ECMAF: An Event-Based Cross-Layer Service Monitoring and Adaptation Framework

Chrysostomos Zeginis, Konstantina Konsolaki, Kyriakos Kritikos, and Dimitris Plexousakis

Department of Computer Science, University of Crete, Greece
and Information Systems Laboratory, ICS-FORTH, Greece
{zegchris,konsolak,kritikos,dp}@ics.forth.gr

Abstract. Although several techniques have been proposed towards monitoring and adaptation of Service-Based Applications (SBAs), few of them deal with cross-layer issues. This paper proposes a framework, able to monitor and adapt SBAs across all functional layers. This is achieved by using techniques, such as event monitoring and logging, event-pattern detection, and mapping between event patterns and appropriate adaptation strategies. In addition, a taxonomy of adaptation-related events and a meta-model describing the dependencies among the SBA layers are introduced in order to "capture" the cross-layer dimension of the framework. Finally, a specific case study is used to illustrate its functionality.

Keywords: event, monitoring, adaptation, cross-layer, service, pattern, Event-Calculus, non-functional.

1 Introduction

Web services are an emerging technology attracting a lot of attention from both academia and industry in recent years. Thus, more and more businesses adopt them to facilitate and automate their business processes. However, once services and business processes become operational, several emerging issues must be considered throughout the life-cycle of a Service-Based Application (SBA), such as the ones concerning service monitoring and adaptation. These two processes are tightly connected and result in improving and customizing non-performing services, so as to adapt to the context changes and satisfy new requirements.

As far as monitoring is concerned, SBAs need to be managed and monitored, so that stakeholders have a clear view of how services perform within their operational environment, take management decisions, and perform control actions to modify and adjust their behavior. In [5], Web service monitoring is defined as the process of collecting and reporting relevant information about Web service execution and evolution. Many monitoring approaches have been proposed, most of them focusing on measuring QoS metrics, gathered on the basis of SLAs, given that an SLA dictates what are the service obligations in terms of performance. The measured metrics are then compared with the corresponding SLA bounds to detect possible violations.

Furthermore, the dynamic and ever-changing nature of the business and physical environment requires Web services to be highly reactive and adaptive to the changes and variations they are subjected to. They should be equipped with mechanisms to ensure that they can adapt to meet changing requirements. Such changes are identified, detected, and foreseen in the running SBA during the monitoring process. In [10] Web service adaptation is defined as a process of modifying SBAs so as to satisfy new requirements dictated by environmental changes on the basis of adaptation strategies designed by the system integrator.

However, it is critical that monitoring and adaptation occur across all the SBA functional layers, as they are adopted by many projects, such as the S-cube (`http://www.s-cube-network.eu`) and SLA@SOI (`http://sla-at-soi.eu`); namely the Business Process Management layer (BPM), the Service Composition and Coordination layer (SCC) and the Service Infrastructure layer (SI). These layers are closely related and many dependencies exist among them. A thorough review of current monitoring and adaptation techniques [25] reveals that these techniques are very fragmented. Although current approaches cover a wide spectrum of monitoring and adaptation [19,2,3], few of them deal with cross-layer issues. The latter ones either do not provide a concrete solution to the problem [12], or do not take into consideration all SBA layers [9,22], or do not elaborate on monitoring issues [26].

In this paper, a cross-layer monitoring and adaptation framework is outlined that is based on monitored events. In addition, a layer-based taxonomy of adaptation-related events and a meta-model describing the dependencies among components of a cross-layer system are introduced to pinpoint the need for such a type of framework. The main strengths of this approach are the ability to handle both functional and non-functional service properties and the use of a reasoning tool deriving appropriate adaptation strategies by exploiting a set of rules. Furthermore, it has also proactive adaptation capabilities using pattern detection techniques and can be easily distributed into many computer nodes. In Section 2 a comparison table clearly states the advantages of this work.

The rest of the paper is structured as follows. Section 2 summarizes the related work. Section 3 introduces the proposed taxonomy of adaptation-related events. Section 4 presents the dependency model. Section 5 presents the proposed framework, while Section 6 exemplifies how this framework supports the case study. Finally, some concluding remarks and future work directions are presented in Section 7.

2 Related Work

The need for monitoring different functional and non-functional requirements, as well as taking adaptation actions is widely recognized by industry and academia, as a means of improving SBAs. There are several works that propose monitoring and adaptation architectures but most of them only focus on the SCC layer.

Baresi et al. [4] present an approach for self-healing of BPEL processes. This approach is based on Dynamo [3] monitoring framework along with an AOP extension of ActiveBPEL and a monitoring and recovery subsystem using Drools

ECA rules. A composition designer provides assertions for invoking, receiving or picking activities in the business process. These assertions can be specified using two domain specific languages (WSCoL and WSReL). The problem of selecting alternative services and dealing with possible interface mismatches when forwarding a request to an alternative endpoint recovery is not explicitly addressed. Additionally, the recovery rules cannot be changed dynamically, as they need to be compiled off line.

