We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

20

Model Driven Adaptive Quality Control in Service Oriented Architectures

Tomasz Szydło and Krzysztof Zieliński
AGH University of Science and Technology
Poland

1. Introduction

Service-Oriented Architecture (SOA) enables the development of applications that are built by combining loosely coupled and interoperable services (Erl, 2004). In SOA, the centre of gravity shifts from development to integration. An application in SOA is composed from a set of services that are connected together in the integration platform. Such an application is described as an abstract composition which can be executed in containers (e.g. the Enterprise Service Bus), responsible for mapping abstract services to concrete service instances during execution. The mapping process can be performed by a user or planner prior to execution or during runtime. It should take into account Quality of Service aspects, enabling quality control to become an integral part of SOA application execution.

Automation of quality control can explore the concept of adaptive or autonomic systems (Ganek & Corbi, 2003). An adaptive software system enables software to modify its structure and behaviour in response to changes in its execution environment (McKinley et al., 2004). Adaptation can be differentiated into static or dynamic, depending on when it takes place. Static adaptation assumes that adaptive behaviour is *hardwired* or configured before the application starts. Dynamic adaptation processes enable extensions or replacement of system components during execution without stopping and restarting the system. Thus, this category of adaptation is particularly suitable for SOA applications.

Autonomic systems (Kephart & Chess, 2003) represent a more mature step in the evolution of dynamic adaptive systems. In such systems the operator does not influence the system directly, but only defines general policies which specify system behaviour. The system is able to collect new knowledge and use it in the adaptation process. In the case of SOA applications, this means that integrated components may be dynamically managed by business rules and policies, taking into account experience gathered from former application runs.

In spite of the differences between adaptive and autonomic systems, their general design follows the same MAPE (Monitor, Analyze, Plan, and Execute) (IBM, 2006) paradigm. This very well known paradigm has not yet been widely adopted by the SOA application execution environment, creating an interesting space for research. Its practical exploitation requires suitable mechanisms implementing the MAPE control process to be embedded into execution containers for SOA services. The analysis and planning steps for SOA applications require a definition of metrics and their calculation on the basis of monitoring data. The values of these metrics could be used for the SOA application execution model construction,

which may subsequently be utilized in the planning step. The model represents knowledge about the SOA application execution history and service interdependency. Its construction is somewhat optional, as a planning strategy might exploit only online monitoring parameters which refer to simple services.

This chapter presents the model-driven adaptive SOA application execution environment as a more mature version of adaptive systems. Such an execution environment is a promising approach to managing complexity, leveraging software models which we refer to as *models@run.time* (Blair et al., 2009). Such a model is defined as a causally connected self-representation of the associated system that emphasizes the structure, behaviour or goals of the system from a problem space perspective. Runtime models provide "abstractions of runtime phenomena" and support reasoning. They may used for dynamic state monitoring and control of systems during execution or to dynamically observe the runtime behaviour in order to understand a specific behavioural phenomenon.

The chapter focuses on the construction of SOA application execution models and mechanisms supporting the MAPE process which allow it to be used for quality control at runtime. The proposed approach takes into account the fact that service delivery is intimately connected to Quality of Service and Quality of Experience requirements, expressed through contracts between service providers and end users. It is also essential that SOA applications – frequently composed at runtime – offer new theoretical and practical possibilities for QoS/QoE management. These aspects are very important for research on next-generation SOA (Erl, 2010) technologies, dealing with management, governance and support for composite services through lightweight orchestration.

The structure of the proposed chapter is as follows. First, the requirements of QoS/QoE control in SOA systems are presented. A clear distinction between QoS and QoE is introduced. The taxonomy of QoS metrics for SOA systems is also presented. When referring to QoS we note that this term is used to describe quality from the point of view of providers as well as clients. Paradoxically, system users might be satisfied with high QoE even if the system itself does not implement all the functionality paid for by the clients. On the other hand, fulfilment of all required quality aspects does not automatically result in user satisfaction if QoE remains low. The importance of QoE has strong temporal characteristics as it is frequently analyzed in brief intervals, whereas QoS is usually averaged over a longer period of time. It should also be noted that clients may disagree with the provider as to the overall quality of the system. Since major differences between these assessments are commonplace, they should also be taken into account when considering quality control in SOA systems.

Following this discussion, model-driven adaptive quality control of SOA applications is presented. A statistical model of SOA application execution which could be used for this purpose is introduced and its runtime updates are considered. An adaptation process based on a fitness function is formulated. The computation of this function relies on the proposed statistical model of SOA application execution. This part of the chapter is based on ongoing research by the authors (Szydlo & Zielinski,2008; Szydlo,2010). The following part of the chapter considers adaptability mechanisms for SOA systems. They exploit the fact that each SOA application is composed of a set of services linked to one another via the integration platform – ESB. Such an application can be described as an abstract composition and then executed in an ESB container responsible for mapping abstract services onto specific instances in the course of service processing. The mapping process can be performed by the user or planner prior to execution or it can take place at runtime. Such an approach allows

instances to be replaced during execution if QoS/QoE requirements are to be fulfilled. This, in turn, leads to practical implementation of QoS-driven composition as a foundation of the investigated runtime adaptability concept. Sensors and effectors for adaptation activity are described and location of their installation within the ESB architecture is explained. The usability of the proposed approach is illustrated by a case study. The chapter ends with conclusions.

