Embedded control and virtual instrument simplifies laboratory automation. Curr. Sci. **90**(6), 765–770 (2006)
16. Corporation, C.S.: PSoC—Technical Reference Manual (TRM) (2006)
17. Czekierda, L., Masternak, T., Zielinski, K.: Evolutionary approach to development of collaborative teleconsultation system for imaging medicine. IEEE Trans. Inf. Technol. Biomed. **16**(4), 550–560 (2012). doi:10.1109/TITB.2012.2194506
18. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: M. Sloman, J. Lobo, E. Lupu (eds.) POLICY Lecture Notes in Computer Science, vol. 1995, pp. 18–38. Springer, Berlin (2001)
19. DIGI: http://www.digi.com/ (2012)
20. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River (2005)
21. Estrin, G.: Reconfigurable computer origins: the ucla fixed-plus-variable (f+v) structure computer. IEEE Ann. Hist. Comput. **24**(4), 3–9 (2002). doi:10.1109/MAHC.2002.1114865
22. Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., Jzquel, J.M.: Modeling and validating dynamic adaptation. In: M.R.V. Chaudron (ed.) MoDELS Workshops Lecture Notes in Computer Science, vol. 5421, pp. 97–108. Springer, Berlin (2008)
23. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer **37**, 46–54 (2004)

24. Goldberg, A., Havelund, K.: Instrumentation of java bytecode for runtime analysis. In: Proc. Formal Techniques for Java-like Programs. Technical Reports from ETH vol. 408(2003).
25. Gorton, I., Liu, Y., Trivedi, N.: An extensible and lightweight architecture for adaptive server applications. Softw. Pract. Exper. **38**(8), 853–883 (2008)
26. Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., Savio, D.: Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. IEEE Trans. Serv. Comput. **3**(3), 223–235 (2010)
27. Guinard, D., Trifa, V., Spiess, P., Dober, B., Karnouskos, S.: Discovery and on-demand provisioning of real-world web services. In: IEEE International Conference on Web Services, ICWS 2009, Los Angeles, CA (2009)
28. Hallsteinsen, S., Floch, J., Stav, E.: A middleware centric approach to building self-adapting systems. In: Proceedings of the 4th International Conference on Software Engineering and Middleware, SEM'04, pp. 107–122. Springer, Berlin (2005)
29. Heinrich, C.: RFID and beyond—growing your business through real world awareness. Wiley, Heinrich (2005)
30. Hayman, C.: The benefits of an open service oriented architecture in the enterprise, (IBM Corp.), IBM Corp., (2005)
31. Jacob, B., Lanyon-Hogg, R., Nadgir, D.K., Yassin, A.F.: A Pratical Guide to IBM Autonomic Computing Toolkit. IBM Corp. (2004).
32. Janiesch, C., Niemann, M., Steinmetz, R. (eds.): The TEXO Governance Framework, SAP Research Brisbane, White Paper, Version 1.1, Working Draft (2011)
33. Kaiser, G.E., Parekh, J.J., Gross, P., Valetto, G.: Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems. In: Active Middleware Services, pp. 22–31. IEEE Computer Society, USA (2003)
34. Karnouskos, S., Bangemann, T., Diedrich, C.: Integration of legacy devices in the future soa-based factory. In: 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM), Moscow, Russia (2009)
35. Karnouskos, S., Savio, D., Spiess, P., Guinard, D., Trifa, V., Baecker, O.: Real world service interaction with enterprise systems in dynamic manufacturing environments. In: L. Benyoucef, B. Grabot (eds.) Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management. Springer (2010) ISBN: 978-1-84996-118-9
36. Kean, T., Buchanan, I.: The use of fpga's in a novel computing subsystem. In: First International ACM Workshop on Field Programmable Gate Arrays (1992)
37. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003). doi:10.1109/MC.2003.1160055
38. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: ECOOP. Springer, Berlin (1997)
39. Klein, C., Schmid, R., Leuxner, C., Sitou, W., Spanfelner, B.: A survey of context adaptation in autonomic computing. In: Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems, ICAS '08, pp. 106–111. IEEE Computer Society, Washington (2008) doi:10.1109/ICAS.2008.23
40. Kon, F., Campbell, R.H., Mickunas, M.D., Nahrstedt, K., Dennis, M., Nahrstedt, M.K., Ballesteros, F.J.: 2k: A distributed operating system for dynamic heterogeneous environments. In: 9th IEEE International Symposium on High Performance, Distributed Computing, pp. 201–210 (1999)
41. Bug Labs: Bug Labs: modular, open source hardware (2009)
42. Lee, K., Sakellariou, R., Paton, N.W., Fernandes, A.A.A.: Workflow adaptation as an autonomic computing problem. In: Proceedings of the 2nd workshop on Workflows in Support of Large-Scale Science, WORKS '07, pp. 29–34. ACM, New York (2007)
43. Lin, K.J., Panahi, M., Zhang, Y., Zhang, J., Chang, S.H.: Building accountability middleware to support dependable soa. IEEE Internet Comput. **13**(2), 16–25 (2009)
44. Lodi, A., Toma, M., Campi, F., Cappelli, A., Canegallo, R., Guerrieri, R.: A vliw processor with reconfigurable instruction set for embedded applications. IEEE J. Solid-State Circuits **38**(11), 1876–1886 (2003)

45. McAffer, J., Lemieux, J.M.: Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications. Addison-Wesley, Upper Saddle River (2005)
46. milkymist.org: milkymist.org (2009)
47. Montenegro, G., Kushalnagar, N., Hui, J., Culler, D.: Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard) (2007)
48. Morin, B., Barais, O., Jzquel, J.M., Fleurey, F., Solberg, A.: Models@run.time to support dynamic adaptation. IEEE Comput. **42**(10), 44–51 (2009)
49. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive monitoring and service adaptation for ws-bpel. In: Proceedings of the 17th International Conference on World Wide Web, WWW '08, pp. 815–824. ACM, New York (2008).
50. Mudry, P.A., Vannel, F., Tempesti, G., Mange, D.: Confetti : a reconfigurable hardware platform for prototyping cellular architectures. In: IPDPS, pp. 1–8. IEEE (2007)
51. Niemann, M., Eckert, J., Repp, N., Steinmetz, R.: Towards a generic governance model for service oriented architectures. In: AMCIS'08, pp. 361–361 (2008)
52. OASIS Web Services Discovery and Web Services Devices Profile (2005)
53. Opencores.org. http://opencores.org/
54. OSGi Alliance: OSGi Service Platform Release 4. [Online]. http://www.osgi.org/Main/HomePage. (2007)
55. Patel, S.V., Pandey, K.: Soa using aop for sensor web architecture. In: Proceedings of the 2009 International Conference on Computer Engineering and Technology—Volume 02, ICCET '09, pp. 503–507. IEEE Computer Society, Washington (2009)
56. Popovici, A., Alonso, G., Gross, T.: Just-in-time aspects: efficient dynamic weaving for Java. In: Proceedings of the 2nd International Conference on Aspect-Oriented Software Development, AOSD '03, pp. 100–109. ACM, New York (2003). doi:10.1145/643603.643614
57. Popovici, A., Gross, T., Alonso, G.: Dynamic weaving for aspect-oriented programming. In: Proceedings of the 1st International Conference on Aspect-Oriented Software Development, AOSD '02, pp. 141–147. ACM, New York (2002)
58. Prasanna, D.R.: Dependency Injection, 1st edn. Manning Publications Co., Greenwich (2009)
59. Prez, F.M., Abarca, J.A.G.M., Morillo, H.R., Gimeno, F.J.M., Jorquera, D.M., Iglesias, V.G.: Wake on lan over internet as webservice system on chip. IEEE Trans. Industr. Electron. **16**(1), 45–69 (2012)
60. Psiuk, M.: AOP-based monitoring instrumentation of JBI-compliant ESB. In: Proceedings of the 2009 Congress on Services—I, SERVICES '09, pp. 570–577. IEEE Computer Society, Washington (2009)
61. Psiuk, M., Bujok, T., Zielinski, K.: Enterprise service bus monitoring framework for soa systems. IEEE Trans. Serv. Comput. **5**(3), 450–466 (2012). doi:10.1109/TSC.2011.32.
62. Psiuk, M., Zmuda, D., Zieliski, K.: Distributed OSGI built over message-oriented middleware (2011). doi:10.1002/spe.1148. http://dx.doi.org/10.1002/spe.1148
63. Rellermeyer, J.S., Duller, M., Gilmer, K., Maragkos, D., Papageorgiou, D., Alonso, G.: The software fabric for the internet of things. In: C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, S.E. Sarma (eds.) IOT. Lecture Notes in Computer Science, vol. 4952, pp. 87–104. Springer, Berlin (2008)
64. Rochwerger, B., Breitgand, D., Epstein, A., Hadas, D., Loy, I., Nagin, K., Tordsson, J., Ragusa, C., Villari, M., Clayman, S., Levy, E., Maraschini, A., Massonet, P., Mun andoz, H., Tofetti, G.: Reservoir—when one cloud is not enough. Computer **44**(3), 44–51 (2011). doi:10.1109/MC.2011.64
65. Ruta, A., Brzoza-Woch, R., Zielinski, K.: On fast development of fpga-based soa services—machine vision case study. Design Autom. Emb. Syst. **16**(1), 45–69 (2012)
66. Ruta, A., Li, Y., Liu, X.: Robust class similarity measure for traffic sign recognition. IEEE Trans. Intell. Transp. Syst. 846–855 (2010)
67. Satyanarayanan, M.: Pervasive computing: vision and challenges. IEEE Pers. Commun. **8**, 10–17 (2001)
68. Shelby, Z., Bormann, C., Frank, B.: Constrained application protocol (coap). IETF Internet draft, 1–81 (2011)

69. Smith, R.B.: Spotworld and the sun spot. In: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, IPSN '07, pp. 565–566. ACM, New York (2007). doi:10.1145/1236360.1236442.
70. Song, H., Lee, S.H., Lee, S., Lee, H.S.: 6lowpan-based tactical wireless sensor network architecture for remote large-scale random deployment scenarios. In: Proceedings of the 28th IEEE Conference on Military Communications, MILCOM'09, pp. 1044–1050. IEEE Press, Piscataway (2009)
71. Sun, M., Li, B., Zhang, P.: Monitoring BPEL-based web service composition using AOP. In: Proceedings of the 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science, ICIS '09, pp. 1172–1177. IEEE Computer Society, Washington (2009)
72. Szydlo, T., Zielinski, K.: Method of adaptive quality control in service oriented architectures. In: Proceedings of the 8th International Conference on Computational Science, Part I, ICCS '08, pp. 307–316. Springer, Berlin (2008)
73. Szydlo, T., Zielinski, K.: Adaptive enterprise service bus. New Generation Comput. **30**(2–3), 189–214 (2012)
74. Tibbo: http://www.tibbo.com/(2012)
75. Vedamuthu, A.S., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., Yalinalp, M.: Web services policy framework (wspolicy) http://www.w3.org/TR/ws-policy (2007)
76. Voros, N.S., Masselos, K. (eds.): System Level Design of Reconfigurable Systems-on-Chip. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
77. Vuković, M.: Context aware service composition, PhD dissertation, Univ. of Cambridge (2007)
78. Walsh, W.E., Tesauro, G., Kephart, J.O., Das, R.: Utility functions in autonomic systems. pp. 70–77 (2004). doi:10.1109/ICAC.2004.1301349.
79. Wang, Q., Shao, J., Deng, F., Liu, Y., Li, M., Han, J., Mei, H.: An online monitoring approach for web service requirements. IEEE Trans. Serv. Comput. **2**(4), 338–351 (2009)
80. Wetzstein, B., Strauch, S., Leymann, F.: Measuring performance metrics of ws-bpel service compositions. In: J.L. Mauri, V.C. Giner, R. Tomas, T. Serra, O. Dini (eds.) Proceedings of the Fifth International Conference on Networking and Services, ICNS 2009, 20–25 April 2009, Valencia, Spain, pp. 49–56. IEEE Computer Society, Washington (2009)
81. Wildstrom, J., Witchel, E.J., Mooney, R.: Towards self-configuring hardware for distributed computer systems. In: Proceedings of the Second International Conference on Automatic Computing, ICAC '05, pp. 241–249. IEEE Computer Society, Washington (2005)
82. Yeung, K., Kelly, P.H.J., Bennett, S.: Performance Analysis and Grid computing. Dynamic Instrumentation for Java Using a Virtual JVM, pp. 175–187. Kluwer Academic Publishers, Norwell (2004)
83. Yuan, H., Choi, S.W., Kim, S.D.: A practical monitoring framework for esb-based services. In: Proceedings of the 2008 IEEE Congress on Services Part II, SERVICES-2 '08, pp. 49–56. IEEE Computer Society, Washington (2008)
84. Zeeb, E., Bobek, A., Bohn, H., Prter, S., Pohl, A., Krumm, H., Lck, I., Golatowski, F., Timmermann, D.: Ws4d: Soa-toolkits making embedded systems ready for web services, In: Proceedings of the Open Source Software and Product Lines Workshop (OSSPL07) (2007)
85. Zielinski, K., Szydlo, T., Szymacha, R., Kosinski, J., Kosinska, J., Jarzab, M.: Adaptive SOA solution stack. IEEE Trans. Serv. Comput. **5**, 149–163 (2012). http://doi.ieeecomputersociety.org/10.1109/TSC.2011.8
86. Zmuda, D., Psiuk, M., Zielinski, K.: Dynamic monitoring framework for the SOA execution environment. Procedia CS **1**(1), 125–133 (2010)