The VieDAME environment [19] extends the ActiveBPEL engine to enable BPEL process monitoring and partner service substitution based on various strategies. The services are selected according to defined selectors. VieDAME requires service registration in a repository and marking services to be monitored and eventually substituted as replaceable. It uses an engine adapter to extend the engine's functionality, but does not explicitly address fault handling.

Barbon et al. [2] present an event-based monitoring approach, developed within the Astro project, which also extends the ActiveBPEL engine and defines RTML, an executable monitoring language to specify SBA properties. Events are combined by exploiting past-time temporal logics and statistical functions. Monitors run in parallel with the BPEL process as independent software modules verifying the guarantee terms by intercepting the input or output messages received or sent by the process. This work does not allow for dynamic (re-)configuration of the monitoring system in terms of rules and meta-level parameters.

In [24,17] the authors present an approach towards extending WS-Agreement [1]. This approach supports monitoring of functional and non-functional properties. EC-Assertion is introduced to specify service guarantees in terms of different types of events, which is defined by a separate XML schema. It is based on Event Calculus (EC) [23]. By proceeding in parallel with the business process execution, it leads to less impact on performance but also to a smaller degree of responsiveness in discovering erroneous situations.

Farrel et al. [8] present an SLA-based monitoring approach exploiting EC as the underlying formalism. This approach addresses the utility computing domain, where the cornerstone aspect is to provide resources with certain quality characteristics. Contracts are defined based on contract patterns, which are then axiomatized using EC. Then, the effects of critical events on the contract state/evolution are defined. The respective framework is based on a particular architecture and comprises an analyzer managing the contract analysis and reporting, and a visualizer representing the SLA monitoring results.

A platform for developing, deploying, and executing SBAs is proposed in [6], incorporating tools and facilities for checking, monitoring, and enforcing service requirements expressed in WS-Policy notations. The Colombo platform comes with a module that manages policy assertions. Apart from evaluating the assertions attached to particular service-related entities at both design and runtime, the framework provides the means for policy enforcement, e.g., it may approve a message's delivery of a message, reject it, or defer further processing.

Despite the previous layer-specific approaches, some approaches towards cross-layer service monitoring and adaptation have been proposed. Gjørven et al. [9]

propose a coarse-grained approach, which exploits mechanisms across two layers (Service Interface and Application corresponding to ours SCC and SI layers) in a coordinated fashion. Kazhamiakin et al. [12] define appropriate mechanisms and techniques to address the adaptation requirements and constitute an integrated cross-layer framework. Both approaches tackle the problem in an inflexible manner, as the adaptation logic is predefined and static. Popescu et al. [22] provide support for dynamic cross-layer adaptation using adaptation templates, composed either directly, through invocations of WSDL operations or indirectly, through events. This approach, though, does not consider the Infrastructure layer. Finally, Zengin et al. [26] introduce an adaptation manager (CLAM) that can deal with cross- and multi-layer adaptation problems. This approach ranks a set of adaptation paths, after constructing and adaptation tree starting form an initial adaptation trigger at any of the three SBA layers.

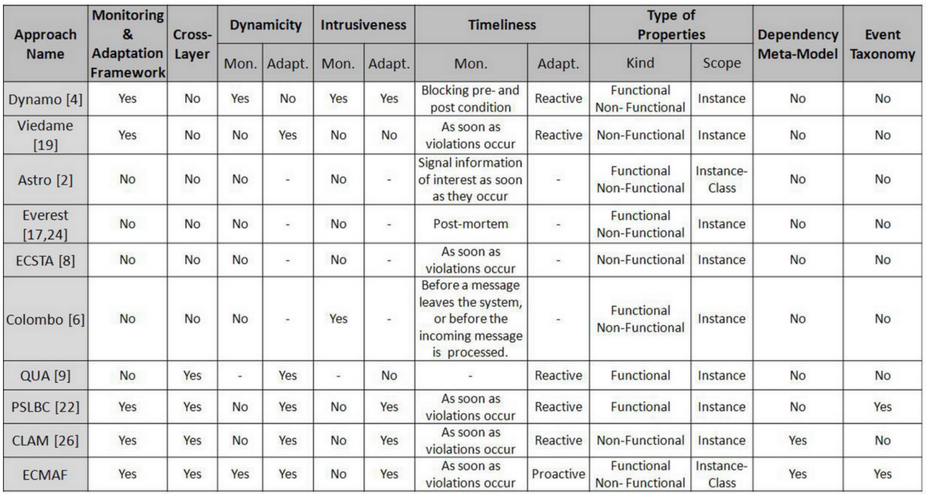
A summary of all the aforementioned approaches is presented in Table 1. The approaches are compared according to their (cross-layer) monitoring and adaptation capabilities, dynamicity, intrusiveness and timeliness. These properties acquire a different meaning towards monitoring and adaptation. The dynamicity of a monitoring framework concerns the ability of the framework to change monitoring properties during the execution of the process whereas the dynamicity of an adaptation framework allows additions and deletions of adaptation rules. Moreover approaches which perform monitoring by weaving code that implements the required checks inside the code of the system that is being monitored are concerned as intrusive approaches. Regarding adaptation approaches, a framework is intrusive whether the applied adaptation actions change the process. We assume that intrusiveness is not desirable for monitoring unlike adaptation. The timeliness of monitoring system presents the ability of the framework to signal violations of the monitoring properties the time they occur and not after the termination of the instance. On the other hand timeliness of an adaptation framework is about the proactive or the reactive execution of the adaptation actions. Furthermore the kind and the scope of the monitored information is provided. The former one refer to the functional and non functional properties of a SBA while the latter one to instance or class application of the approach. Finally, the last two properties refer to the availability of a dependency meta-model describing the dependencies among the SBA layers and a taxonomy of monitored events.