2. Quality management in SOA systems

Quality management is a crucial element as it ensures that systems meet their requirements with respect to specific performance metrics. Dealing with QoS is a sign that technology is progressing beyond initial experimentation to a production deployment stage – as in the case of service orientation. In order to manage quality, a set of metrics has to be defined. Clients and providers are bound by SLAs which define the terms and conditions of service quality that the provider must deliver to customers. A key aspect of SLA is QoS information which consists of several criteria such as execution duration, availability, execution time, and many others. SLA also comprises financial information such as the price for using a service, and the way in which penalties are calculated. To define quality requirements, a common understanding of offered quality between users and providers is necessary (Dobson & Sanchez-Macian, 2006).

It is important for both consumer and provider to understand metric semantics (Ludwig, 2003). Clear definitions must be associated with particular parameters (e.g. response time as average value or the maximum response time over the last ten minutes). Similarly, the point of measurement is important. Metric values gathered on the client side are often different than those gathered within the service container or the service itself.

Fig. 1. Perceiving quality

Analysis of existing frameworks in terms of QoS shows that this term is used interchangeably for describing quality from the provider as well as from the client point of view. Authors have therefore decided to use term Quality of Experience (QoE) for Quality of Service observed at the client endpoint. This idea is presented in Fig. 1. Paradoxically, system users might be satisfied with system behaviour that exhibits high QoE even if the system does not implement all the functionality for which the clients actually paid. On the other hand, the system may fulfil all the required quality aspects, but users might still not be satisfied given low QoE. The importance of QoE has a temporal character as well, as it is observed over a finite interval, whereas QoS is more of an average statistic. Client may disagree with the system provider as to the quality of the system. In fact, it is very common for both assessments to differ fundamentally.

The definition and evaluation of metrics is essential for model-driven adaptive quality control of SOA applications as they could be composed at runtime with services offered by many providers. It is also a fundamental element of the MAPE process and the execution model. This is why the metrics classification used for in the context of SOA applications should be considered in more detail.

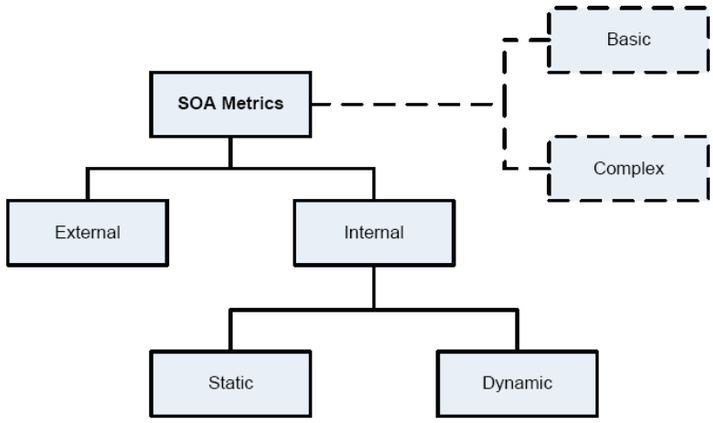

Fig. 2. Taxonomy of SOA metrics

We refer the reader to the taxonomy presented in Fig. 2 which distinguishes the following categories:

- **Internal and External Metrics** Internal metrics addresses solely those aspects of a system which are available exclusively for system architects. They might provide detailed information on source code, or the number of used software modules. External metrics are available for system users and provide information about system reliability, efficiency or response time. Evaluation of external metrics is more difficult than of internal ones, as such metrics often have subjective meaning. Additionally, metrics calculated at early stages of product development may not reflect the values of final versions;
- **Static and Dynamic Metrics** Static metrics describe invariable parameters. These might include parameters derived from software documentation or semantic description, or metrics describing source code. Examples of such metrics include configuration and security aspects of software (Hershey et al., 2007). Dynamic metrics are evaluated during runtime. Sample metrics of this kind include execution time and network throughput;
- **Basic and Complex Metrics** Basic metrics are directly gathered from system monitoring. They might provide simple information about system behaviour, e.g. the time necessary to deliver a message. Complex metrics are combinations of basic ones. For example, response time might be defined as the time needed to deliver a message to a service plus the time needed to process this message.

A number of authors have tried to organize metrics for SOA (Rud et al, 2007) (Hershey & Runyon, 2007); however, in practice, most such schemas are variations of availability, execution time and throughput.

The presented metrics could refer to simple or composite services which constitute an SOA application. Typically, computing composite service metrics requires additional information e.g. on how the execution of one service may impact the execution of another service (Menasce, 2004). The problem becomes even more complicated if services are invoked in parallel. Such additional information may be provided by the execution model which typically describes existing interdependencies. This problem is further elaborated in Section 3 where the SOA application execution model is presented.

3. Model driven adaptive quality control in SOA systems

Runtime adaptability requires detailed information on how the system behaves and how certain services influence others. This information is necessary for proper formulation and implementation of adaptation strategies. The use of models for SOA adaptability seems to be a promising approach (Blair et al., 2009). Models represent knowledge about the working system and hide unnecessary information, providing only facts related to the problem on hand. An execution model may be constructed on the basis of monitoring data collected while observing service QoS/QoE and execution infrastructure load. Updating models at runtime enables the system to respond to changes in the execution environment. The model may then be used to modify composition of the application in order to achieve high-level system management goals. Such modifications address service selection, binding protocol and interaction policy choices.

