Computer Science • 15 (2) 2014 http://dx.doi.org/10.7494/csci.2014.15.2.173

Przemysław Dadel
Mariusz Balawajder
Dominik Radziszowski
Krzysztof Zieliński

# ADAPTIVE SOA STACK-BASED BUSINESS PROCESS MONITORING PLATFORM

**Abstract** *Executable business processes that formally describe company activities are well placed in the SOA environment, as they allow for the declarative organization of high-level system logic. However, to fully benefit from that element of abstraction, appropriate business process monitoring systems are required for both technical and non-technical users. Unfortunately, current solutions remain unsatisfactory.*

*This paper discusses the problem of business process monitoring in the context of the service orientation paradigm, in order to propose an architectural solution and provide implementation of a system for business process monitoring that alleviate the shortcomings of the existing solutions.*

*Various platforms have been investigated to obtain a broader view of the monitoring problem and to gather functional and non-functional requirements. These requirements constitute input for further analysis and system design. The monitoring software is then implemented and evaluated according to the specified criteria.*

*An extensible business process monitoring system was designed and built on top of OSGiMM – a dynamic, event-driven, configurable communications layer that provides real-time monitoring capabilities for various types of resources. The system was tested against the stated functional requirements, and its implementation provides a starting point for further work.*

*It has been concluded that providing a uniform business process monitoring solution that satisfies a wide range of users and business process platform vendors is a difficult endeavor. It is furthermore reasoned that only an extensible, open-source monitoring platform based on a scalable communication core has a chance to address all current and future requirements.*

**Keywords** BPM (Business Process Monitoring), BAM (Business Activity Monitoring), OSGi, Adaptive SOA

## 1. Introduction

Each established company enforces a set of rules, constraints, and guidelines that collectively describe its business activities. Depending on the maturity of the enterprise, corporate processes may refer to activities at various levels of abstraction, starting with a detailed description of a simple task all the way up to high-level business scenarios.

With the development of modern computational infrastructures, more and more tasks associated with definition, tracking, managing, and automating business processes are being handed over to computer systems. Business processes play a vital role in Service Oriented Architecture (SOA) [6], which is one of the most important paradigms for implementing enterprise software systems [21]. The Open Group's SOA Reference Architecture and IBM SOA Solution Stack (S3) model [2] places the business process layer near the top of the stack. In this layer, SOA supports application development by introducing composite services which orchestrate the information flow among a set of software services and human actors. These composite services are collectively called **business processes**.

Currently, there are an abundance of tools which support the composition of business processes from atomic services. They are typically used to describe specific steps which need to be taken in order to complete a given task. Such tools allow non-technical users to declaratively describe the business process flow, either as a document in a special-purpose language (e.g. BPEL – Business Process Execution Language) or with graphical interfaces. Formally-defined business processes run in execution engines which interact with services according to the specified flow.

Along with the notion that modern systems – particularly SOA-driven ones – are highly dynamic and support elastic reconfiguration to address changes in business requirements, a need emerges for a solution that enables tracking of the execution business process in such environments and perform effective analysis of the collected data. Ideally, this solution should allow target monitoring to be described in a declarative manner. This would, in turn, enable the monitoring system to be configured in a way that allows the efficient propagation of data necessary for the execution analysis process. The monitoring system should also address the heterogeneity of business process platforms. Moreover, as this layer of the SOA stack is particularly exposed to end-user interaction, proper visualization of process execution remains indispensable.

The goal of the presented study is to address the needs presented above. It is achieved through:

- investigation of existing business process platforms with respect to their monitoring capabilities,
- design of a generic architecture for business process monitoring,
- implementation of a system which supports end-to-end business process monitoring,

- evaluation of the system with respect to its functional and non-functional requirements.

The presented study adopts an event-driven architecture and several concepts related to dynamic monitoring of SOA systems [22], specifically declarative monitoring scenarios as well as separation of monitoring domains. Monitoring scenarios allow for on-demand monitoring of selected elements of the SOA environment, while domain separation allows multiple monitoring solutions to coexist in the same system. The presented solution is targeted for systems which exploit the OSGi-based Enterprise System Bus [13]. This means that the monitoring system naturally conforms to SOA principles and seamlessly integrates with existing software.

The proposed system is based on an Adaptive SOA [21] stack and inherits many of its valuable properties. These include an event-driven approach to data propagation as well as OSGiMM – a scalable and efficient communication backbone for system components that operate in distributed mode using OSGi containers.

In conclusion, the main contribution of this work is the proposed architecture of a business process monitoring system that matches the requirements presented, along with a working implementation that can be used to verify architectural assumptions and serve as a basis for further development.

This paper is organized as follows: section 2 specifies system requirements and describes related work, section 3 presents the architecture of the proposed system, while later sections contain evaluation results and conclusions.

## 2. Requirements and related work

The functional and non-functional requirements for Business Process Monitoring Platform are specified in this section. They should enable evaluation of the presented work's contribution and comparison with existing monitoring platforms. The presented requirements are a result of investigating and prototyping various monitoring approaches and expanding the focus of the monitoring solution to include business analysts along with technical users.

Functional requirements:

- FR1 – discovery and presentation of the existing business process engines, deployed processes, and active process instances,
- FR2 – on-demand monitoring of the selected discovered elements (mainly business processes),
- FR3 – presentation and up-to-date overview of business process definitions and the execution state of active process instances.

Non-functional requirements:

- NR1 – only necessary data should be transmitted between system elements,
- NR2 – the monitored business processes should not be affected by the monitoring process,

- NR3 – the architecture must be scalable and extensible, supporting multi-vendor infrastructures,
- NR4 – a standard data model must be provided for all monitored components.

An insightful researcher would point out that nearly all existing business-process-execution environments provide monitoring and management capabilities; however, these environments are typically incompatible with one another and expose only limited information about process instances and their activities. Filtering, data aggregation, and bottleneck analysis are rarely supported; moreover, there is often no direct support for the preselection of monitoring data (which introduces needless overhead resulting from collecting and storing unnecessary information). Other frequently-encountered problems include a lack of support for multiple-vendor engine implementations, inconvenient data access, inflexibility, and low information selectivity.

The motivation to reconsider and redesign the monitoring approach is twofold. Firstly, it is the need for efficient, selective monitoring of business process in the SOA environment regardless of the used business process language and execution environment. Secondly, this work completes an AS3 stack (by providing monitoring in the top layer) and verifies the distributed communication mechanisms proposed by AS3.