As shown in Table 1 few of current monitoring and adaptation approaches deal with cross-layer issues [9,22,26]. These works focus mainly on adaptation process and lack some important properties that ECMAF efficiently address, such as proactive adaptation, functional and non-functional aspects consideration and wide scope, enriched with a dependency meta-model and an event taxonomy.

3 Taxonomy of Adaptation-Related Monitored Events

Many of the proposed monitoring approaches can detect different event types. These events deliver information about the SBA evolution and context change.

Table 1. Comparison Table



Approach Name	Monitoring & Adaptation Framework	Cross-Layer	Dynamicity		Intrusiveness		Timeliness		Type of Properties		Dependency Meta-Model	Event Taxonomy
			Mon.	Adapt.	Mon.	Adapt.	Mon.	Adapt.	Kind	Scope		
Dynamo [4]	Yes	No	Yes	No	Yes	Yes	Blocking pre- and post condition	Reactive	Functional Non- Functional	Instance	No	No
Viedame [19]	Yes	No	No	Yes	No	No	As soon as violations occur	Reactive	Non-Functional	Instance	No	No
Astro [2]	No	No	No	-	No	-	Signal information of interest as soon as they occur	-	Functional Non-Functional	Instance-Class	No	No
Everest [17,24]	No	No	No	-	No	-	Post-mortem	-	Functional Non-Functional	Instance	No	No
ECSTA [8]	No	No	No	-	No	-	As soon as violations occur	-	Non-Functional	Instance	No	No
Colombo [6]	No	No	No	-	Yes	-	Before a message leaves the system, or before the incoming message is processed.	-	Functional Non-Functional	Instance	No	No
QUA [9]	No	Yes	-	Yes	-	No	-	Reactive	Functional	Instance	No	No
PSLBC [22]	Yes	Yes	No	Yes	No	Yes	As soon as violations occur	Reactive	Functional	Instance	No	Yes
CLAM [26]	Yes	Yes	No	Yes	No	Yes	As soon as violations occur	Reactive	Non-Functional	Instance	Yes	No
ECMAF	Yes	Yes	Yes	Yes	No	Yes	As soon as violations occur	Proactive	Functional Non- Functional	Instance-Class	Yes	Yes

They are used to indicate whether the SBA execution evolves normally and whether there are some deviations or even violations of the desired or expected functionality. Most events are recurring during service executions and usually with the same order. Thus, it is desirable to introduce a taxonomy of monitored events to enable the mapping between these events and the suitable adaptation strategies as well as the event derivation applied in the proposed framework.

The taxonomies of common monitored events proposed are either generic or domain-specific (e.g. real-time SBAs). [22] introduces an event taxonomy for three possible application layers: organization, behavior and service, to semi-automate the discovery and selection of adaptation templates needed to fulfil complex adaptation requirements. [14] categorizes monitored events into Interface-level mismatches, i.e., services with similar functionality but through different WSDL interfaces, and Protocol-level mismatches, i.e., mismatches concerning the order or number of supplied and required messages.

The proposed high-level event taxonomy is based on two different criteria: a) the affected SBA layer and b) the service aspect concerned (functional, non-functional). The affected layer concerns the three functional layers analyzed in Section 1, while the service aspect concerns the service functional and non-functional characteristics [20]. The former ones detail the operational aspects that define the overall service behavior, such as the way and time it is invoked, while the latter concern quality attributes (e.g., response time and throughput). Its main advantage upon the other taxonomies is that it fits perfectly to the adopted SBA layers as well as the consideration of both functional and non-functional service aspects.

Fig. 1 illustrates the proposed taxonomy of adaptation-related events for the three functional SBA layers. Indicatively, at the BPM layer there are classified mismatches regarding the business process, such as KPI violations or monitored events stemming from modifications at this layer (e.g., business goal or process model modifications). At the SCC layer, monitored events focus on mismatches about service execution and QoS violations, such as I/O failures and SLA violations. Finally, at the SI layer, events mainly concern device failures affecting the overall SBA, such as limited resources or network failures.