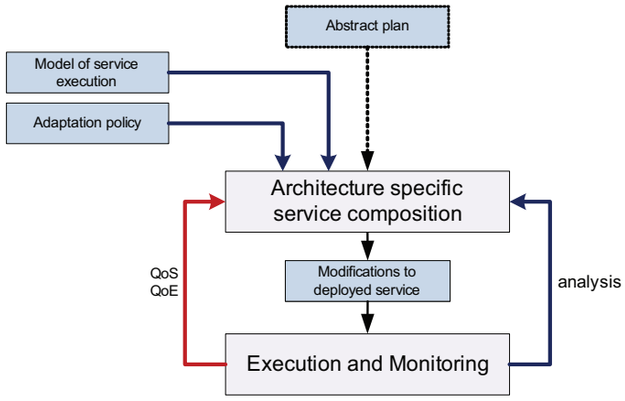

Fig. 3. Model-driven adaptation

The concept of model-driven adaptation is presented in Fig. 3. The service provider composes an application in a selected technology and provides an adaptation policy to the system along with a composite service execution model. The architecture-specific service composition layer continuously modifies the deployed service according to the provided adaptation policy. The abstract plan which acts as input for architecture-specific service composition might be hidden and used only by the programmer during application development. The knowledge of how the system behaves is represented by the service execution model. For this purpose, statistical models can be used. The use of such models for adaptive quality control requires mechanisms for model identification, updates and exploitation in the adaptation process.

3.1 Statistical models of composite service execution

Composite service execution is a collection of activities applied during each service invocation. In other words, each invocation is a particular flow in a graph of possible activities represented as a model *M*. By observing the composite service activity at a given point in time, these flows can be described by statistical data associated with edges of a graph. Thus, the model is composed of a graph and static information which can be provided by the system designer or identified automatically based on system activity monitoring. Once used for adaptation, the model can be updated at runtime, reflecting the current behaviour of the system.

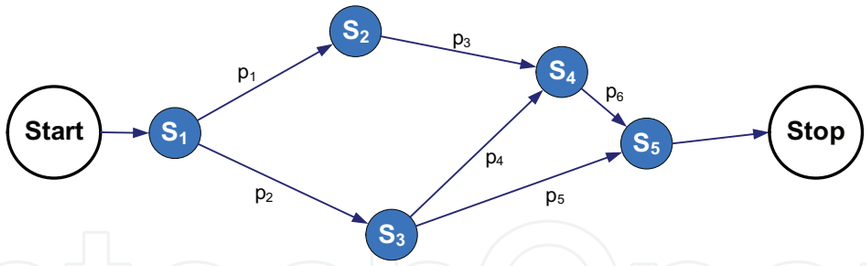

Fig. 4. Sample model of complex service execution at $Level_1$

A formal composite service execution model is represented as a *Directed Acyclic Graph (DAG) M = (V;E)* and includes two special nodes: *Start* and *Stop*. At $Level_1$, nodes in this model represent abstract services – specified by description of their functionality, e.g. WSDL files, while at $Level_2$ particular *Service* instances (specific services) are considered. A sample model of complex service execution at $Level_1$ is shown in Fig. 4. Edge $(s1; s2) \in E$ means that service s_1 communicates with service s_2 with probability $p(s1; s2)$. For each vertex s_j the probabilities on edges originating in a given node add up to 1 provided that successors are not invoked in parallel:

$$\sum_{s_j \in SUCC(s_k)} p\left(s_j, s_k\right) = 1$$

Once the model is defined and statistical information is collected, it can be used for adaptation purposes. From the service provider's perspective, statistical information can be very valuable because it enables identification of the busiest services and bottlenecks, as well as optimization of resource allocation.

3.2 Model updates at run-time

Updating statistical models at runtime strongly depends on the implementation technology of integration middleware. SOA leverages loose coupling of services, which might be connected by additional infrastructural elements, e.g. Enterprise Application Integration (EAI) patterns (Hohpe & Woolf, 2003).

The composite service execution model at $Level_1$ contains abstract services, whereas in the execution environment each abstract service might be represented by several instances, used interchangeably. Hereinafter, the number of messages sent between abstract services s_1 and s_2 is understood as a sum of messages sent between all instances of these abstract services.

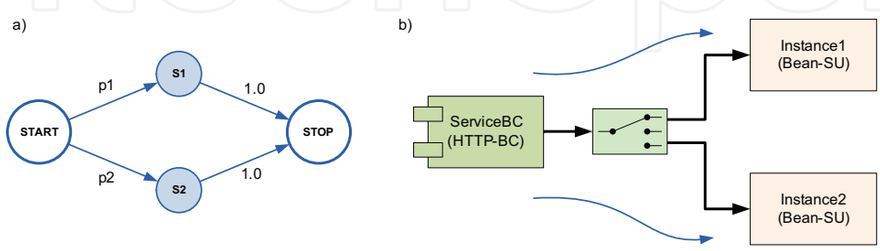

Fig. 5. Model synchronization a) complex service execution model; b) EAI model

When a model contains all the services involved in message passing, calculating the probabilities for model edges becomes straightforward – they reflect the number of

messages sent between services divided by the total sum of messages originating at a source node. Nevertheless, the complex service execution model may not contain all the services deployed in ESB, as shown in Fig. 5 where the EAI content router is excluded from the model. This is justified by the fact that such an element plays a strictly technical role and does not influence the functionality of the composite service. It has also been optimistically assumed that the number of messages sent between services s_1 and s_2 is the minimum number of messages sent between services that are on the shortest path between s_1 and s_2.

3.3 Adaptation process

The proposed execution model can be used for computation of expected values of selected metrics if some service instances take part in execution. The adaptation process may analyze all the possible combinations of service instance invocation flows that might occur in the execution process and select one for execution, based on metric values.

When decisions are made locally, i.e. without analysis of how a particular service instance may affect the overall application, the results often lead to unsatisfactory solutions. The concept of statistical models presented in this chapter enables estimating global behaviour of systems when a certain invocation flow of instances is selected for execution.

