A Framework for Hierarchical and Recursive Monitoring of Service Based Systems[1]

Marco Comuzzi and George Spanoudakis
Department of Computing, City University
Northampton Square, EC1V 0HB, London, UK
{sbbd286, G.Spanoudakis}@soi.city.ac.uk

Abstract

Runtime monitoring of Service Based Systems (SBSs) usually relies on information derived from I/O messages exchanged within business processes implementing services. When service provisioning is regulated by complex Service Level Agreements (SLAs) between service requesters, (composed) services, and infrastructure providers, monitoring may require additional features, such as (i) coordination among events captured at different sources involved in service provisioning and (ii) delegation of properties monitoring to local sites. This paper discusses an architecture and engagement protocol supporting the two aforementioned requirements for monitoring complex SLA-driven service provisioning.

1. Introduction

Runtime system monitoring, as opposed to static system analysis and testing is often the only meaningful way to perform verification of *Service Based Systems (SBSs)*, in which both the involved software services and infrastructural elements may change dynamically according to contextual factors, such as the system load or the availability of new components. SBSs are dynamically evolving software systems comprising loosely coupled software services that may be substituted at runtime when they become unavailable or no longer satisfy non functional requirements, usually referred to as quality of service (QoS) properties [7]. Furthermore, the infrastructure on which the services of an SBS are executed may incorporate heterogeneous components and change dynamically. Services may, for example, be accessed over local area networks or through mobile devices, while service providers may change their service provisioning infrastructure (e.g web and hardware servers) according to the system loads they experience or adaptable quality profiles negotiated with service consumers.

Usually, existing approaches on runtime monitoring of SBSs focus on monitoring workflow-based systems (i.e. systems in which a *reference business process* coordinates the constituent software services) through either the interception of I/O messages exchanged between the business workflow that coordinates the services of the system and these services [3] or the instrumentation of the workflow executable code with monitoring-related activities [2]. Thus, current approaches to SBS monitoring do not consider some basic features that may characterize a complex SBS.

More specifically, it should first be noted that an SBS can be *recursively* defined, that is, a reference business process requiring monitoring may orchestrate local services which are recursively defined as a composition of other local services. In such cases, monitoring information may need to come from each of the services in the complex recursive service composition. Furthermore, SBS are *hierarchically* implemented. Required monitoring information may derive from business level Key Performance Indicator (KPIs) reported in an SLA established with the end-user. Such KPIs result in properties verifiable at the workflow or service interface and these can be then translated into properties of the infrastructure on which services are being are executed. Typical is the case of response time, usually an archetypical dimension defining service QoS in an SLA. A business KPI may specify constraints on the average response time of a service in a given time window. The KPI is translated on properties referring to the timestamps of service calls and responses that can be captured at the service interface. Eventually, further properties influencing the service response time, such as the length of required DB queries, server load, or network delay, can be captured from the infrastructure on which the service executes.

In this context, we aim at designing a framework for event-based monitoring of SBS for complex SLA-driven service provisioning. On the one hand, the framework should be hierarchical, allowing the monitoring of properties at different layers of the SBS, such as service composition, i.e. the workflow execution environment, service invocation, and service execution, i.e. the set of resources on which each single service executes. On the other hand, the framework needs to be recursive, allowing the monitoring of properties of a workflow and, recursively, of all the (composed) services which constitute the workflow.

The framework is constituted by the architecture of the monitor and an engagement protocol. The monitor is able to coordinate monitoring events coming from different elements composing the SBS and to delegate the monitoring of rules

[1] The research has been supported by the European Community under the SLA@SOI FP7 Project (grant agreement n. 216556).

derived from SLAs to local monitors, such as the ones that may be deployed at the infrastructural level. The engagement protocol is required to set up at runtime the monitoring infrastructure in a transparent way.

Besides addressing the recursive and hierarchical nature of SBS, the delegation of monitoring rules aims at (i) improving the scalability and performance of the monitoring process, avoiding a single centralized monitor and (ii) exploiting the specificity of local monitors designed for monitoring properties at different layers of an SBS. On the one hand, in fact, a centralized monitor may become a bottleneck for the monitoring process, since it needs to process monitoring information provided by several elements in the SBS. This becomes a paramount issue, for instance, if the SBS reference workflow is primarily hosted on a mobile device, with limited computing power and memory capacity. On the other hand, our monitor is also able to provide a coordination framework for delegating rule monitoring to local monitors attached, for instance, at the workflow interface or at specific infrastructural elements that need to be monitored.