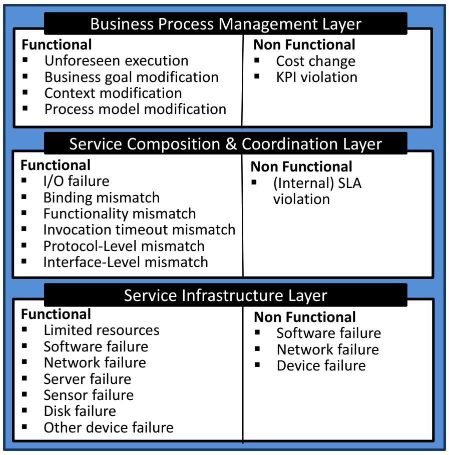


Fig. 1. Taxonomy of adaptation-related monitored events

4 Dependency Meta-model

When faults occur, it is imperative to be able to detect the root of the problem by following a process, usually called root-cause analysis. Such a process benefits from the use of a specific model called *dependency model*. This model should describe both the functional and non-functional dependencies between the system components across all possible layers to enable the detection of the component causing the fault through a top-down traversal of the respective dependencies, starting from the component where the problem is detected. Dependency models should also allow a bottom-up traversal of the respective dependencies. Such a traversal is essential so as to enable the derivation of events at higher-layers with respect to the events occuring at lower levels. Moreover, dependency models should be able to describe both static as well as dynamic component dependencies. The former are usually known and established at design time, while the latter are detected and created at runtime. Dependencies may also change during the lifetime of a SBA or system. For example, the memory requirements for installing a service may be different from those related to the service execution.

Dependency models must also conform to a particular structure and must be described in a formal way so as to enable reasoning on them. To this end, a novel dependency meta-model is proposed in this paper, which has been specified through the OWL ontology and is depicted in Fig. 2. This meta-model has been carefully designed to capture all the relevant aspects of component dependencies.

The central concept in this meta-model is *Dependency*, which is characterized by the following four main properties [13]: a) *strength*: how strong (optional or mandatory) is the dependency between two or more components, b) *formalization*: what is the dependency's degree of formalization which directly relates to the automation degree with respect to the dependency's capturing, c) *criticality*: how critical is to capture this dependency, d) and *period*: what is the time period for which this dependency holds. Dependencies can be either *Functional* or *NonFunctional*. Functional dependencies exist between functional components, while non-functional dependencies typically exist between non-functional components. Both the *FunctionalComponent* and *NonFunctionalComponent* concept are sub-concepts of *Component*. Functional components are characterized by two main properties: a) *type*: indicating if the component is a hardware, software, or logical entity (e.g. service, activity), and b) *activity*: if the component can be directly queried or instrumented or requires the existence of an intermediary for obtaining the component's information. On the other hand, a non-functional component can be either a *QoSAttribute* or *QoSMetric*.

Non-functional dependencies can be *Qualitative* or *Quantitative*. Both dependency types can be expressed with the OWL-Q [16] semantic, non-functional based service description language, which has been slightly extended to enable the description of process and infrastructural quality attributes and metrics. Fig. 2 shows with black color which novel non-functional concepts have been introduced and with grey color which were the original OWL-Q concepts. Qualitative dependencies between two non-functional components describe particular, general non-functional relationships which can be either only qualitatively assessed, or also quantitatively assessed through e.g. instrumentation but their quantitative dependency model holds only for particular systems and services. Such a type of dependencies is characterized by two main properties: a) *valueDirection*: indicating that the value of the first non-functional component changes in the same or opposite way with respect to the value of the second one; b) *valueImpact*: describes the impact that the first non-functional component's value has on the second non-functional component's value.

The OWL-Q extensions introduced allow the description of quantitative dependencies across the same or different layers. In particular, three different metric types have been introduced: *ProcessMetric*s, *ServiceMetric*s, and *InfrMetric*s, where each metric type not only corresponds to one of the respective layers considered but also measures a particular *QoSAttribute* and applies to particular functional components at this layer. Similarly, a *QoSAttribute* can be either a *ProcessAttribute*, a *ServiceAttribute*, or an *InfrAttribute*. Each metric can then be measured through the application of a *Function* on other metrics at the same or lower layers. For example, a *ProcessMetric* can be measured through process, service, and infrastructural metrics and concerns a particular process component, such as the process itself or one of its activities. It must be noted that OWL-Q

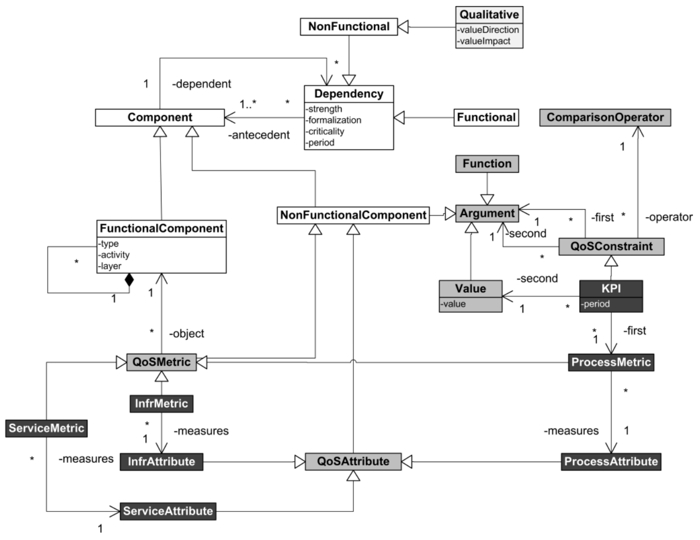


Fig. 2. The dependency meta-model

is already capable of describing the way both single and composite metrics can be computed from measurement directives and other metrics, respectively.