## 2.1. Adaptive SOA Solution Stack

Service orientation is a common approach to designing and implementing modern IT systems. To facilitate widespread use of SOA-based solutions, a set of widely-accepted architectural guidelines have been proposed along with an architectural template called SOA Solution Stack (S3), which can drive the development of SOA-compliant systems. S3 recommends a layered architecture, where each layer represents different concerns of the complex distributed system.

The Adaptive SOA Solution Stack Studio (AS3 Studio) [1] project adds adaptability to S3 as a means of alleviating the complexity inherent in managing and adjusting a distributed system. AS3 Studio is inspired by the notion of Autonomic Computing [8] and aims to enable the development of systems which will provide better utilization of IT infrastructures as well as more reliable Quality of Service (QoS) and Quality of Experience (QoE).

Adaptive SOA Solution Stack (AS3) proposes a policy-driven system adaptation which also supports cross-layer adaptation [21]. In addition to adaptation strategies, AS3 also deals with the problem of extending various S3 layers with adaptability mechanisms.

AS3 proposes a uniform pattern-based approach to adaptive S3 layer extensions, with dedicated components which constitute an adaptation loop with policy-driven management within each layer.

The presented Business Process Monitoring Platform exploits the adaptive integration layer of AS3 and follows the technology selection for adaptive SOA systems. This monitoring solution can be perceived as a partial implementation of adaptive

mechanisms in the S3 business process layer. The system may provide meaningful data concerning e.g., QoS or user-defined Key Performance Indicators (KPI), which can be used in the adaptation process in lower layers or affect the enterprise policy for redeployment of a more-optimal process.

## 2.2. Related work

Problem of Business Process Monitoring has a lot in common with the research on process mining in workflows. One interesting example [16] uses an event-based approach for unknown process discovery. Even though the examples presented in this work assume a predefined process model, system architecture can support processes that are not yet formalized. In the simplest case, the Complex Event Processing component could be used to dynamically enrich or build process definitions.

Monere [18] is a monitoring system that challenges the problem of availability in large-scale service-based systems. It monitors software system at various layers: BPEL processes, web services, application servers, and operating systems. Various data collection techniques are applied in this system: JMX, command-line tools, subscription to BPEL engines, request interceptors and custom component status pooling. Monere aims to help maintenance engineers in problem discovery in order to reduce Mean Time To Repair (MTTR). The platform presented in this article does not aim for such comprehensive monitoring of a software stack. It trusts that lower layers of Adaptive SOA systems will take care of critical conditions, either by problem notification or by means of autonomous problem resolution and adaptation of a specific system layer. The system that is illustrated here can be successfully used to monitor low-level metrics of business processes and consequently-targeted web services. However, the authorsñ target is to enable monitoring and discovery of real-time, non-trivial service interferences and tracking (Author's remark: (or measureing) in order to sustain or become aware of) business goals realized bythe business process.

ECMAF [20] is a framework which focus on life-cycle monitoring and adaptation of Service-Based Applications. Starting from end-user interest in web service execution and evolution, it tackles the problem of monitoring and adaptation across all functional system layers – with motivation and concepts similar to AS3. ECMAF assumes a different, more-coarse grained and layered view of a service-based system with the following layers: Business Process Management, Service Composition and Coordination, and Service Infrastructure, but useses similar concept of a well-defined event model to describe and transmit useful information. As this article is focused on a business process layer, it also addresses other problems specific to this layer: how to architecturally organize and abstract monitoring of heterogeneous processes regardless of technology used by a business process executor.

SOA-based systems are often built on Java-related technologies; subsequently, JMX is frequently a natural technology to control and monitor components at various system layers [10, 17]. In this work, OSGi is not a replacement technology for JMX. The component model of OSGi, with its dynamic service discovery capabilities, allows

us to conveniently build a monitoring system with SOA principles in mind. This is consistent with the authors' strong focus on promoting SOA not only by facilitating implementation and maintenance of SOA systems in general, but also by building solutions that are intrinsically compliant to SOA principles. Regarding the monitoring of a Java-based business process engine, an OSGi component could use JMX exported beans of such an engine to retrieve useful information and transform it into monitoring events.

Several investigated research works discuss the subject of monitoring information extraction from business process engines. One example, [19] proposes a solution based on a process performance model that is fed by events generated on the basis of an enriched business process deployment descriptor. Another attempt at business process monitoring is described in [5], where the authors explore instrumentation of the business process engine with a view towards generating informative events. The approach presented in this work is similar to both studies in that it applies a well-defined event model. Moreover, there is an apparent convergence with the first study's focus on business activity monitoring; however, the goal of the presented work is the effective organization of business process monitoring and delivery of results to end users. Nonetheless, both approaches mentioned above can be incorporated as partial solutions that fit into the generic architecture proposed in this paper.

As the first step of the presented research, various business process execution environments were studied – among them ApacheODE [4], JBoss jBPM [12], IBM WebSphere [11], and Oracle BPEL Process Manager [14]. Most of them provide certain built-in monitoring capabilities and offer various levels of external monitoring support. ApacheODE, BPEL Process Manager, and jBPM offer monitoring APIs that support notifications or pull-mode information extraction. No similar APIs were identified in Cardiff LiquidBPM [3] or Active Endpoints ActiveVOS data may potentially be fetched by other means; e.g., from the product's database or via business process instrumentation.

During the course of the presented study, an increase has been noted in the maturity of business activity monitoring solutions provided by leading IT vendors. For example, SAP NetWeaver now enables comprehensive monitoring of various aspects related to business process execution, and its Slipstream adds support for complex event processing [9]. Oracle Business Activity Monitoring is another mature product for business processes execution management and monitoring that offers a rich set of analytical tools. Newer versions of JBoss jBPM introduce powerful Drools-based processing of business process events.

Certain similarities emerge when comparing the products mentioned above with the presented study. However, most of the concepts detailed in this paper have evolved independently from third-party business process monitoring products (which have also undergone maturation in recent years). The presented system does not aim to compete with highly integrated business process and activity monitoring solutions offered by top IT vendors. Instead, the value of this work (and of the presented system) lies in its multi-vendor, open, and pluggable architecture which shares many features

with cutting-edge enterprise solutions. Business Process Monitoring Platform is unconstrained by proprietary APIs, and it is also well integrated with the Adaptive SOA environment. This enables optimization of SOA-based systems.

## 3. Architecture

This section presents the layered architecture of the business process monitoring system introduced above. First, it emphasizes the application of OSGiMM and then explains how its features facilitate development of the business process monitoring system.

### 3.1. OSGi Management and Monitoring

OSGi Management and Monitoring (OSGiMM) [22] is a comprehensive Enterprise System Bus (ESB) management and monitoring framework built on top of the OSGi container federation. The monitoring configuration assumes the form of user-specified scenarios that can be dynamically installed and modified during runtime in a distributed environment. The designed system provides selective monitoring capabilities which focus on specific system components.