Referring to the example in Fig. 4, let us assume that for services $s_1,\ldots, s_5$, instances $i_1,\ldots,i_5$ are selected. Thus, the presented model now is at $Level_2$. Let us now consider the execution time metric. Let t_k be the execution time of service i_k. If $Time_C$ is the predicted total execution time for subsequent invocations of a composite service, then for the presented model and selected instances, it can be given as:

$$\begin{aligned} Time_C = {} & p_1 p_3 p_6 \left(t_{i_1} + t^{RTT}_{i_1,i_2} + t_{i_2} + t^{RTT}_{i_2,i_4} + t_{i_4} + t^{RTT}_{i_4,i_5} + t_{i_5} \right) \\ & + p_2 p_4 p_6 \left(t_{i_1} + t^{RTT}_{i_1,i_3} + t_{i_3} + t^{RTT}_{i_3,i_4} + t_{i_4} + t^{RTT}_{i_4,i_5} + t_{i_5} \right) \\ & + p_2 p_5 \left(t_{i_1} + t^{RTT}_{i_1,i_3} + t_{i_3} + t^{RTT}_{i_3,i_5} + t_{i_5} \right) \end{aligned}$$

The composite service model at $Level_1$ may generate several models C at $Level_2$ because for each abstract service several instances may be available. Given a set of metrics Q, for each service C a QoS_C vector can be calculated. For each *i-th* metric $QoS^i_C \in [0;1]$ is the value of that metric based on historical invocations of a complex service and information stored in the composite service execution model.

There might be several ways to select the most suitable solution. For example, let us define a fitness function that determines how well a selected set of services matches user expectations. The weights of each metric express its importance in the overall fitness indicator:

$$fitness(QoS_C) = \sum_i w_i \times QoS^i_C$$

Providing exact weights for each metric is a multidimensional decision problem. Despite its inherent complexity, it can be approached through decomposition. The Analytic Hierarchy Process (Forman & Selly, 2001) is a technique based on mathematics and human psychology for prioritizing elements of a decision problem. For each pair of metrics the user specifies which one is preferred, in the form of a fraction between 1/9 and 9/1. The result of AHP is a

vector of weights w_i for each metric i. From the number of possible solutions, one chooses the solution with the highest fitness factor and modifies the execution container in order to use the selected service instances. A different approach assumes the use of QoE during adaptation. For example, when the client pays for a particular level of quality, the system may constantly switch between sets of service instances so that the client finally receives what he paid for.

The presented concept of quality control requires suitable mechanisms which should extend the SOA integration to support execution of service instances selected by the adaptation policy. Contrary to other solutions (Vuković, 2006; Gubala et al., 2006; Chafle et al., 2006), the proposed approach requires the service provider to compose applications in the selected technology and then provide an adaptation policy along with a model of composite service execution. The adaptation system monitors the deployed composite services and modifies their operating parameters in accordance with the provided adaptation policy. All these stages of adaptation are described in detail in next section, referring to the MAPE paradigm.

4. Adaptability mechanisms for SOA systems

The MAPE paradigm consists of four sets of functionality: monitor, analyze, plan and execute. Analyzing and planning responses to changes in the execution infrastructure can be affected by model-driven adaptation described in the previous section. Other sets of functionality are strictly related to the managed resource. Monitor functions collect details from the managed resource and correlate them into symptoms that can be analyzed. The execute functions provide mechanisms for performing necessary changes to the system.

Currently, the Enterprise Service Bus (ESB) looms large as a promising approach for deployment of SOA applications. Most of the ESB implementations are compliant with Java Business Integration (JBI), developed under the Java Community Process (JCP) and specified as Java Specification Request (JSR) 208, as an approach to implementing SOA. ESB provides mechanisms for message normalization, routing between selected components and location transparency, making it possible to access services by name without prior knowledge of their exact location. This is why ESB is the suitable place for mechanisms required by the adaptation process.

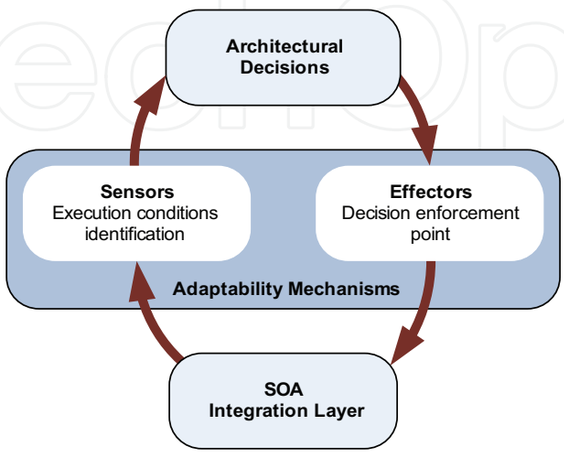

Fig. 6. Extending the integration layer with adaptability mechanisms

Fig.6 depicts a processing cycle based on the MAPE paradigm and corresponding to a standard control/adaptation loop. The Architectural Decision block continuously monitors the system and handles events whenever an action needs to be taken. It monitors the environment using *sensors*, analyses what is found and then plans and executes specific actions through *effectors* operations. *Sensors* monitor current system state while *effectors* have the ability to change the state or configuration in order to perform adaptation.

Sensors

An adaptive system acts in accordance to various goals, requiring several sorts of information gathered from ESB. In most cases, this information will be disjunctive, so one would expect to deploy specialized types of sensors rather than generic ones. It has been found that the interceptor design pattern (Schmidt et al., 2000) fulfils requirements for dynamic installation and reinstallation of sensors. This leads to the conclusion that sensors can be implemented as interceptors and installed whenever necessary.