Therefore, OWL-Q allows for the description of both quantitative and qualitative cross-layer metric models. Quantitative models can be exploited not only for monitoring but also for conformance checking as they do not only allow the calculation of the values of the metrics at the higher levels from the values of metrics at the same or lower levels, but also the comparison of the computed values against the stated requirements. On the other hand, qualitative models allow inspecting the correctness of the monitored values, as e.g. particular qualitative dependencies may not be respected by the monitored values of specific quality components, produced by particular error-prone sources of information.

5 Monitoring and Adaptation Framework

Fig. 3 presents the architecture of the proposed cross-layer monitoring and adaptation framework. This framework comprises a Monitoring Engine able to collect the monitored events during the service execution, an Adaptation Engine able to perform adaptation actions, and an Execution Engine. The first two engines communicate with each other via events through a publish/subscribe mechanism.

Monitoring Engine. The Monitoring Engine comprises a **Monitor Manager** and a number of individual **Monitoring Components**. Each of the latter components is assigned to detect events at a specific SBA layer and immediately deliver them to the Monitor Manager. The Monitor Manager, in turn, continuously delivers information about the service execution produced by the Execution Engine while collecting events from the Monitoring Components. The Monitor

Manager communicates with the Translator via a publish/subscribe mechanism. A needed monitored event is delivered to the Translator as soon as it is detected. It is imperative to send the events in the order that they are received so as to have an as reliable as possible pattern matching mechanism. As there are many Monitoring Components delivering events, specific techniques are required to ensure this, such as event timing [18] and clock synchronization [15].

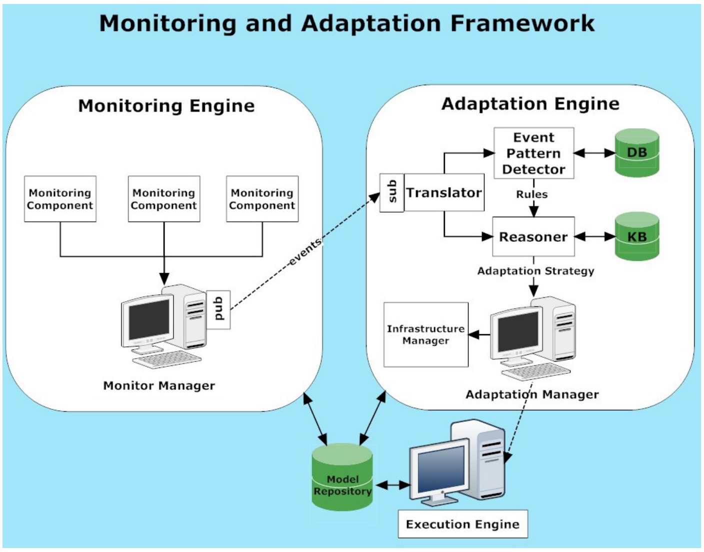


Fig. 3. Architecture of Monitoring and Adaptation framework

Adaptation Engine. The Adaptation Engine comprises a number of components supported by suitable repositories:

- The **Translator**, also incorporating a *subscribe mechanism*, receives the events sent by the Monitor Manager and translates them into a suitable format required by the Event Pattern Detector and Reasoner components.
- It is very common that failures at the SI layer lead to other failures and violations both at the same and higher layers, forming a chain of monitored events. So, it is desirable to detect those event patterns causing service failures by exploiting specific mechanisms (e.g. pattern matching ones [11]). These patterns are detected by the **Event Pattern Detector**.
  Our approach relies on using such a technique to detect dynamically patterns of monitored events to prevent other chains of events from occurring. As such, it enables the *proactive service adaptation* by mapping the detected patterns to suitable adaptation strategies. For example, having received two monitored events the Event Pattern Detector detects a specific pattern composed of one more event. This pattern is mapped to an adaptation strategy, and then, this mapping is translated into a rule that is passed to the Reasoner.

The adaptation strategy which is finally derived from the Reasoner, into an appropriate format, prevents the third event from occurring. A **Database** supplies the needed information for the mapping and the translation.
– The **Reasoner** receives events from the Translator and rules from the Event Pattern Detector, and derives an adaptation strategy to be performed by the **Adaptation Manager**. The rules provide the required data to perform the derivation, supported by a Knowledge Base (KB), containing the appropriate information to take the right decision.
– The **Adaptation Manager** executes the adaptation strategy exported by the Reasoner with the aid of two components. The **Infrastructure Manager** is able to treat malfunctions regarding the SI layer, which is the main source of many service failures, and the **Model Repository** supplies the appropriate information, such as service descriptions and requirements, layer dependencies, and metric and SLA models, so as to fulfil the supported adaptation strategies. The Execution Engine is called to support the adaptation process, especially for strategies regarding the BPM and SCC layers.