Elements of the federation comprise an abstract dynamic structure called topology. The highest level of this structure is made up of OSGi containers that can contain various topology elements depending on domain implementation. For instance, in the OSGi domain, containers report the installed bundles and the exported services. Another example is the ESB domain, which consists of Service Assemblies instead of OSGi bundles. All features are transparent from the perspective of the working system. Following initial system configuration, management and topology monitoring features become seamlessly available to the user.

Elements of the architecture can be defined as follows:

- The Monitoring Scenario represents abstract monitoring goals defined by the monitoring configuration. An arbitrary number of independently managed monitoring scenario instances may be present in the system.
- The Instrumentation layer is responsible for discovering the topology, container state, and activity of the monitored service.
- The Data Collection layer defines methods for collecting and distributing monitoring data.
- The Data Processing layer collates statistics on the basis of monitoring data by applying the Complex Event Processing (CEP) engine.
- The Data Presentation layer communicates the status of the underlying layers to end users.

In OSGiMM, a domain is a collection of components and resources which interact with one another to reach a common goal that is subjected to the monitoring process. This system partitioning concept was introduced by OSGiMM to separate monitoring concerns for various aspects of an SOA-based system.

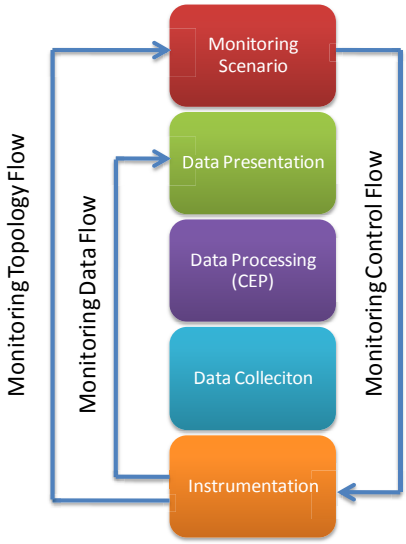


**Figure 1.** OSGiMM Architecture.

It offers the following features:

- component separation between domains, ensuring that any modifications within a given domain do not impact other domains,
- virtual communication channels dedicated to a specific topology within a domain,
- remote OSGi (OSGi) service management within a domain, separating services registered in different domains.

At present three concurrent domains are available for the OSGiMM communication layer: ESB, SCA and BPEL (the latter of which is the subject of extensions presented in this work).

## 3.2. Architecture of Business Process Monitoring Platform

Key assumptions made when designing and implementing the business process monitoring solution came from investigating how business process execution platforms are deployed in enterprise infrastructures. It is uncommon for business process engines to act as standalone components. Instead, they usually constitute broader service-oriented infrastructures. One consequence of this fact is that multiple heterogeneous business process components may coexist, forming an interaction topology that needs to be discovered.

As a result, the proposed monitoring solution abandons the popular concept of extending business process engines with a dedicated monitoring console. Instead, it exploits a data bus architecture which enables different, multi-vendor engines to be connected by exchanging well-defined messages through dedicated connectors.

The presented system is an event-based solution where execution of each business process element generates a piece of data contributing to an execution trace of the orchestrated process. These pieces of data are events based on a well-defined model. Effective distribution of events across the system is facilitated by OSGiMM.

The core layers of the system architecture are presented in Figure 2:

- ESB with OSGiMM – a communication backbone that provides seamless integration of system components, their discovery as well as transport of monitoring data using mechanisms specified by the monitoring scenario.
- The Monitoring Domain – a specification of hierarchical component types that can be discovered and monitored. The business process monitoring domain is called the BPEL Domain. Runtime configuration of BPEL Domain elements follows the BPEL Domain Monitoring Topology, hereafter simply called the topology. The Monitoring Domain consists mainly of business process engines with their respective sensors – BPM Engine Monitors, business processes, independent event processing components and infrastructure nodes where these elements reside.
- The Monitoring Console – a GUI component that enables configuration of the monitoring system and presents key system features to the user.

Business Process Monitoring Platform was designed with service orientation principles in mind.

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 (e.g. Monitoring Consoles). The architecture allows for event interceptors (e.g., rule-based event processors) that are hybrids of these two client types. Such processors may intercept the flow on events from other event sources to Monitoring Console for the purposes of event processing [7].

An advanced implementation that leverages the benefits of OSGiMM may use event type and topology shape information to intelligently route events to interested subscribers, complying with the imposed restrictions on accepted events. Such a communication layer facilitates development of large-scale, highly configurable and filterable event-driven systems of which the presented monitoring solution is an example.

Each monitored business process engine is associated with a dedicated monitor that generates standardized notifications of changes in the executed process. The communication layer is responsible for the propagation of such events as well as filtering them when no subscriber is interested in a particular event type.

The presented monitoring platform uses OSGiMM which admits model-based declarative definitions of the monitoring process by providing a *monitoring scenario*. In light of the above, business process monitoring is an example of a well-defined monitoring scenario. Installing a monitoring scenario is equivalent to creating a *monitoring subscription* for the business process execution events, whose flow is enabled by the activation of the scenario itself. As the system's main goals include monitoring of

specific topology elements as well as the topology itself, *topology subscription* is also supported.

When compared to the layered architecture of OSGiMM, a relocation of the business process analysis logic can be observed. While standard OSGiMM metric events are processed by internal CEP processors, analytical logic(rule-based event processors) is shifted upwards in the layered architecture and becomes an optional and reconfigurable topology element. This change enables easier business rule management by users.

The system architecture relies heavily on the mechanisms provided by OSGi, which naturally form an SOA environment in a single Java Virtual Machine. System elements are exposed as OSGi services, which can dynamically discover one another and support runtime reconfiguration.