The most appropriate location of sensors is the place where inter-service/inter-component message transmission takes place. JBI specification defines elements which participate in sending and passing messages. A message is created in a service, passes through a specialized Service Engine (SE) or Binding Component (BC) and is then sent (through a Delivery Channel – DC) to a Normalized Message Router (NMR), which reroutes it to a particular destination through the same components. Injecting interceptors into services or SEs requires knowledge of various types of services and engines. Thus, a better place to inject message exchange interceptors is the DC through which every SE is connected to NMR.

The amount of monitoring information gathered from ESB would be overwhelming. A more suitable approach is to correlate or aggregate events and send notifications only if complex events take place. For this purpose, the stream of simple events from ESB could be processed by a Complex Event Processor (CEP) (Luckham, 2002) which, in due time, would notify the Architectural Decision block.

Effectors

Implementing an adaptation policy requires action, i.e. a set of operations which modify the execution of an application. These operations are performed by effectors which have to be implemented and installed in the execution environment. It has been found that modifying message routes between service instances is crucial for implementation of the proposed SOA application adaptation concept.

Rerouting messages to other instances is justified only when these instances share the same interface and provide identical functionality. This process could be performed effectively by installing suitable effectors in ESB. Current implementations of ESBs that are compliant with the JBI specification share some common attributes used during message processing. Each invocation of a complex service is described by its CID (Correlation ID) and is constant for one invocation, even when passed among services. A particular message sent between services is described by EID (Exchange ID). Generally speaking, the invocation of a complex service is described by CID and consists of several message exchanges described by EID which complies with the execution model proposed in Section 3. Extending NMR with effectors implementing a selected routing algorithm, capable of modifying message routing, would yield an adaptable ESB.

Priority	CID	Intentional Service Name	EID	Destination Service Endpoint
1	n/a	n/a	+	Service Endpoint 1
2	+	+	n/a	Service Endpoint 2
3	n/a	+	n/a	Service Endpoint 3
4	+	n/a	n/a	Service Endpoint 4
5	n/a	n/a	n/a	Service Endpoint 5
6	n/a	n/a	n/a	Service Endpoint 6

Table 1. Examples of routing rules

Once the message reaches the NMR, the routing algorithm checks conditions of routing rules in the routing table (see Table 1). If the message matches a routing rule, that rule is fired and the Service Endpoint from the routing rule substitutes the intended destination Service Name. Routing rules are split into groups with different priorities that are analyzed in a particular order. In every message, header parameters such as Correlation ID, intended Service Name and Exchange ID, are matched to routing rules. If the message matches several rules, one of them is selected on a *round-robin* basis to provide load balancing.

5. Case study

In order to verify the presented concept of SOA application composition and execution, the Enterprise Service Bus (ESB) as an integration platform has been extended by adaptability mechanisms. The prototype design follows the architecture of Adaptive ESB presented in the previous section. Adaptability mechanisms are implemented using Aspect Oriented Programming (AOP) (Kiczales et al., 1997), where new features can be enabled and disabled without modifying original ESB container source code.

The prototype implementation of adaptive ESB is exploited in a scenario which demonstrates how the proposed concept of quality control can be applied to a situation where clients should receive variable QoS depending on some constraints. The provided composite service execution model is used to generate a possible set of service instances that can be invoked during execution. Subset selection bases on a fitness function defined in Section 3.3, which determines user preferences by associating weights with the available composite service metrics. Possible solutions are then ordered according to the values of the calculated metrics and the adaptation policy selects a particular set of service instances based on high-level goals, i.e. the cost of execution and the overall client budget.

5.1 Composite service

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

Widely-available weather services do not provide weather information for GPS coordinates. Instead, they provide weather information for a given readable address; thus all possible input formats have to be converted to addresses and then polled for weather information.

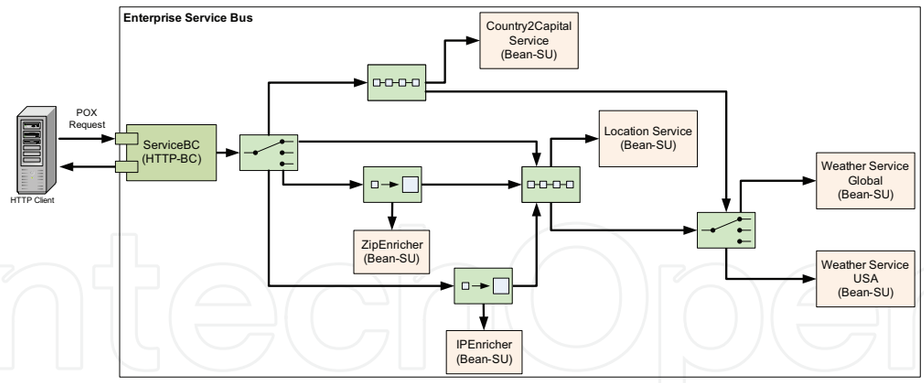

Fig. 7. EAI model of complex service used in the case study

To deploy this application into ESB, it has to be described in terms of EAI patterns. Invoking services in a sequence might be performed with the use of *RoutingSlip* integration patterns, while modifying the message content can be done using the *ContentEnricher* pattern. An EAI-based model of the presented composite service is depicted in Fig.7.

Let us assume that the client operates an Internet website that provides information for amateur pilots and that he wants to include very accurate weather forecasts for airports on the main website. The client is interested in a service which returns weather information for given geographical coordinates. The provider notices that this website receives 8,200 visits per week (on average) and, given the QoS selected by the client, it costs 54,000 credits. As the website become more popular, the number of invocations may significantly increase. The client therefore decides to buy a service with variable QoS and with the maximum budget of 55,000 credits. Detailed statistics of user invocations are presented in Fig. 8. Two different profiles can be distinguished. The first one reflects typical interest in the website, while the second one occurs when there is increased interest in the website. The increased numbers of page visits before the end of the week can be attributed to a national flying contest.