The main benefits of the monitoring and adaptation framework are as follows:

– **Distributed workload**. As there are layer-specific Monitoring Components that pass monitored events to the Monitor Manager, monitoring is distributed among the available monitoring mechanisms. In addition, there can be many computer nodes with a separate Monitoring and Infrastructure Manager component, that can a handle a portion of the whole monitoring and adaptation workload, as depicted in Fig. 4.
– **Extensibility**. As services evolve, new monitoring and adaptation techniques are required in order to cope with continuous context changes and other unpredictable malfunctions. This framework can integrate such techniques with the existing ones, while preserving its functionality and integrity.
– **Cross-layer capability**. The framework is able to support all three SBA functional layers. It incorporates mechanisms to detect events across all layers and derive additional events using pattern matching techniques.
– **Pro-active adaptation**. The use of pattern matching techniques, as well as the mapping between patterns and adaptation strategies allows for proactively adapting the SBA.

Some preliminary implementation solutions have already been investigated. As far as the monitoring is concerned we plan to use the Astro project [2], which has already been discussed in Section 2. In addition, we have decided to use the ESPER language (`http://esper.codehaus.org`) for the specification of the events and the pattern detection process. ESPER provide efficient event processing, comprehensive pattern detection and publish/subscribe capabilities. Finally, the reasoning process can be efficiently accomplished by the powerful EC-Jess Reasoner [21], which translates Event Calculus axioms into Jess rules and then encodes the domain description as a Jess program. Its main benefits are that it handles Event Calculus rules and produces models as output. The joining of these mechanisms as well as other gaps filling within this framework is in our future plans.

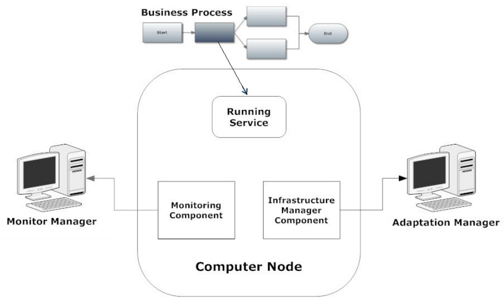


Fig. 4. Distributed workload performed by a computer node

6 Traffic Management Case Study

This case study describes a traffic management system designed to manage normal traffic situations as well as emergency cases [7]. Such emergency case handling includes several different actions, such as directing the rescue forces to the accident location and managing traffic deviations. Fig. 6 and Fig. 5 respectively illustrate these two cases. Each figure depicts the three functional layers. In both cases, workflow tasks are executed either manually or by mapping them on (Web) services. Each service is then mapped to the appropriate infrastructure.

The actors involved are *traffic managers*, i.e., individuals accountable for entities controlling the traffic management system, generic *rescue forces* (e.g., police and ambulances), and *citizens*, such as motorists and pedestrians.

Fig. 6 illustrates normal traffic conditions, where the system tries to optimize some parameters such as total noise, overall throughput, and air pollution. In particular, the system shall consider different needs, such as the ones of pedestrians and motorists, and other factors like heavy traffic, public events, school and working hours, holidays or public regulations which may alter traffic demand and needs during conditions that do not involve emergencies. The system interrupts the Normal Traffic Situation process, when an accident happens, and jumps to the Critical Traffic Situation subprocess.

Fig. 5 depicts a critical traffic situation, in which a serious car accident occurs at a central road. In particular, the involved citizens inform the traffic manager that must control the overall traffic situation (control traffic devices, inform citizens) and assess the incident so as to inform the appropriate rescue forces about the accident and direct them to the specific location. Moreover, the traffic manager monitors the environment variables, such as air pollution and noise. Different adaptation actions must be taken by the traffic manager as well as by the rescue forces, such as:

– Traffic management device reconfiguration (e.g., traffic lights) by the traffic manager, in order to reduce stop-and-go traffic. This should also help to keep air pollution low, even if it is not critical during emergency situations.
– Accident reporting to citizens via their devices (e.g, GPS, mobile phones) by the traffic manager to avoid traffic congestion at the accident location.

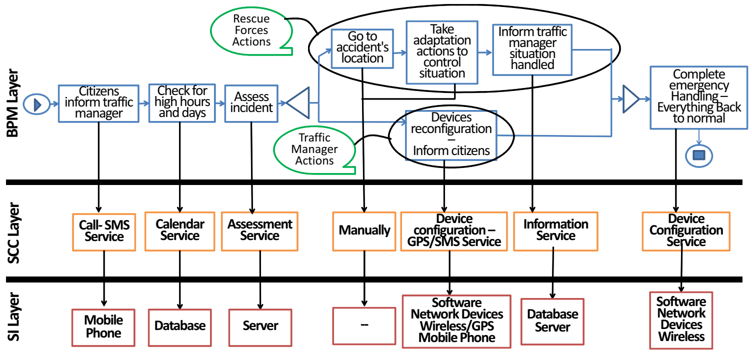

BPM Layer
SCC Layer
SI Layer
Rescue Forces Actions
Go to accident's location
Take adaptation actions to control situation
Inform traffic manager situation handled
Citizens inform traffic manager
Check for high hours and days
Assess incident
Devices reconfiguration – Inform citizens
Traffic Manager Actions
Complete emergency Handling – Everything Back to normal
Call- SMS Service
Calendar Service
Assessment Service
Manually
Device configuration – GPS/SMS Service
Information Service
Device Configuration Service
Mobile Phone
Database
Server
–
Software Network Devices Wireless/GPS Mobile Phone
Database Server
Software Network Devices Wireless

Fig. 5. Critical traffic situation

– Traffic closing/limiting to or from the involved location by the rescue forces.
– Traffic deviation by the rescue forces through alternative places not intended for heavy traffic.

After a complete emergency handling, there is a gradual return back to the normal situation. The rescue forces inform the traffic manager, who updates the system and informs the citizens through their devices.

As already discussed, there are various dependencies among the three SBA layers. The occurrence of a failure at one layer may result in a failure at other layers. This work aims at locating the failure event and taking adaptation actions in order to prevent its spread at the others layers, as soon as possible.

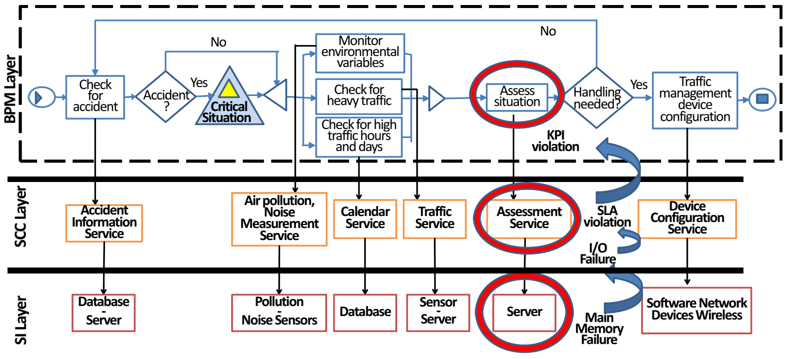


Fig. 6. Layers' interaction during normal traffic conditions

Fig. 6 presents an illustrative example. Suppose that a KPI exists dictating that the maximum duration of the process should be less than 10 seconds. Further, suppose that an SLA exists for the assessment service *AS* dictating that its maximum execution time must be less than 6 seconds. Thus, as it can be seen, a violation of the respective SLA constraint may cause a violation to the KPI, by considering that the previous process activities do not run longer than 3 seconds. It must also be indicated that *AS*'s execution time is inverse proportional to the main memory size and the CPU percentage allocated for its execution. Moreover, there is a low limit for the main memory allocated, after which the SLA violation will be unavoidable as the service behavior will be unpredictable and even if it does not fail it will certainly take a long time to execute. In fact, after 2 seconds from the *AS*'s execution, the main memory allocated to it has indeed surpassed the low level of 50 MB.

A Monitoring Component, running at the server where *AS* is executed, detects that the available main memory is not sufficient (SI layer) for *AS*. At the same time, another Monitoring Component detects that there is an I/O failure at the SCC layer as *AS* has produced a wrong output. Both events are first sent to the Translator, which transforms them to the appropriate format and sends them to the Reasoner. Based on the two events received, a specific rule is fired which derives that the best strategy is to execute another instance of the *AS* service at a more powerful server and with a better memory and CPU allocation. The suitability of the strategy lies on the fact that by executing a "better" service instance and with better allocation for the hardware resources, the probability that the SLA is not violated becomes very high (as we do not know if another failure may occur in the near future regarding the new instance) and in this way also the KPI violation may be avoided. Such a rule has been derived by the Event Pattern Detector based on the previous history log and has been already inserted into the Reasoner. The derived strategy is sent to the Adaptation Manager which executes it with the assistance of the Infrastructure Manager and the Execution Engine.

As can be understood, ECMAF handles perfectly such a cross-layer scenario. The adaptation actions performed are the appropriate ones, based on the event history and the current context. Moreover, the dependencies among the layers are clearly discerned.

7 Conclusions and Future Work

To sum up, this paper describes a framework that is able to detect monitored events across the three functional layers of a SBA and derive suitable adaptation strategies through a reasoning mechanism. The communication between the monitoring and adaptation engines is achieved by a publish/subscribe mechanism. In addition to the framework's description, a taxonomy of adaptation-related events and a dependency meta-model between the system components across all functional SBA layers are provided, in order to pinpoint the need for such a cross-layer approach. The main benefits of this approach are that it can handle both functional and non-functional service aspects, as well as it can proactively adapt the SBA across all the functional layers.

The following future directions are planned. First, developing such a distributed cross-layer monitoring and adaptation framework, using existing technologies and mechanisms and extending them if necessary. Second, extending the current taxonomy of adaptation-related events, exploiting additional aspects. Third, extending the proposed dependency meta-model. Finally, experimentally evaluating and optimizing the framework.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

1. Andrieux, A., et al.: Web Services Agreement Specification (March 2007), `http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.graap-wg/docman.root.published_documents.web_services_agreement_specifica/doc14574`
2. Barbon, F., Traverso, P., Pistore, M., Trainotti, M.: Run-time Monitoring of Instances and Classes of Web Service Compositions. In: ICWS, pp. 63–71. IEEE (2006)
3. Baresi, L., Guinea, S.: Dynamo: Dynamic Monitoring of WS-BPEL Processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 478–483. Springer, Heidelberg (2005)
4. Baresi, L., Guinea, S., Pasquale, L.: Self-healing BPEL Processes with Dynamo and the JBoss Rule Engine. In: ESSPE 2007 in conjunction with the 6th ESEC/FSE joint meeting, pp. 11–20. ACM (2007)
5. Benbernou, S., Cavallaro, L., Hacid, M.S., Kazhamiakin, R., Kecskemeti, G., Pazat, J.L., Silvestri, F., Uhlig, M., Wetzstein, B.: PO-JRA-1.2.1, State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs. Tech. rep., S-cube (July 2008)
6. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Colombo: Lightweight middleware for service-oriented computing. IBM Systems Journal 44(4), 799–820 (2005)
7. Di Nitto, E., Mazza, V., Mocci, A.: Collection of industrial best practices, scenarios and business cases (2009)
8. Farrell, A., Sergot, M., Salle, M., Bartolini, C.: Using the Event Calculus for the Performance Monitoring of Service-Level Agreements for Utility Computing. In: WEC, vol. 6. Citeseer (2004)
9. Gjørven, E., Rouvoy, R., Eliassen, F.: Cross-layer self-adaptation of service-oriented architectures. In: MW4SOC, pp. 37–42. ACM (2008)
10. Hielscher, J., Kazhamiakin, R., Metzger, A., Pistore, M.: A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing. In: Mähönen, P., Pohl, K., Priol, T. (eds.) ServiceWave 2008. LNCS, vol. 5377, pp. 122–133. Springer, Heidelberg (2008)
11. Karp, R.M., Rabin, M.: Efficient randomized pattern-matching algorithms. IBM Journal Research and Development 31(2), 249–260 (1987)
12. Kazhamiakin, R., Pistore, M., Zengin, A.: Cross-Layer Adaptation and Monitoring of Service-Based Applications. In: Dan, A., Gittler, F., Toumani, F. (eds.) ICSOC/ServiceWave 2009. LNCS, vol. 6275, pp. 325–334. Springer, Heidelberg (2010)
13. Keller, A., Blumenthal, U., Kar, G.: Classification and Computation of Dependencies for Distributed Management. In: ISCC. IEEE, Antibes (2000)

14. Kongdenfha, W., Motahari-Nezhad, H.R., Benatallah, B., Casati, F., Saint-Paul, R.: Mismatch Patterns and Adaptation Aspects: A Foundation for Rapid Development of Web Service Adapters. IEEE Trans. Serv. Comput. 2, 94–107 (2009)
15. Kopetz, H., Ochsenreiter, W.: Clock synchronization in distributed real-time systems. IEEE Trans. Computers 36(8), 933–940 (1987), `http://dblp.uni-trier.de/db/journals/tc/tc36.html#KopetzO87`
16. Kritikos, K., Plexousakis, D.: Semantic QoS Metric Matching. In: ECOWS. IEEE Computer Society, Zurich (2006)
17. Mahbub, K., Spanoudakis, G.: Monitoring WS-Agreements: An Event Calculus-Based Approach. Springer (2007)
18. Mok, A.K., Liu, G.: Efficient run-time monitoring of timing constraints. In: IEEE Real-Time and Embedded Technology and Applications Symposium, p. 252 (1997)
19. Moser, O., Rosenberg, F., Dustdar, S.: Non-intrusive Monitoring and Service Adaptation for WS-BPEL. In: WWW, pp. 815–824. ACM (2008)
20. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson, Prentice Hall (2008)
21. Patkos, T., Plexousakis, D.: DECKT: Epistemic Reasoning for Ambient Intelligence. ERCIM News (84), 30–31 (2011)
22. Popescu, R., Staikopoulos, A., Liu, P., Brogi, A., Clarke, S.: Taxonomy-driven Adaptation of Multi-Layer Applications using Templates. In: SASO (October 2010)
23. Shanahan, M.: The Event Calculus Explained. In: Veloso, M.M., Wooldridge, M.J. (eds.) Artificial Intelligence Today. LNCS (LNAI), vol. 1600, pp. 409–430. Springer, Heidelberg (1999)
24. Spanoudakis, G., Mahbub, K.: Non-Intrusive Monitoring of Service-Based Systems. International Journal of Cooperative Information Systems 15(3), 325–358 (2006)
25. Zeginis, C.: Monitoring the QoS of Web Services using SLAs - Computing metrics for composed services. Master's thesis, University of Crete, Greece (2009), `http://www.csd.uoc.gr/~zegchris/master_thesis.pdf`
26. Zengin, A., Marconi, A., Pistore, M.: CLAM: Cross-layer Adaptation Manager for Service-Based Applications. In: QASBA 2011, pp. 21–27. ACM (2011)