The paper is organized as follows. In Section 2, we introduce an example illustrating the need for the architecture we introduce in the paper. In Section 3, we describe the monitoring capabilities of services required by our monitoring framework. In Section 4, we present the architecture and usage scenarios of the monitoring framework. In Section 5, we discuss the interface through which services make their monitoring capabilities available to the framework and the engagement protocol to establish the monitoring process. In Section 6, we discuss related work and in Section 7 we present some basic concluding remarks for our approach.

2. A Motivating Example

As an example of the heterogeneity and complexity of SLA monitoring that our approach aims to address, consider a retail SBS supporting the management of purchases on a mobile e-commerce website. The coordinating business process of this SBS is called *Purchase Business Process* (PBP). As shown in Figure 1, PBP is executed locally on the customer's mobile phone and implemented as the sequential composition of three different local services, namely *ManageCart*, *Checkout*, and *BookSale*. The customer is operating in an area covered by a mobile 3-G network managed by the generic Mobile Network Manager. Each service in PBP is offered by a different service provider, possibly on a heterogeneous set of service provisioning infrastructures.

More specifically, the *Checkout* service (CP) is a composite service itself involving the services *ExecutePayment* and *ConfirmPayment*. The *ExecutePayment* service is implemented as a workflow, called *Execute Payment Process* (*EPP*), which is hosted and executed by a Financial Service Provider (e.g. a bank). In this process, the credit card number provided by the customer is first validated. Validation is performed by the *ValidateCard* service which issues a transaction ID. Then, EPP debits the total amount of the purchase to the cardholder's account using the service *DebitCard*.

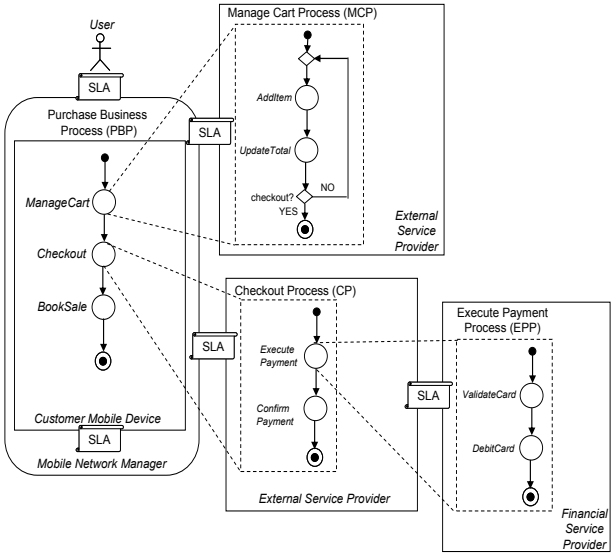

Figure 1 – A running example

The provision of PBP to a specific consumer may be regulated by a set of different SLAs, established following negotiation among the service consumer, the service providers, and the mobile network manager (as shown in Figure 1). Furthermore, service providers may have internal SLAs between the different departments of their organizations that are in charge of the provision of different components necessary for the provision of a service.

Starting from the functional description of a service and the guarantees expressed in the SLAs between it and its clients, we can create monitoring rules that during the execution of the service will be checked against events generated from it to assess whether the SLA has been violated. Examples of such rules are shown in Table 1.

R1	The average response time of the *Checkout* service as seen at the side of PBP should be less than X seconds.
R2	PBP should not allow purchases for which the total price is greater than £100
R3	EPP should always be able to decrypt a card number provided to it by one of its clients
R4	There should be at least 2 separate servers executing the instances of EPP during peak transaction hours (i.e., from 9.00am to 5.00pm)
R5	PBP should not issue a checkout request when the customer is involved in a phone call or when the remaining battery power of the handheld falls below a threshold X
R6	PBP should issue checkout requests only when the customer is in a 3G-covered cell

Table 1 – Examples of monitoring rules

The rules in SLAs may be of different types depending on: (a) the types of information that their runtime check requires and (b) the type of the property they express.

With respect to criterion (a), SLA monitoring rules are distinguished into rules that can be checked based on: (i) events captured only at the interface of services (i.e., the set of I/O messages exchanged between services and their environment) such as rule R1, (ii) information about the internal state of the service (e.g. rules R2 and R3), or (iii)

information captured from the execution environment of the service (e.g., rule R6).

With respect to criterion (b), rules can be distinguished into rules that express functional properties (e.g., R2, R5, and R6) and rules that express QoS properties (e.g., R1, R3, and R4).

Finally, for some rules it is possible to exercise pre-emptive control (i.e., block some operation or drop an inter-service message when the rule is violated) whilst in other only post-mortem control actions are possible. Rules R1 and R6 in Table 1, for example, can trigger pre-emptive control actions whilst rule R3 in the table can only by associated with post-mortem control actions.

3. Describing Monitoring Capabilities

We argue that when a monitoring framework requires information about the services internal state or execution infrastructure, such information can only be made available by services through an interface.

Services expose to the monitoring framework a set of monitoring capabilities. It should be noted that, in this context, the reference workflow can be considered as a service itself, which may expose monitoring capabilities to its own monitoring infrastructure. Monitoring capabilities can be distinguished in two basic types: *Event Emission* and *Internal Monitoring* capabilities (see Figure 2).

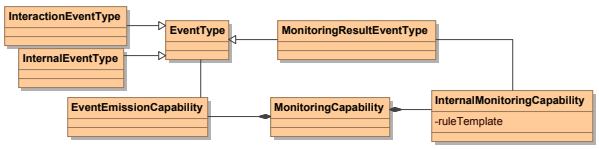

Figure 2 – Monitoring Capabilities

An *Event Emission* capability refers to the ability of a service to provide the monitoring infrastructure with basic events that may then be used for checking the violation of monitoring rules. An *Internal Monitoring* capability, on the other hand, signifies the ability of a service to monitor internally a given rule and notify violations of this rule to the monitoring infrastructure. To appreciate this distinction, consider the PBP process in Section 2 and a monitoring rule requiring the response time of the CP service to be less than N time units. CP may be able to notify to the monitoring infrastructure the timestamps of the calls and responses of its operations, thus enabling the evaluation of its response times and, consequently, the check of the monitoring rule. In this case, CP would have an event emission capability. Alternatively, CP may be able to monitor the response time monitoring rule internally (the BPEL process implementing CP, for example, could use an internal monitoring infrastructure to check the rule) and report cases where the rule does not hold to PBP. In this case CP would have an internal monitoring capability.

The events related to monitoring can also be of three different types, namely *Internal*, *Interaction* or *Monitoring Result* events.

Internal events provide information about the internal state of the execution of a service or the status of the infrastructure on which a service is being executed. In their simplest form, these may be represented by the values of variables involved in the internal execution of a service. In our example, the notification of the partial total amount made by the MCP workflow, required for monitoring rule R3, represents an internal event. When a model of the internal execution state of a service (e.g. a state transition machine or an algebraic specification [9]), is available, internal events may assume a more complex form, representing also states or state transitions. Examples of events providing information about the execution infrastructure of a service are the CPU or memory load of this infrastructure and the number of the different instances of a service that are being executed at a given time point. In the case of mobile services, such as the PBP service in our example, infrastructure events may be also provide information about the status of a mobile device. They may, for example, indicate the remaining battery power of the device or the network which the device is connected to, currently.

Interaction events provide information concerning service operation calls and responses. For atomic services, events are captured at the service container level, whereas for composed services events can be captured through the instrumentation of the workflow execution engine. Instrumentation may regard the definition of event captors which intercept and analyse SOAP messages exchanged by a service and external clients or the extraction of events from the workflow engine or service container log files.

Monitoring result events are events that represent the results of the monitoring process, i.e. violations or verifications of the satisfaction of a given monitoring rule, and can be generated only by services that have *Internal Monitoring* capabilities.

Events of different types may be transmitted from their source to their recipient under a *push* or *pull* communication policy. In *push* communication, local services and/or the reference workflow of an SBS pushes events proactively to the monitor. In *pull* communication, the monitor should periodically retrieve events from the local services or the reference workflow.

4. The Architecture of the Monitoring Framework

The general architecture of the Monitoring Framework that we propose in this paper is shown in Figure 3. The figure shows the design of the Monitoring Framework of the reference workflow, i.e., the PBP process in our example, and what instrumentation is required by this framework in the reference workflow and local services execution environment.

Local services may be implemented as a workflow (e.g., the MCP and CP services in our scenario), or they can be atomic services (e.g. the *BookSale* service) associated with local monitors and event captors. Event captors provide Interaction and Internal events captured at local services that have *Event Emission* capabilities, whereas local monitors are capable of monitor rules internally (*Internal Monitoring* capabilities). Additional capabilities, required for running the engagement protocol for establishing the monitoring process, are exported by the Capability Manager module. The

reference workflow and local services expose their monitoring capabilities through a *Monitoring Interface*. The purpose of a monitoring interface in our architecture is to standardize access to the monitoring capabilities of the reference workflow or the local services.

The proposed monitoring framework does not impose any constraints on the local monitors that are associated with different services. Thus, it allows the different services to have monitors with different implementations and property checking capabilities whilst providing a framework for deploying them together and making use of their results. Similarly, the proposed framework does not impose any restriction on the monitoring framework at the reference workflow either. The only requirement imposed by the framework is that all the monitors and event captors that are associated with the services/workflow of an SBS must adhere to a common communication interface and be described according to the capability model adopted by the framework. Thus, different monitors and event captors may be plugged in the proposed architecture as long as they are able to perform monitoring using events and the monitoring capabilities of the workflow and local services of an SBS.

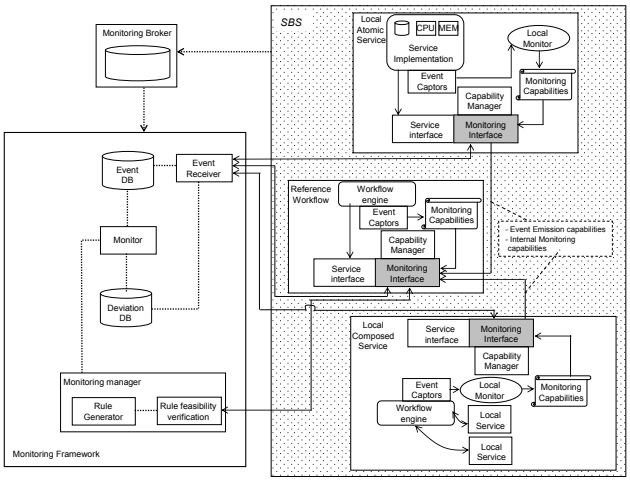

Figure 3 – The Monitoring Framework

The functionality of each module in the Monitoring Framework of the reference workflow is as described below:

Rule Generator. It automatically generates monitoring rules starting from the SLAs defined for the business process and its local services. The automated generation of monitoring rules from negotiated SLAs is out of scope in this paper.

Rule verification. This module is in charge of checking whether the generated rules can be actually monitored according to the monitoring capabilities exported by the reference workflow and the component services (rules monitorabillity). Some rules generated from the SLA may not be monitored because required events are not exposed as Event Emission monitoring capabilities by the orchestrated services. Moreover, the monitoring of a subset of rules may be delegated to local services in case exposed Internal Monitoring capabilities match a subset of the rules generated from the SLAs.

Event Receiver. This module communicates with the services' monitoring interfaces during the SBS execution. The communication may involve receiving events from Event Emission monitoring capabilities or being notified of rule violations for Rule Type monitoring capabilities (push communication policy). Events and information concerning rule violations may also be queried by the Event Receiver on monitoring capabilities (pull communication policy). Events are stored in the Event DB, whereas rule violations are directly stored in the Deviation DB.

Monitor. This is the monitoring engine, which checks monitoring rules (expressed, for instance, as Event Calculus formulas in [5, 3]) against events stored in the Event DB. When a violation of a monitoring rule is detected, this is stored in the Deviation DB.

Monitoring Broker. The services in the SBS register to this registry the end point references of their monitoring interfaces. This is required to establish the engagement protocol between the Monitoring Framework and the SBS in order to start the monitoring process. Details on such an engagement protocol are discussed later in Section 5.1.

5. The Monitoring Interface

The paradigm of service oriented computing decouples the functional description of the service interface (i.e. exposed operations and format of input and output messages), from the actual implementation of the service. Hence, the service interface represents a contract stating how external applications need to interact with a service. This approach has also been adopted in our monitoring framework, where the monitoring interface provides access service monitoring capabilities at local sites in a contract-based design approach.

The monitoring interface enriches the functional interface of a service by exposing standard monitoring-related operations that external entities can invoke in a service of an SBS. These standard operations are:

- void setLocalCapability (MonitoringCapability localCapability) – This operation allows a local service to submit a Monitoring Capability at the reference workflow monitoring interface. A capability reports the End Point Reference (EPR) of the monitoring interface at the local service;
- Capability getCapability ( ) – This operation allows the Rule Verification module in the Monitoring Framework to retrieve a capability at the reference workflow. The reference workflow is in fact in charge of assembling a global capability including also the capabilities declared by local services. Such a global capability is processed by the Rule Verification module to assess the monitorability of rules. This functionality is implemented by the reference workflow Capability Manager;
- void setMonitorableCapability(Capability monitorableCap) – This operation allows an external client, i.e. the Rule Verification or the reference workflow, to deploy the list of monitorable rules at a given monitoring interface;
- void setEventReceiverEPR(EPR er_epr) – This operation allows an external client to set the EPR of the Event Receiver. In the case of the reference workflow, this is

done by the Event Receiver, whereas the reference workflow monitoring interface acts as a client of local services' interfaces to set the EPR of the Event Receiver; and

- Event getEvent(Event e) – This operation allows the Event Receiver to pull en event from a local monitoring interface, in case the rule verification process has established the need for a pull communication policy. As per the description of monitoring capabilities (see Section 2), rule violations and interaction event events are defined as subtypes of the generic type Event, and, therefore, retrieved by the Event Receiver through this operation.

5.1 The Monitoring Engagement Protocol

The sequence diagram in Figure 4 specifies the engagement protocol between the monitoring framework, i.e. Rule Verification, Monitor, Event Receiver, and Monitoring Broker, and the SBS for establishing the monitoring process.

Initially, the local services submit their monitoring capabilities to the reference workflow interface (message 1), in order to enable the Capability Manager of the reference workflow to assemble and configure an SBS-wide monitoring capability. It should be noted that this part of the engagement may be defined recursively. For example, a composite local service may first assemble the monitoring capabilities of its internal local services and then submit it to the reference workflow. Consequently, at the level of the workflow the assembled capabilities become visible through the capability associated with the particular local service. In our earlier scenario, this would be the case with the CP and EPP workflows. More specifically, EPP will first register its capability at CP monitoring interface. CP will then assemble a capability to be registered by calling the PBP (reference workflow) monitoring interface. After having assembled the global capability, the reference workflow can register the EPRs of all involved monitoring interfaces to the Monitoring Broker (2). At this stage, the reference workflow is ready to be monitored by a generic monitor.

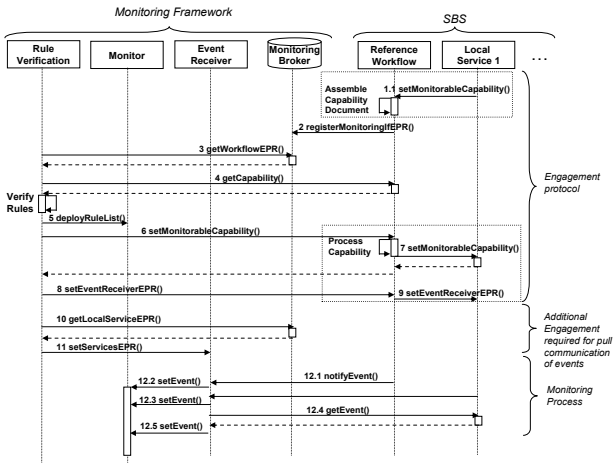

Figure 4 – Engagement Protocol

When the Monitoring Framework needs to monitor rules associated with the reference workflow and its related service landscape, the Rule Verification module retrieves the EPR of the reference workflow monitoring interface from the Monitoring Broker (3) and, consequently, the global capability from the reference workflow (4). After having verified the monitorability of rules, the Rule Verification deploys the list of monitorable rules in the Monitor (5) and to the reference workflow (6), which, in turn, processes the received document to send required capabilities to the local services (7). This last phase of the engagement protocol may also be recursive, that is, a composed local service may process the received document in order to generate required capabilities for its local services. When the list of required capabilities has been processed by the SBS, the Rule Verification sends to the reference workflows the EPR of the Event Receiver (8), which forwards this information to local services (9).

At the current stage, the engagement protocol is set up for monitoring rules that require only pull communication of events by the SBS. An additional engagement phase is required when some of the events required for monitoring need to be requested by the Event Receiver according to the pull communication policy. In this case, the Rule Verification retrieves the EPRs of the local services' monitoring interface from which the Event Receiver needs to pull events (10) and register such EPRs to the Event Receiver (11). This optional last phase ends the engagement phase, allowing the monitor process to start (12). In Fig. 5, we represent the case in which, during service execution, events are notified proactively by the SBS (12.1-3) and the case in which events are pulled from the SBS (12.4-5).

The proposed engagement protocol is built on the assumption that the Monitoring Framework should not be aware of which local services are involved in the reference workflow it is bound to monitor. This is required because local services may be provided by different organizations, they may change over time, and they can be complex, i.e. defined recursively as workflows that may orchestrate other local services. With the proposed engagement protocol, the Monitoring Framework only calls operation exposed at the reference workflow monitoring interface, but, at the same time, it can receive events (push communication) from all the local services involved in such a workflow. The Monitoring Framework requires knowledge of the EPRs of local services only in case pull communication events is needed. This is accommodated through additional engagement messages.

6. Related Work

The need for establishing clear and machine-readable SLAs between service providers and consumers has been widely recognised in industry and academia [2, 4, 8]. For what concerns runtime monitoring, intrusive monitoring relies on the instrumentation of the process or service executable code in order to perform monitoring [2]. Executable code for monitoring is interleaved with the process or service executable code and generated automatically starting from annotations made at design time. On the one hand, such approaches do not require the design

and deployment of external components dedicated to monitoring, since monitoring is executed directly by the SBS execution environment, e.g. the BPEL engine. However, on the other hand, with intrusive monitoring it becomes harder to achieve separation of concerns between the business and monitoring logic of a service. Moreover, monitoring depends on the reliability and performance of the BPEL engine, i.e. if the BPEL engine fails or becomes unavailable, also the monitoring infrastructure fails.

Non-intrusive monitoring [5, 3, 6, 1, 4] requires the establishment of mechanisms for capturing runtime information on service execution, e.g. service operation calls and responses. In this way, the business logic of the SBS process and the monitoring logic remain separate. Moreover, non-intrusive monitoring decouples the monitor infrastructure reliability from the reliability of the SBS execution environment. The monitoring infrastructure may indeed detect the failure of the BPEL engine as long as events that indicate such a failure can be captured. Non-intrusive monitoring introduces a computational overhead in the SBS execution environment, arising from the cost of capturing and communicating events between the SBS execution environment and the monitoring infrastructure. This may affect the performance of the SBS if the SBS and the monitor are executed on the same infrastructure. A different approach to non-intrusive monitoring is proposed in [10], where monitoring is delegated to service clients and regulated by an incentive mechanism that guarantees truthful reporting of monitoring information from the service provider side.

7. Conclusions and Future Work

The paper has presented an innovative approach to hierarchical and recursive monitoring of complex SBS. In particular, the proposed monitoring framework can conceptually be adopted with runtime monitors that adopt different techniques, as long as these monitor are able to process monitoring information in the format required by the monitoring interface attached to services in the SBS.

Future work concerns the implementation of the proposed framework. We plan to include in the framework several different monitors, implemented according to different techniques, in order to demonstrate the flexibility of our approach.

Furthermore, the monitoring rule verification algorithm will be designed with a twofold objective. From the monitoring framework side, the verification algorithm has the objective of detecting which rule can actually be monitored according to exposed monitoring capabilities and which policies must be satisfied in order to perform monitoring. From the service side, the analysis of the rules that can be monitored according to the exposed capabilities and policies may be exploited to build a metric regarding the monitorability of a service in an SBS. Such a metric could then be used as a further criterion to enhance common approaches to SLA-driven service discovery.

References

1. F. Barbon, P. Traverso, M. Pistore, M. Trainotti, Run-Time Monitoring of Instances and Classes of Web Service Compositions, Proc. IEEE ICWS 2006.
2. L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes, Proc. ICSOC 2005.
3. Mahbub K., Spanoudakis G.: Monitoring WS Agreements: An Event Calculus Based Approach, Test and Analysis of Service Oriented Systems, (eds) L. Baresi, E. di Nitto, Springer Verlag, 2007.
4. O. Moser, F. Rosenberg, and S. Dustdar, Non-intrusive monitoring and service adaptation for WS-BPEL, Proc. WWW 2008.
5. Spanoudakis G., Mahbub K.: Non Intrusive Monitoring of Service Based Systems, International Journal of Cooperative Information Systems, 15 (3), pp. 325-358, 2006.
6. W.M.P. Van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek, Conformance checking of Service Behavior, ACM TOIT, 8 (3), May 2008.
7. Papazoglou, M., Traverso, P., Dustdar, S., and Leymann, F. 2007. Service-Oriented Computing: State of the art and research challenges. IEEE Computer 11, 38–45.
8. Keller, A. and Ludwig, H. 2004. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management, 11(1), 57-81.
9. D. Bianculli and C. Ghezzi. Monitoring Conversational Web Services. Proc. IW-SOSWE'07 Workshop, pp. 15-21.
10. Jurca, R., Binder, W., and Faltings, B. Reliable QoS Monitoring Based on Client Feedback, in WWW2007, pp. 1003-1011.