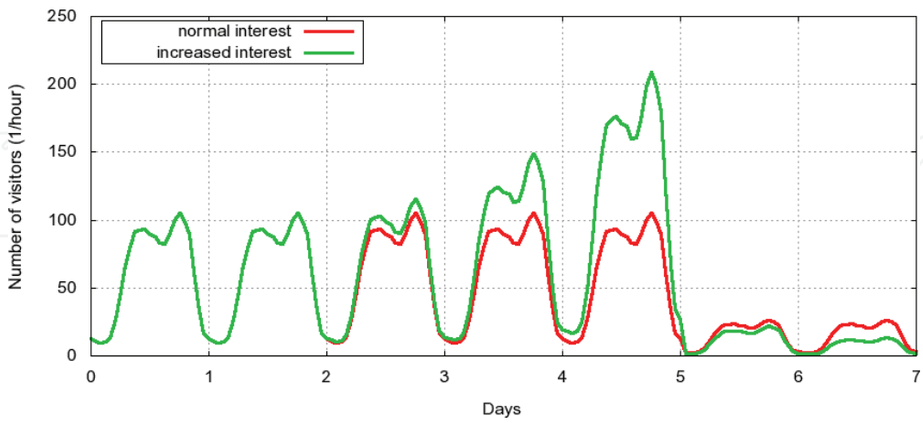

Fig. 8. Number of visitors per hour

5.2 Importance of metrics

For this type of service, it is better (from the customer's point of view) to achieve very accurate data slower than to rapidly access inaccurate information. The author therefore

decides to apply the fitness function presented in Section 3, which expresses the degree of overlap between the functionality of the composite service and the user's preferences. This means that each metric has to be normalized and its value has to be in the [0; 1] range where 1 means that this metric has the best possible value.

Three metrics are defined for this scenario: availability, execution time and data quality. Availability is defined as the percentage of successful invocations. Execution time metric is normalized linearly (for response time equal to 0*ms* the value of the metric is 1, while for response time above 1000*ms* the corresponding value is 0). The data quality metric equals 0 if data is accurate to within 100*km*, and 1 if it is accurate to within 10*km*.

Weights for each metric in the fitness function are calculated using the mentioned earlier AHP method. The customer determines the relative importance of each metric:

- *availability* is three times as important as *execution time*;
- *availability* is five times less important than *data quality*;
- *execution time* is five times less important than *data quality*.

Following calculations, the fitness function used in this case can be given as:

$$fitness = 0,20 * availability + 0,097 * execution\_time + 0,702 * data\_quality$$

5.3 Composite service execution model

In order to react to changes in the execution environment, a composite service execution model and an adaptation policy have to be defined. The model of the analysed service and its projection onto instances found in the ESB is depicted in Fig. 9. This model can be derived by simply transforming the EAI-based model shown in Fig. 7. The service is only a subset of a larger, complex service; hence the model contains only these services that are used in the invocations.

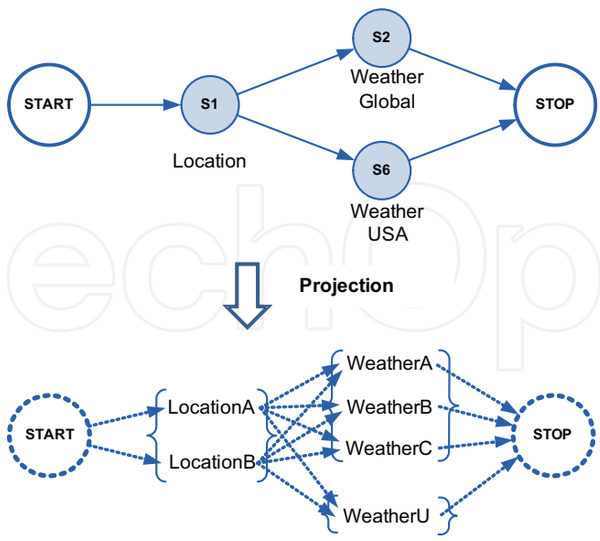

Fig. 9. Projection of the model of complex service execution at $Level_1$ onto composite services

There are six possible combinations of service instances that might be executed. The model, along with the expected values of metrics for projected composite services, is updated at runtime and used by the adaptation policy.

Composite service	Service *Location* instance	Service *Weather* instance	Cost [unit/100]	Availability [%]	Data quality [%]	Execution time [ms]	*fitness factor*
C_1	*LocationA*	*WeatherA*	650	0.81	0.95	200	0.737
C_2	*LocationA*	*WeatherB*	600	0.81	0.90	300	0.657
C_3	*LocationA*	*WeatherC*	350	0.72	0.70	500	0.535
C_4	*LocationB*	*WeatherA*	550	0.89	0.80	300	0.623
C_5	*LocationB*	*WeatherB*	500	0.89	0.80	400	0.614
C_6	*LocationB*	*WeatherC*	250	0.79	0.70	600	0.526

Table 2. Fitness factors of composite services for the variable QoS test case

Table 2 contains expected metric values for the composite services that might be invoked and the evaluated fitness factor for each composition.