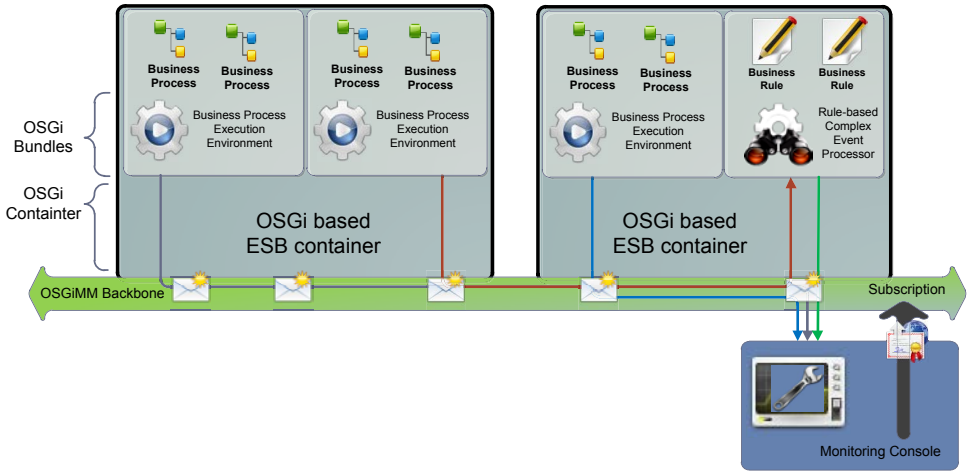


**Figure 2.** Business Process Monitoring Platform – High Level Architecture.

## 3.3. Communication layer Extensions

This section describes communication layer enhancements for business process monitoring. The main concepts that are subject to extension and implementation are the monitoring domain and event subscriptions.

For the purpose of business process monitoring, a new domain called the BPEL Domain has been introduced. (This step reflects the early stage of research and implementation where only BPEL processes were the subject of research interest). The specification of this new domain comprises of event types that are exchanged within the domain, a definition of the domain topology, and implementation of domain-specific agents residing on each of the available infrastructure nodes.

In OSGiMM, the structure of the domain is defined by its implementers. The structure used for the purposes of business process monitoring aims to reflect potential deployment of business process related elements in the adaptive SOA infrastructure. The BPEL domain is made up of the following components:

1. BPM Container – deployed on each of the OSGiMM's infrastructure nodes,
2. BPM Components, which are mainly BPM Engine Monitors associated with business process execution engines or event processing interceptors,
3. deployed business processes.

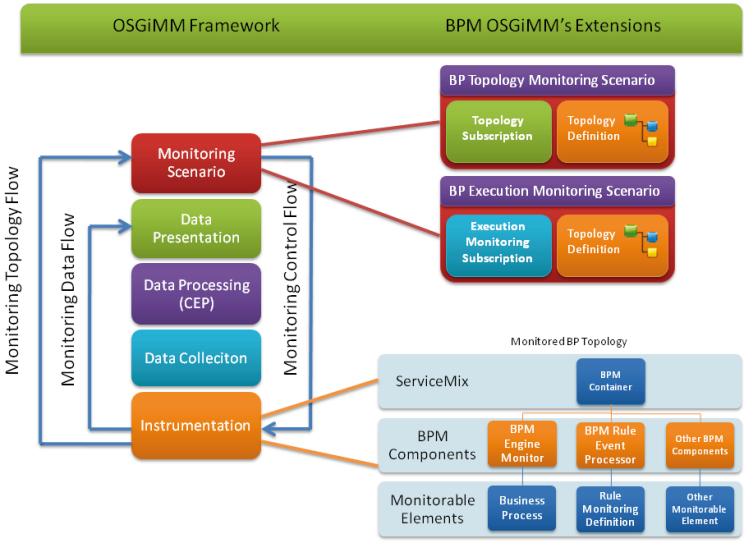


**Figure 3.** OSGiMM-based business process monitoring.

BPM Engine Monitor is one of the most important elements of the BPEL domain topology. It provides a bridge between the process-execution environment and the monitoring system. Its responsibilities focus on enabling business-process monitoring by generating events on process-execution progress as well as contributing to the domain topology by providing a list of processes deployed in the monitored engine. BPM Engine Monitor can be perceived as an adapter that monitors the execution of a specific engine via instrumentation, registration of custom listeners, or polling for status data.

### 3.3.1. Business Process Monitoring Events and Subscriptions

For the propagation of business process monitoring data in the presented system two event categories are defined in OSGiMM:

- Business processes, along with all dependent entities such as XML definition files deployed in the business process engine, business process execution history, and information about changes which correspond to a specific topology category.

- Monitoring category, which carries information related to process-execution-state changes, changes in variable values, web service calls, evaluation failures, and exceptions. Monitoring data enables the console to recreate the business process execution path for the purposes of business process visualization and analysis.

Monitoring and topology data are propagated across the system in the form of **monitoring and topology events.**

OSGiMM's monitoring scenario concept is similar to the publish/subscribe approach commonly used by Java Messaging Service (JMS).

Installation of the monitoring scenario, which defines the system endpoints (explicitly or implicitly), results in a subscription for monitoring and topology events in domain components by event receivers (Monitoring Console or event processing interceptors).

On the basis of OSGiMM's monitoring scenarios, a differentiated set of subscriptions has been proposed to match the purposes of business process monitoring.

Topology subscriptions are used to gather information about topology-element-state changes, whenever a new container, domain component, or business process is deployed. When such a change occurs in the topology, relevant topology events are passed to receivers (mainly Monitoring Console) through the channel created by each receiver's topology subscription. In the presented system, topology subscriptions are further differentiated on the basis of the provided level of detail. A generalized topology view requires no more than component identifiers, their types and lists of children; however, for more-exhaustive element analysis, detailed descriptions of topology elements have to be sent as well. In both cases, specific topology subscriptions are introduced for more-efficient utilization of the communication layer resources. As the user is not typically interested in monitoring the entire topology, topology subscription is extended with a convenient mechanism for restricting its scope. This allows users to precisely define which topology elements are subject to monitoring.

Monitoring subscriptions are used to create information channels that carry events concerning process execution and rule-based event processing. Similarly to topology subscriptions, monitoring subscriptions can also be restricted to a given fragment of the topology.

There are two categories of monitoring subscriptions:

- Process Subscription, which enables monitoring of business processes contained within a specified topology.
- Rule Monitoring Subscription, which configures rule-based event processing interceptors to collect events from other components and dispatch results to subscribers.

Figure 4 presents an example of a well structured BPEL Domain Monitoring Topology, managed by OSGiMM with support for selective subscription to monitoring data from a given fragment of the topology.

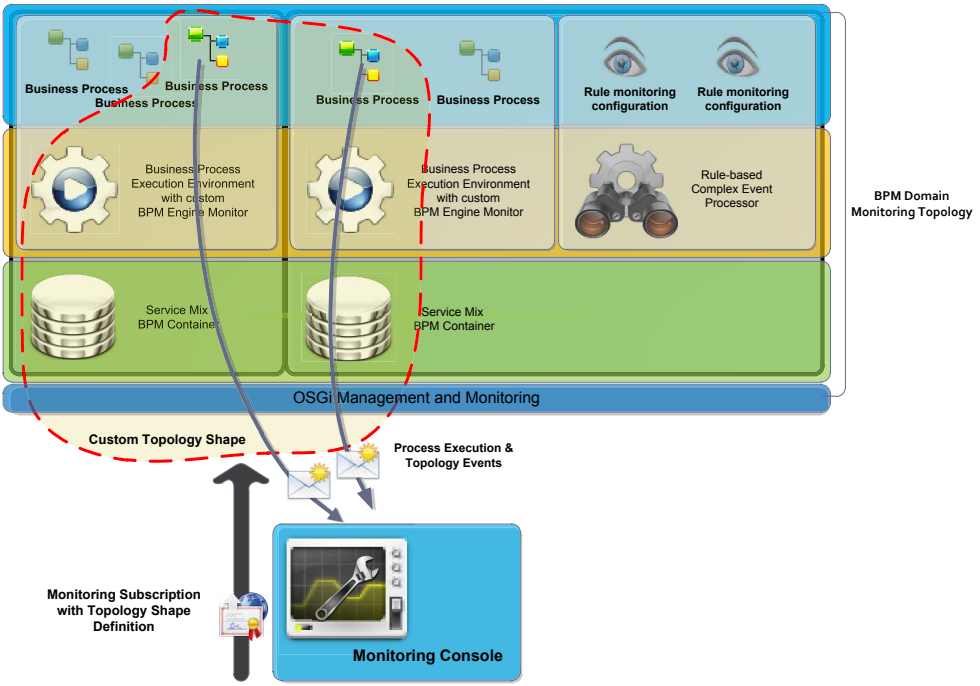


**Figure 4.** BPM Domain Monitoring Topology with selective subscriptions.

## 4. Implementation

As the implementation of the system follows the component model, all application modules are OSGi bundles deployed in a distributed environment based on ServiceMix or, in the case of Monitoring Console, in an Equinox OSGi container. One of the most significant benefits of OSGiMM is its support for distributed OSGi services. This enables flexible binding to services exported in the ServiceMix federation by remote Monitoring Consoles.

### 4.1. BPEL Monitoring Domain

The BPEL domain topology elements form a tree-like structure, while its topology class structure follows the composite design pattern presented in Figure 5.

*ITopologyElement* is an interface which all of the topology elements need to implement. Each element must, therefore, provide the following:

**name** – simple element name (for example the monitoring engine's name, the name of the deployed process etc.);

**path** – used to uniquely identify topology elements; the path consists of topology element predecessors arranged in a tree-like structure;

**list of children** – list of *ITopologyElement* children of the element, enabling the creation of tree-like structures.

This structure greatly simplifies implementation and enables uniform treatment of all topology elements.

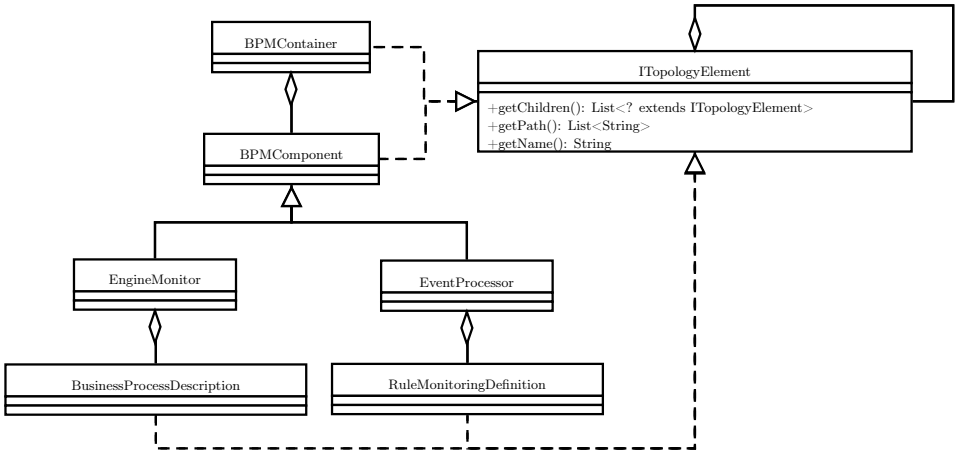


**Figure 5.** Topology element class diagram.

The implementation of the main BPEL Domain elements is described in the following sections.

BPM Container is a topology component responsible for managing OSGiMM subscriptions and OSGiMM remote listeners in the BPEL Domain. A BPM Container is available on each of the business process monitoring enabled instances of ServiceMix. It controls and provides access to the communication backbone for BPM Components running within the same ServiceMix instance. It is also capable of dynamically discovering BPM components within a ServiceMix instance, using OSGi mechanisms.

When a new component is found by *ComponentServiceTracker*, the BPM Container automatically forwards all of the relevant active subscriptions to the newly-registered BPM Component. Additionally, the container provides mechanisms for aggregating statistical data related to the subscribed events.

In line with OSGiMM guidelines, domain agents responsible for processing subscriptions and managing the domain topology structure had to be implemented. In the BPEL Domain, *BPMMonitoringAgent* and *BPMTopologyAgent* serve this purpose. Both are implemented in accordance with the OSGiMM developer guide [15].

When a subscription is registered in the system, the registration method method is invoked on *BPMMonitoringAgent* or *BPMToplogyAgent*, depending on the subscription type. The BPM Container uses *TopologySubscriptionProcessor* or *MonitoringSubscriptionProcessor* to verify whether it matches the target topology of the subscription. If this condition is satisfied, the subscription is added to the

container's *SubscriptionRegistry*; otherwise, it is ignored. As the next step, the BPM Container processes all of the registered BPM Components, trying to match the target topology with each BPM Component's partial topology. If the BPM Component is within the scope of the target topology, the container registers the subscription in that particular BPM Component.

BPM Component is a generic monitoring domain element that represents the common functionality of the components deployed in a BPM Container. It contains base classes that can be used to implement BPM Components with:

- one-way communication (e.g. BPM Engine Monitor), that can only receive subscriptions and send events to subscribers,
- two-way communication (e.g. BPM Rule Event Processor), which, additionally, can register their own subscriptions in the OSGiMM communication layer.

BPM Component registration in the BPM Container is based on the OSGi service tracking mechanisms. The BPM Component manifests itself in the system by implementing the *IBPMComponent* interface and exporting the *IBPMComponent* OSGi service.

The ApacheODE Engine Monitor can be examined as a sample BPM Component implementation.

ApacheODE Engine Monitor is ApacheODE's [4] dedicated implementation of BPM Engine Monitor. One of the main design guidelines for the business process engine monitoring is transparency of the monitoring component. In ApacheODE, it is not necessary to instrument the executed process or to modify the execution engine itself.

Data is collected using two channels exposed by ApacheODE:

- Custom implementation of *BPELEventListener* that is registered in the engine, which collects execution events and transform them to the standardized event format used in the system. The converted events are sent by the ApacheODE Engine Monitor to the registered subscribers,
- ManagementAPI which enables the retrieval of information about the deployed processes. ApacheODE is periodically polled for changes which trigger events, informing subscribers about any topology changes.

BPM Rule Event Processors are BPM Components responsible for intercepting and processing the monitoring events. A Business Process Description corresponds to a process deployed in a business process execution engine. Rule Monitoring Definition represents user-supplied configuration of the rule-based event processors.

## 4.2. System subscriptions

As described in section 3.3, subscriptions are divided into two categories: **topology subscription** and **monitoring subscription**.

Topology subscription is used to create data channels in the communication backbone which are then used to propagate topology events. Business Process Monitoring Platform uses a differentiated set of topology subscriptions aimed at selective and

effective use of communication layer resources. A description of these subscriptions follows.

**BPM Topology Subscription** is a base topology subscription used within the BPEL domain. Topology Subscriptions that do not extend this base class are automatically ignored. BPM Topology subscriptions can define target topology shapes by using topology restrictions.

**Global Topology Subscription** is used to connect to a global virtual channel that gathers information about all topology changes within a domain. When a topology change occurs, an appropriate event is triggered and dispatched to all registered receivers. By design, events sent through the channel created by the Global Topology Subscription should be relatively small. For example, when a new business process is deployed, the topology change notification event will only contain the name of the deployed process, its deployment time-stamp, and a unique topology path. It will not provide the business process definition of the process. The rationale behind this approach is to reduce network traffic since, in a large enough topology, only a few business processes will typically be subject to monitoring. The same approach is used for Rule Monitoring Definitions.

**Detailed Topology Subscription** complements the features of the Global Topology Subscription. This subscription enables clients to acquire detailed information concerning specific topology elements. For example, it can be used to obtain Business Process definitions or Rule Monitoring Definitions when needed. Whenever a change in the target topology occurs, an event describing this change is triggered.

Monitoring subscriptions create data channels used to propagate monitoring events. They rely on the same scope restriction mechanisms as topology subscriptions; however, they are differentiated in regard to the information conveyed instead of their granularity.

**Monitoring Subscription** is the base class for monitoring subscriptions used within the BPEL domain. All subscriptions have to inherit from *BPMMonitoringSubscription*; otherwise, they are ignored.

**Process Subscription** handles monitoring subscriptions for a given subset of business processes.

**Rule Monitoring Subscription** is used to configure BPM Rule Event Processor to collect events from the defined topology fragment which should be processed by the rule-based event processing engine with a given set of rules.

Each subscription is dedicated to a specific topology. Topology restrictions can be applied to filter out a subset of the topology. The mechanism is based on a hierarchical set of pattern-based restrictions that refer to topology element names and types.

Each topology restriction consists of a list of patterns that can be either positive (matching elements included in the topology fragment) or negative (matching elements excluded from the monitoring scope). Negative patterns take precedence over positive ones.

## 5. Case study

This section presents the monitoring process for a selected business process. Monitoring is performed in the test environment, and the system is evaluated in regard to its functional and nonfunctional aspects. The presented use-case scenario and its scalability tests highlight the advantages of Business Process Monitoring Platformin a distributed environment.

### 5.1. Test environment

Tests of the monitoring platform were conducted in the following distributed environment:

- Three virtualized environments (SUN x6440, 2xPC Core i5, 4 GB RAM) running 9 Ubuntu Server 11.04 VMs (KVM-QUMU).
- Each virtual machine running a single ServiceMix 4.2 instance named after a Polish city (Krakow, Warszawa, Gdansk, etc.).
- BPEL engine (ApacheODE) and its respective BPM Engine Monitors deployed in each ServiceMix container.
- Monitoring Console deployed on a personal computer under Ubuntu 10.04.
- Each ApacheODE instance running an identical set of deployed processes: CreditApp (8), ScoringService, HelloWorld2, HelloWorkd2-RPC, Ping, Pong.

The main business process user test is *CreditApp* which processes banking loan requests. This application may delegate some of its processing duties to the *ScoringService* .

### 5.2. Monitoring Process

To better explain the operation of the monitoring framework, a generic monitoring scenario (Fig. 6) is presented.

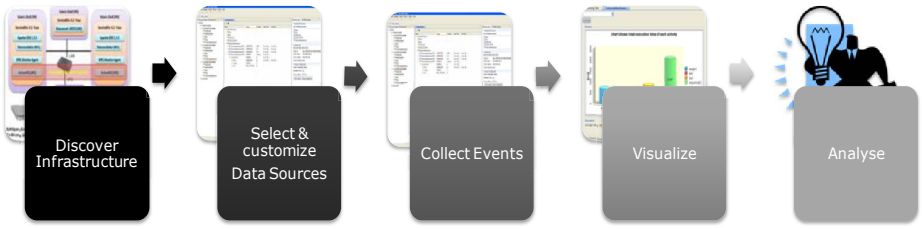


**Figure 6.** Monitoring Process.

Since business processes, their execution platforms and other business process related components initially exist in an undiscovered infrastructure, and the primary function of Business Process Monitoring Platform is the discovery of these elements and their connection topology. Monitoring Console can then visualize the topology structure and provide information about its individual elements (*requirement FR1* ).

Once the possible data sources become known, Monitoring Console can be configured to monitor interesting topology elements. The user selects a topology fragment and Monitoring Console subscribes to changes in the selected elements (Fig. 7). Following this step, the selected fragment is subject to monitoring, and the console collects events concerning topology changes and execution of business processes (*requirement FR2*).

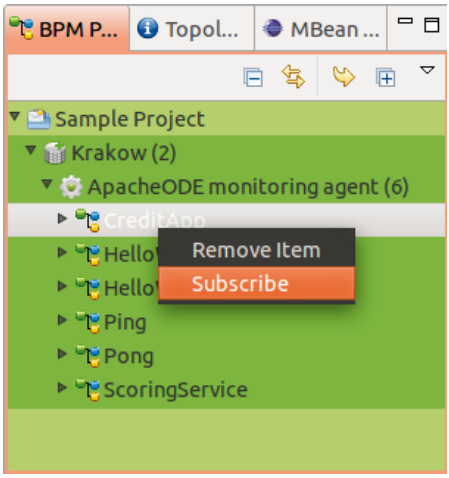


**Figure 7.** Subscription to the selected process.

When a process instance is executed, Monitoring Console can visualize its progress by dynamically highlighting the completed elements in process model view (Fig. 8). Detailed progress status and a list of collected events are made available for analysis. Process summary charts and statistics are also provided (*requirement FR3*).

Finally, the user may create rules for detecting additional business-specific conditions in the executed process and configure rule-based event processors (if they are available in the topology) to test the exchanged events against these rules.

## 5.3. Performance and Scalability Test

To evaluate nonfunctional requirements concerning data duplication (NF1) and scalability (NF3), another scenario is proposed. This test case shows how the system behaves depending on the number of active Monitoring Consoles. Starting with a single console attached to the monitoring infrastructure, the number of consoles is progressively increased (adding one per iteration). Prior to starting a new console, the CreditApp business process is executed on a single ServiceMix instance. The monitoring scenario relies on a subscription which is restricted to a subset of the topology, selecting only these CreditApp and ScoringService processes deployed on Krakow, Gdansk and Warszawa nodes.

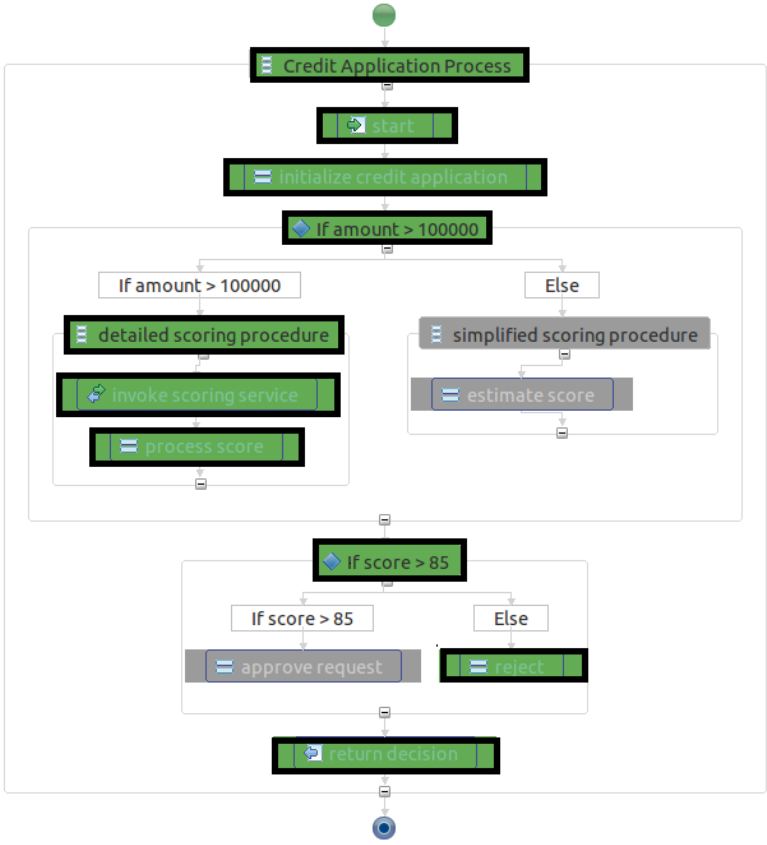


**Figure 8.** Execution path of the monitored process.

The steps listed below are repeated for up to four active instances of Monitoring Console:

1. Start monitoring console.
2. Create monitoring project.
3. Subscribe to CreditApp and ScoringService.
4. Execute the CreditApp process three times on a randomly-selected monitored engine.

For a single Monitoring Console 309 monitoring events describing process progress are generated (103 per process instance execution). CreditApp and ScoringService process definitions are sent from each of the monitored execution engines as topology events. This step is required, as process definitions may differ slightly between execution engines. Table 1 shows that the number of generated monitoring events remains constant and does not change with the number of receivers, while the topology event count grows linearly with the number of monitoring consoles.

**Table 1**
Amount of monitoring (m. ev.) and topology (t. ev.) events propagated in the system with different number of receivers.

| | 1 console | | 2 consoles | | 3 consoles | | 4 consoles | |
|---|---|---|---|---|---|---|---|---|
| **ServiceMix** | **t. ev.** | **m. ev.** | **t. ev.** | **m. ev.** | **t. ev.** | **m. ev.** | **t. ev.** | **m. ev.** |
| Krakow | 2 | 206 | 4 | 0 | 6 | 206 | 8 | 103 |
| Gdansk | 2 | 103 | 4 | 309 | 6 | 103 | 8 | 103 |
| Warszawa | 2 | 0 | 4 | 0 | 6 | 0 | 8 | 103 |
| **Total** | **6** | **309** | **12** | **309** | **18** | **309** | **24** | **309** |

The number of generated monitoring events is independent from the number of monitoring consoles due to intelligent subscription management provided by the OSGiMM communication layer. Linear growth in the number of topology events is due to the fact that process definitions need to be sent separately to each instance of Monitoring Console (this behavior is expected). Business process definitions may be relatively extensive, and they do not follow the propagation rules described in 4.2. Instead, they are sent on demand upon creation of a new topology subscription. As console instances become active at different times, they are unable to reuse a single subscription.

## 6. Conclusions and further research

Analysis of various business process platforms leads to the conclusion that uniform provisioning of monitoring capabilities for all such platforms is impossible. Instead, a specification of an event-driven system is defined, where platform-specific adapters (BPM Engine Monitors) conceal the differences between the specific implementations of the process orchestration engine. BPM Engine Monitors generate events that reflect the business process engine state, while the presented system takes care of distribution and visualization of such events.

The generic business process event model can be applied to other process-like execution engines, or indeed to any execution platform that follows the concept of workflow-based declarative description of component interaction. The only requirement is to ensure that the platform generates events that are based on the presented model and contributes to well-defined Business Process Monitoring Platform extensions.

The resulting system is an example of an event-driven solution that leverages the innovative mechanisms provided by AS3. More specifically, it implements efficient integration and communication, reducing data redundancy, enabling direct subscriptions to specific infrastructure elements, and facilitating dynamic discovery of the business process infrastructure. The presented business process monitoring solution also serves the purpose of validating the AS3 Studio's adaptive integration layer in terms of its functionality and performance. Moreover, it shows how the extensible

nature of OSGiMM can be leveraged to introduce a new monitoring domain whose structure, events, and subscriptions correspond to the nature of the monitored elements.

The field of business process monitoring is brimming with challenges. The issues addressed here should be treated as the tip of the iceberg, representative of a whole class of problems. When discussing technical aspects related to system implementation, there is always room for improvement in performance, communication layer efficiency, transmitted data filtration and selectivity. From a business point of view, visualization and statistical components might be significantly extended. For greater ease of use, stronger relationships between the process model, its semantics, and monitoring rules could be introduced. The idea is that the user could semantically tag process elements that supply important data, while the system would hint at a typical KPI that could be monitored, as well as automatically generate monitoring rules suitable for these processes.

Another interesting direction of development is adding support for new engines and business process notations. Ideally, a standardized notation such as Business Process Modeling Notation (BPMN) should be used.

In the context of AS3 Studio, it would be highly beneficial to close the automated adaptation loop in the business processes layer. An enticing possibility is to develop a solution capable of adjusting business processes and ensuring that that business goals are met in the most efficient way possible. The monitoring and analysis parts of such a solution are already in place; however, effectors need to be investigated and implemented.

## Acknowledgements

*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] AGH University of Science and Technology: *AS3-Studio*. `https://www.soa.edu.pl/web/guest/tools`.

[2] Ali A., Zhang-Jie, Michael E., Abdul A., Kishore C.: S3: A Service-Oriented Reference Architecture. *IT Professional*, 9: 10–17, 2007. ISSN 1520-9202. `http://dx.doi.org/10.1109/MITP.2007.53`.

[3] Autonomy Cardiff: *Cardiff LiquidBPM: Embeddable Business Process Management*, 2007. `http://publications.autonomy.com/pdfs/Power/Product\%20Briefs/Business\%20Process\%20Management/LiquidBPM/`.

[4] Brown P.: *An Introduction to Apache ODE*. InfoQ, 2007. `http://www.infoq.com/articles/paul-brown-ode`.

[5] Duszka T., Janczak J.: *Badanie wydajności systemów o architekturze SOA*. Master's thesis, AGH University of Science and Technology, 2008.

[6] Erl T.: *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007. ISBN 0132344823.

[7] Etzion O., Niblett P.: *Event Processing in Action*. Manning Publications Co., Greenwich, CT, USA, 1st ed., 2010. ISBN 1935182218, 9781935182214.

[8] Ganek A. G., Corbi T. A.: The dawning of the autonomic computing era. *IBM Syst. J.*, 42: 5–18, 2003. ISSN 0018-8670. http://dx.doi.org/10.1147/sj.421.0005.

[9] Janiesch C., Matzner M., Müller O.: Slipstream: A BAM Proof of Concept using Standard Software. In: *Proceedings of the 9th International Conference on Business Process Management (BPM) Demo Track*, p. 41. Eigenverlag, Clermont-Ferrand, 2011.

[10] Jarzab M., Zielinski K.: *Framework for Consolidated Workload Adaptive Management*. In: *Software Engineering in Progress*, L. Madeyski, M. Ochodek, D. Weiss, J. Zendulka, eds., pp. 17–30. NAKOM, 2007. ISBN 978-83-89529-44-2.

[11] Kloppmann M., Konig D., Leymann F., Pfau G., Roller D.: Business process choreography in WebSphere: Combining the power of BPEL and J2EE. *IBM Systems Journal*, 43(2): 270 –296, 2004. ISSN 0018-8670. http://dx.doi.org/10.1147/sj.432.0270.

[12] Koenig J.: *JBoss jBPM white paper*. 2004. http://www.jboss.com/pdf/jbpmwhitepaper.pdf.

[13] Masternak T., Psiuk M., Radziszowski D., Szydlo T., Szymacha R., Zieliński K., Zmuda D.: *ESB - Modern SOA Infrastructure*. 2010.

[14] Oracle: *Oracle BPEL Process Manager*. http://www.oracle.com/technetwork/middleware/bpel/overview/index.html.

[15] Psiuk M.: *OSGiMM Developer Guide*. AGH University of Science and Technology. https://www.soa.edu.pl/AS3-Studio/?q=node/37.

[16] Van der Aalst W., Weijters T., Maruster L.: Workflow mining: discovering process models from event logs. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9): 1128–1142, 2004. ISSN 1041-4347. http://dx.doi.org/10.1109/TKDE.2004.47.

[17] van Lessen T., Leymann F., Mietzner R., Nitzsche J., Schleicher D.: A Management Framework for WS-BPEL. In: *on Web Services, 2008. ECOWS '08. IEEE Sixth European Conference*, pp. 187–196. 2008. http://dx.doi.org/10.1109/ECOWS.2008.25.

[18] Wassermann B., Emmerich W.: Monere: Monitoring of Service Compositions for Failure Diagnosis. In: *Proceedings of the 9th International Conference on Service-Oriented Computing*, ICSOC'11, pp. 344–358. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-25534-2. http://dx.doi.org/10.1007/978-3-642-25535-9_23.

[19] Wetzstein B., Strauch S., Leymann F.: Measuring Performance Metrics of WS-BPEL Service Compositions. In: *Networking and Services, 2009. ICNS '09. Fifth*

*International Conference on*, pp. 49–56. 2009.
http://dx.doi.org/10.1109/ICNS.2009.80.

[20] Zeginis C., Konsolaki K., Kritikos K., Plexousakis D.: ECMAF: An Event-Based Cross-Layer Service Monitoring and Adaptation Framework. In: *Service-Oriented Computing - ICSOC 2011 Workshops*, G. Pallis, M. Jmaiel, A. Charfi, S. Graupner, Y. Karabulut, S. Guinea, F. Rosenberg, Q. Sheng, C. Pautasso, S. Mokhtar, eds., *Lecture Notes in Computer Science*, vol. 7221, pp. 147–161. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-31874-0.
http://dx.doi.org/10.1007/978-3-642-31875-7_15.

[21] Zieliński K., Szydło T., Szymacha R., Kosiński J., Kosińska J., Jarząb M.: Adaptive SOA Solution Stack. *Services Computing, IEEE Transactions on*, **vol. PP**(99), p. 1, 2011. ISSN 1939-1374.
http://dx.doi.org/10.1109/TSC.2011.8.

[22] Zmuda D., Psiuk M., Zieliński K.: Dynamic monitoring framework for the SOA execution environment. *Procedia Computer Science*, 1(1) 125–133, 2010. ISSN 1877-0509. http://dx.doi.org/10.1016/j.procs.2010.04.015. ICCS 2010.

## Affiliations

**Przemysław Dadel**
AGH University of Science and Technology, Krakow, Poland, pdadel@agh.edu.pl

**Mariusz Balawajder**
AGH University of Science and Technology, Krakow, Poland, balawajd@student.agh.edu.pl

**Dominik Radziszowski**
AGH University of Science and Technology, Krakow, Poland, dr@agh.edu.pl

**Krzysztof Zieliński**
AGH University of Science and Technology, Krakow, Poland, kz@agh.edu.pl

**Received**: 21.10.2013
**Revised**: 20.02.2014
**Accepted**: 3.03.2014