5.4 Adaptation strategy

In the presented test case, a simple accounting system has been implemented which constantly monitors user invocations and updates the associated accounts. The same module calculates the expected number of invocations until the end of the accounting period. The number of invocations per week is calculated as a sliding window from the last 7 days. It is then used to calculate the expected number of invocations for the remaining days until the end of the accounting period. Given the number of credits left and the expected numbers of invocations until the end of accounting period, the module calculates the maximum price for subsequent invocations. This module is expressed as a fact and inserted into the working memory of the policy engine.

5.5 Results

Detailed statistics of user invocations are presented in Fig. 10. Increased numbers of page visits before the end of the week can be attributed to a national flying contest. The total number of invocations is not the expected 8,400 but approximately 10,200 as depicted in Fig. 10B. During the week, the system tries to estimate the total number of invocations during the accounting period – this fact is reflected in the figure. Because the actual number of invocations is greater than expected, system has to switch to a different set of instances to bring the total sum below the assumed 55,000 credits. Coloured areas in Fig. 10 show which composite service is used by the system to provide the service. One can notice that on the 4th day the system has decided to switch the composite service to the less expensive C_2 with fitness factor 0.657 instead of C_1 with 0.737 and then to C_4 with 0.623. As the expected number of invocations during the whole accounting period increases, the maximum number of credits that might be spent on each invocation keeps decreasing. This leads to further adjustments (on day 5) where the composite service is switched to C_5, C_3 and finally C_6. During the last 2 days, the number of invocations is noticeably smaller, so the system returns to its initial state i.e. to composite service C_1 with the best fitness value. Without an adaptation policy as depicted in Fig. 10A, the client would have to pay 64,000 credits in the described case; however the adaptation policy ensures that the cost is kept below the assumed limit of 55,000 credits. Execution time of provided service is depicted in Fig.10C, data quality in Fig. 10D and availability in Fig. 10E.

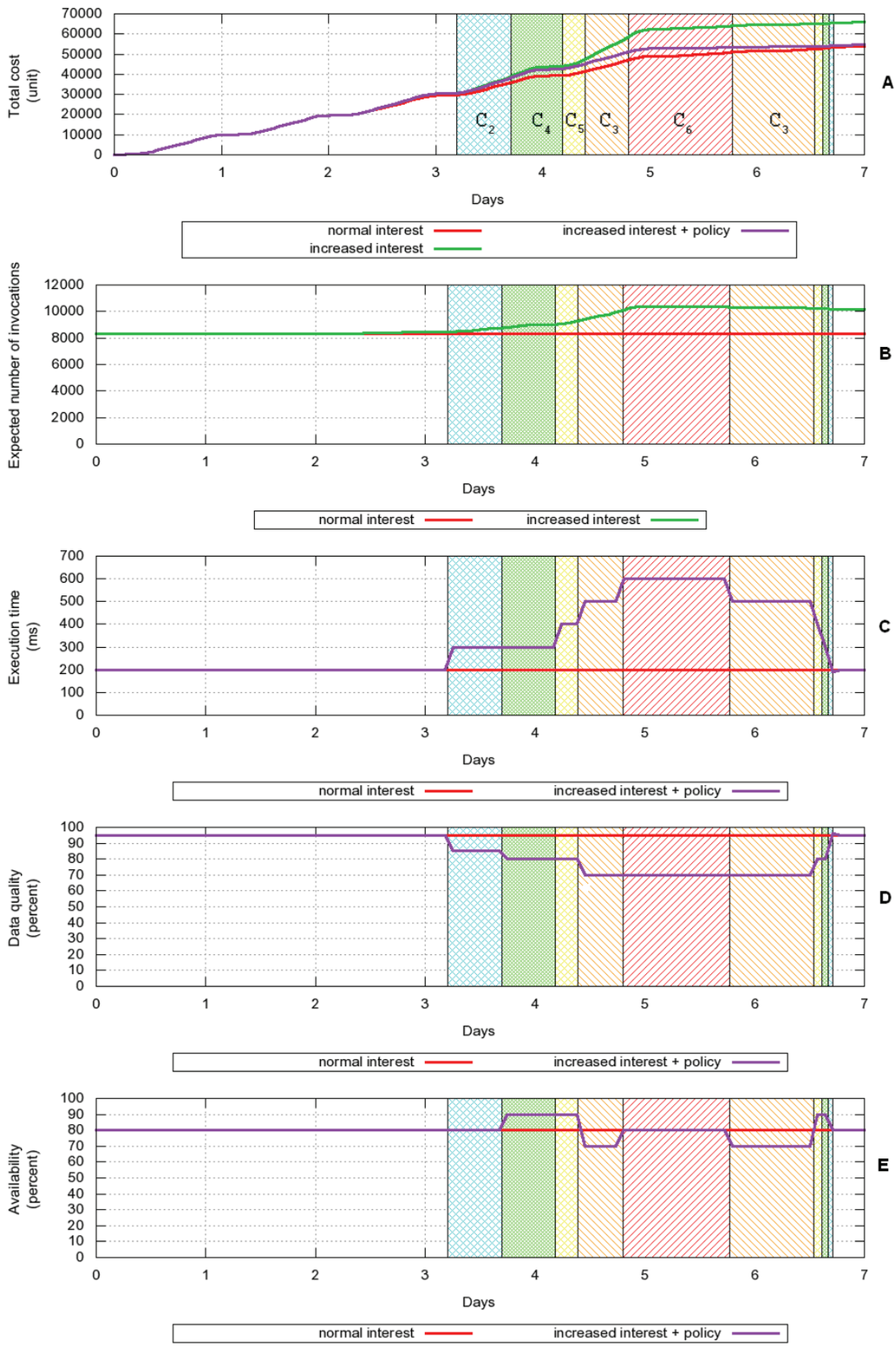

Fig. 10. Test case scenario with variable QoS

6. Summary

SOA applications which are loosely coupled and composed from self-contained, autonomous services with well-defined contracts are particularly well suited for adaptive control of QoS. Their execution environments (such as ESB) could be effectively equipped with mechanisms which allow implementation of MAPE paradigms.

A crucial element of the adaptation loop is a decision subsystem which enforces application adaptation actions, maintaining QoS/QoE guarantees under changing operational conditions and constrains. This rather complex process may be practically performed with SOA application execution model support. The presented study shows that a statistical model could be used for this purpose.

Statistical models, which can be identified at runtime, represent knowledge and experience derived from the SOA application execution history. This knowledge may be transparently used by the execution environment to control QoS according to a selected policy. The proposed approach leads to a new, important category of adaptive SOA systems which may satisfy consumer requirements under changing QoS conditions without involving a human operator.

7. References

Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, and Kishore Channabasavaiah. *S3: A Service-Oriented Reference Architecture*. IT Professional, 9(3):10–17, 2007.

Gordon Blair, Nelly Bencomo, and Robert B. France. Models@ run.time. Computer, 42:22–27, 2009.

G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in Web Service Composition and Execution. In *Web Services*, 2006. ICWS '06. International Conference on, pages 549–557, Sept. 2006.

Glen Dobson and Alfonso Sanchez-Macian. Towards Unified QoS/SLA Ontologies. In SCW '06: *Proceedings of the IEEE Services ComputingWorkshops* (SCW'06), pages 169–174, Washington, DC, USA, 2006. IEEE Computer Society

Thomas Erl. SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl). Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

Thomas Erl. Modern SOA Infrastructure: Technology, Design, and Governance, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2010

Ernest H. Forman and Mary Ann Selly. Decision By Objectives – How To Convince Others That You Are Right. World Scientific Publishing Co. Pte. Ltd., Singapore u.a., 2001.

Alan G. Ganek and Thomas A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

T. Gubala, D. Harezlak, M.T. Bubak, and M. Malawski. Constructing abstract workflows of applications with workflow composition tool. , *Cracow'06 Grid Workshop*, Vol 2: K-Wf Grid: the knowledgebased workflow system for Grid applications: October 15–18, 2006 Cracow, Poland

Paul Hershey and Donald Runyon. SOA Monitoring for Enterprise Computing Systems. In *EDOC '07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, page 443, Washington, DC, USA, 2007. IEEE Computer Society.

Paul Hershey, Donald Runyon, and Yangwei Wang. Metrics for End-to-End Monitoring and Management of Enterprise Systems. In *Military Communications Conference*, 2007. MILCOM 2007. IEEE, pages 1–7, Oct. 2007.

Gregor Hohpe and Bobby Woolf. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

IBM Corporation, An Architectural Blueprint for Autonomic Computing, June 2006

Kiczales, Gregor; John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin (1997). "Aspect-Oriented Programming". *Proceedings of the European Conference on Object-Oriented Programming*, vol.1241. pp. 220–242.

Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.

David Luckham. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional, May 2002.

Heiko Ludwig. Web services QoS: external SLAs and internal policies or: how do we deliver what we promise? In *Web Information Systems Engineering Workshops*, 2003. Proceedings. Fourth International Conference on, pages 115–120, 2003.

P. K. Mckinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

Daniel A. Menasce. Composing Web Services: A QoS View. *IEEE Internet Computing*, 8(6):88–90, 2004.

Aad Van Moorsel. Metrics for the Internet Age: Quality of Experience and Quality of Business. Technical report, *5th Performability Workshop*, 2001.

Brice Morin, Olivier Barais, Jean-Marc Jezequel, Franck Fleurey, and Arnor Solberg. Models@ run.time to support dynamic adaptation. *Computer*, 42:44–51, 2009.

Dmytro Rud, Andreas Schmietendorf, and Reiner Dumke. Resource metrics for service-oriented infrastructures. In Daniel L¨ubke, editor, *Proceedings of the Workshop on Software Engineering Methods for Service-oriented Architecture* 2007 (SEMSOA 2007), Hannover, Germany, pages 90–98. Leibniz Universit¨at Hannover, FG Software Engineering, May 2007.

D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. Wiley, 2000.

Tomasz Szydlo and Krzysztof Zielinski. Method of Adaptive Quality Control in Service Oriented Architectures. In *ICCS '08: Proceedings of the 8th international conference on Computational Science*, Part I, pages 307–316, Berlin, Heidelberg, 2008. Springer-Verlag.

Tomasz Szydlo, QoS driven Semantics Based SOA Applications Composition and Execution, Ph.D. dissertation, AGH-UST Krakow, October 2010.

Maja Vuković, Context Aware Service Composition. Technical report, University of Cambridge, 2006.

Applications and Experiences of Quality Control
Edited by Prof. Ognyan Ivanov

ISBN 978-953-307-236-4
Hard cover, 704 pages
Publisher InTech
Published online 26, April, 2011
Published in print edition April, 2011

The rich palette of topics set out in this book provides a sufficiently broad overview of the developments in the field of quality control. By providing detailed information on various aspects of quality control, this book can serve as a basis for starting interdisciplinary cooperation, which has increasingly become an integral part of scientific and applied research.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Tomasz Szydło and Krzysztof Zieliński (2011). Model Driven Adaptive Quality Control in Service Oriented Architectures, Applications and Experiences of Quality Control, Prof. Ognyan Ivanov (Ed.), ISBN: 978-953-307-236-4, InTech, Available from: http://www.intechopen.com/books/applications-and-experiences-of-quality-control/model-driven-adaptive-quality-control-in-service-oriented-architectures

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen