Comuzzi, M. & Spanoudakis, G. (2010). Dynamic set-up of monitoring infrastructures for service based systems. In: S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal & C. Hung (Eds.), Proceedings of the 2010 ACM Symposium on Applied Computing. (pp. 2414-2421). ACM. ISBN 978-1-60558-639-7

Dynamic set-up of Monitoring Infrastructures for Service Based Systems

Marco Comuzzi*
Eindhoven University of Technology
School of Industrial Engineering
Eindhoven, The Netherlands
+31 40 247 2183
m.comuzzi@tue.nl

George Spanoudakis
City University London
School of Informatics
London, UK
+44 20 740 8413
g.spanoudakis@soi.city.ac.uk

ABSTRACT
Service based systems are intrinsically dynamic as the services deployed by them can be replaced at runtime. When this happens, the Service Level Agreements (SLAs) that regulate the provision of services may also need to change. Following such changes, the monitoring infrastructure that is used to monitor SLAs may also need to be modified to ensure the continuous provision of the necessary runtime checks. This paper presents a framework that supports the dynamic assessment of the monitorability of SLAs terms and the dynamic setup of an appropriate infrastructure for monitoring them following such changes. The monitorability checks are based on comparisons between the SLA terms for specific services and descriptions of the monitoring capabilities of these services which are expressed in languages introduced in the paper. The paper presents a prototype implementation of the framework and the results of a preliminary evaluation of it.

Categories and Subject Descriptors
D.2.11 [**Software Architectures**]

General Terms
Algorithms, Design.

Keywords
Run-time SLA monitoring, SLA monitorability.

1. INTRODUCTION
The paradigm of Service Oriented Computing (SOC) is changing the way of building IT-based systems. Initially, SOC was seen as a way of restructuring the IT stack within an organization in the form of services, and integrating previously non communicating systems through invocations of such services. More recently, however, SOC has evolved into a mechanism for cross-organizational service deployment, in which the systems of an organisation are realised by deploying services offered by other organisations. In this business context, services need to be provided to customers under well-defined conditions. The common approach for specifying such conditions formally is to specify and agree a Service Level Agreement (SLA) between the provider and the customer of a service. An SLA provides a formal specification of the exact conditions (functional and non-functional) under which a service should be delivered to a specific customer (or group of customers) and should be monitored at runtime to ensure that the service provision fulfils it.

Over the last few years, several approaches have been developed to support the monitoring of SLAs [1,4,5,10]. Typically, these approaches collect events during service executions and use them to check whether the properties of service provision as specified in an SLA are satisfied. Existing approaches to service monitoring provide powerful mechanisms for performing the basic checks of service compliance with SLAs but fell short of providing adequate support when replacements of the services deployed in a service based system (SBS) occur at runtime and/or the terms of the SLA under which a service is provided change dynamically. Such dynamic changes may render the monitoring mechanisms which are used to monitor the terms of an SLA no longer applicable. This can happen for different reasons. A new replacement service, for example, might not able to provide the runtime events required for monitoring some of the terms in an SLA. Also, after changes in the deployment infrastructure and composition of an existing service, it might no longer be possible to provide the events and monitors for checking the established SLAs for the service. For example, when the deployment of a service is migrated to a new web server which does not support the interception of SOAP messages sent to or by the service, it will no longer be possible to execute SLA term checks that are based on these messages.

To provide effective monitoring support when such changes happen, it is necessary: (a) to check whether the monitorability of the required SLA terms and conditions is affected by the changes, and (b) possibly modify the deployed monitoring infrastructure in order to ensure the continuous execution of the required runtime checks. Monitorability in this context is the assessment of whether particular SLA terms and conditions can be monitored given the monitoring resources (i.e., event captors and monitors) that are available in an SBS. The capabilities (a) and (b) above are not offered by existing monitoring approaches

To address this gap, in this paper we introduce a framework that we have developed to support (a) and (b), called "SLA Management for Monitoring" framework or briefly SLAM4M. The key characteristic of this framework is the separation of the actual service monitoring from the assessment of SLA monitorability, and the dynamic set up of the monitoring resources (i.e., event captors and monitors) for checking an SLA.

SAC'10, March 22-26, 2010, Sierre, Switzerland.

* This research was conducted whilst the author was at City University

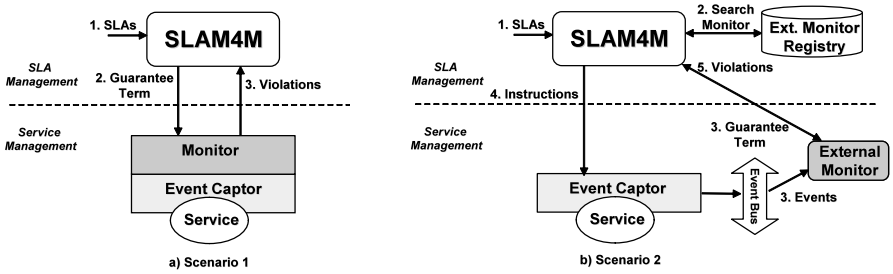

Figure 1 – Scenarios for dynamic setup of monitoring infrastructures

SLAM4M groups the activities related to monitorability assessment and the dynamic set up of monitoring infrastructure into a separate monitoring management layer and defines interfaces for integrating this layer with different monitors and event captors. The assessment of the monitorability of a given set of SLA terms by SLAM4M is based on descriptions of the monitoring capabilities of the services that are currently deployed or are going to be deployed in an SBS. These descriptions are represented according to an XML based schema that we introduce in this paper. The schema enables the specification of the event captors and monitors that a service might have and the event emission and SLA term checks that these captors and monitors offer, respectively. Furthermore, to achieve interoperability with different types of monitors and event captors, SLAM4M adopts an event-based monitoring architecture [1,4]. According to this architecture, monitoring is performed through events captured in the service execution environment by event captors. These events are sent to one or more monitors, which check the satisfaction of SLA terms based on them. In addition to the basic assessment of SLA monitorability, SLAM4M supports the dynamic setup of the service monitoring infrastructure, including the selection of appropriate event captors and monitors, the initiation of communication channels between them, and the delegation of checks of different SLA terms to individual monitors.

The rest of the paper is organized as follows. Section 2 provides an overview of our approach, by introducing the scenarios in which the dynamic setup of monitoring infrastructures is required. Section 3 introduces the languages for describing the monitoring capabilities of services and SLA guarantee terms. Section 4 introduces the design of the framework and the algorithms that it implements for the monitorability checks and the dynamic setup of monitoring infrastructures. Section 5 outlines the prototype implementation of the framework and the results of an initial evaluation of it. Finally, Section 6 overviews related work, and Section 7 provides concluding remarks and outlines current and future work.

2. OVERVIEW OF THE APPROACH

A key characteristic of the approach underpinning the design of SLAM4M is the distinction between two key layers in service provision, namely the *SLA management* and the *Service management layer (*see Figure 1). In this distinction, the SLA management layer is concerned with SLA management activities (e.g. SLA specification, negotiation, modification) and the service management layer is concerned with the software stack required for making a service manageable according to an SLA. From a monitoring perspective, the SLA management layer incorporates the mechanisms required for performing the SLA monitorability checks and the dynamic set up of monitoring infrastructures that can enable the monitoring of an SLA whilst the service management layer incorporates the Event Captors and Monitors required for service event capturing and performing the actual SLA checks, respectively. Given this distinction, SLAM4M belongs to the SLA Management layer, as shown in Figure 1.

An instance of SLAM4M can, in general, manage one or more atomic or composite services or the composition process (i.e., the "customer" of services) of an SBS. SLAM4M can interact with different event captors and monitors at the service management layer. In general, a service may have different types of event captors that are responsible for capturing and emitting different types of service events. These captors may also have different implementations. An event captor can, for instance, be realized as an instrumentation of the SOAP container or an instrumentation of the BPEL process that realizes a composite service (as in [2]). Similarly, a service may be associated with different monitors which are able to check different properties. For instance, a service may have a general purpose monitor that can check functional and non functional service properties (e.g. the generation of specific outputs for given inputs and the average service response time, respectively) as well as specialized monitors that can provide information about the infrastructure in which the service is deployed (e.g., server loads, number of running service instances etc.).

SLAM4M assumes SLAs described in WS-Agreement [7] – an XML standard schema for specifying SLAs. According to this schema, an SLA may contain a set of Guarantee Terms specifying the functional and non-functional properties that a service should provide during its deployment. The guarantee terms of an SLA may need to be monitored by service providers and customers during service execution. Monitoring at the service provider side is important in order to ensure that the provision is according to the SLA and no liability to service customers will arise as a result of deviations from it. At the service customer side, monitoring might also be important to ensure the adherence of the provider to the terms of the agreed SLA or some pre-conditions associated with them related to the service customer. For example, the agreed maximum response time for a service operation in an SLA may be guaranteed only if the number of invocations of the particular operation by the specific customer does not exceed a certain threshold per second. Thus, SLAM4M may exist both at the side of the service customer and the service provider offering monitorability checks and support for the dynamic configuration of monitoring infrastructure to either of these sides. It should also

be noted that since WS-Agreement does not provide a specific language for specifying SLA guarantee terms, SLAM4M uses a special language for defining such terms.

To support the monitorability checks and the dynamic setup of the service monitoring infrastructures, SLAM4M extracts the guarantee terms of an SLA and matches them with the known monitoring capabilities of a service [9]. These capabilities include the event reporting and the SLA checking capabilities of the service. Event reporting capabilities describe the types of events that can be provided by the event captor(s) associated with the service. Examples of events types required for monitoring include time stamped service operation calls and responses or records of time stamped values of internal process variables for composite services realized by service composition processes. The SLA checking capabilities of a service are provided by the monitor(s) associated with it. These capabilities are represented by the list of SLA guarantee terms specification languages that the monitors of a service support. A monitor is said to support an SLA guarantee term specification language if it can directly monitor terms expressed in this language or it incorporates a mechanism for translating terms expressed in this language into some internal operational monitoring specification. In our prototype, for example, we have used the monitors presented in [4] and [1] which use monitoring specifications expressed as Event Calculus and RTML rules, respectively.

Hence, at the SLA Management layer, SLAM4M processes SLAs in order to extract their Guarantee Terms, and orchestrates the dynamic setup of the service monitoring infrastructure. To set up a service monitoring infrastructure, SLAM4M retrieves the capabilities of the Event Captor of the managed service and the local and external Monitor engines. On the basis of such capabilities, SLAM4M decides whether an SLA Guarantee Term that is defined for a service can be monitored and, if it can, whether the term will be checked by a local (Scenario 1) or an external service Monitor (Scenario 2). In the latter case, SLAM4M starts the engagement protocol between the local Event Captor of the service and the External Monitor.

Figure 1 shows the two scenarios for dynamic service monitoring setup in SLAM4M. In the first scenario (see Figure 1a), the managed service is provided with both Event Captor(s) and a local Monitor and has, therefore, both event reporting and SLA checking capabilities. Thus, when it receives an SLA, SLAM4M checks if each guarantee term in it can be monitored locally, according to the capabilities exposed by the Event Captor and the Monitor. In particular, in order for a Guarantee Term to be locally monitored, the Event Captor should be able to provide the required events, while the Monitor should support the language used for expressing the Guarantee Term. The second scenario (see Figure 1b) applies to the following two cases:

(i) The Event captor provides the events required for monitoring a Guarantee Term, but the Monitor does not support the Guarantee Term language;
(ii) The managed service has only an Event Captor but no associated local Monitor.

In the second scenario, SLAM4M first assesses if the required events are available from the local event captor of the service and then tries to identify an external monitor that can support the Guarantee Term language. This identification takes place through a monitor registry that is accessible to SLAM4M and, if an appropriate external monitor can be found, SLAM4M submits the guarantee term to the external monitor and instructs the event captor of the service to provide events to this monitor. It should be noted that the external monitor may be available at some URI on the network and, therefore, an engagement protocol and an event communication infrastructure are required for establishing and realizing the communication of events between the service event captor and the external monitor.

In the prototype implementation that we discuss in Section 5, we use a "publish/subscribe" event communication infrastructure designated as "Event Bus" in Figure 1b. More specifically, after locating a Monitor, SLAM4M gets from it a token designating an event channel of interest and uses this token to subscribe the monitor to the Event Bus. The same token is passed to the Event Captor to be used when it publishes events to the bus so that these events can be forwarded to the appropriate monitor.

3. LANGUAGES FOR THE SLAM4M FRAMEWORK

To support the assessment of SLA monitorability and the dynamic setup of service monitoring infrastructures, we developed XML schemas for describing: (i) event types, (ii) types of SLA Guarantee Terms, and (iii) monitoring capabilities.

The XML schema for describing Event Types is used to specify the required event types for monitoring a Guarantee Term type and the available event types in the Monitoring capability of an Event Captor. This schema is shown in Figure 2a. According to it, an Event Type is described by a context (*ETContext*) and a list of required fields (*ReqFieldList*). The context is described by a name (*ETname*), a unique identifier (*ETid*), a textual description (*ETDescription*), and the URI of the XML schema for describing event instances (*EventSchemaURI*). Each event type may be expressed according to a different schema. Each required field is then described by a name, a type (*FieldClass*), and a value. Events representing time stamped service operation calls and responses, for example, are required for monitoring a guarantee term about the completion time of a synchronous service operation. The fields required for describing a time stamped web service operation call are the service name, operation name, and timestamp. Besides these, a time stamped operation response requires also a field reporting the identifier (*Id*) of the matching operation call.

The XML schema for describing Guarantee Term Types is shown in Figure 2b. According to it, a Guarantee Term type is described by a name (*GTermTypeName*), a unique identifier (*GTermTypeId*), and a reference to the URI at which the guarantee term type specification schema is available (*GTermTypeLanguage*). A guarantee term type is also described by a list of instantiation fields (*InstantiationFieldList*), a *Qualifying Condition*, and a *Service Level Objective*. The instantiation fields contain information required for instantiating a guarantee term type into an actual guarantee term in an SLA. The value of the attribute *target*, in particular, is set to "qualCondition" or "slo" for instantiating the Qualifying Condition and the Service Level Objective (*SLO*) in an SLA, respectively (according to WS-Agreement, a guarantee term is specified as a condition of the type: IF Qualifying Condition THEN Service Level Objective). Qualifying conditions and service level objectives are expressed as binary predicates in the current implementation.

Figure 2 – Schemas for describing (a) Event Types and (b) types of Guarantee Terms

Finally, the definition of a guarantee term type includes a list of the event types required for monitoring it. As discussed above, event types are described by a list of required fields. In the specification of guarantee term types, the fields in required event types do not have a value. The value for these fields is specified in the actual guarantee term in an SLA as we show later. In particular, an SLA specifies the value of the fields for which the attribute *forInstantiation* is true in the event type description. Field values are used by SLAM4M to match the guarantee term type description with the event reporting capability of the event captor of a service (see Section 5 for examples).

```
    <wsag:GuaranteeTerm
wsag:Name="CompletionTime_GetProductInformation_Gterm"
    wsag:Obligated="ServiceProvider"xmlns:terms=http://completion-time-uri.org>
     <terms:GTermTypeName="CompletionTime"/>
    <terms:GTermTypeId="556678"/>
     <terms:InstantiationFieldList>
      <terms:GTermField target=”eventInstantiation”>
            <terms:FieldName>serviceName</terms:FieldName>
            <terms:FieldValue>InventoryService</terms:FieldValue>
        </terms:GTermField>
        <terms:GTermField target=”eventInstantiation”>
            <terms:FieldName>operationName</terms:FieldName>
            <terms:FieldValue>GetProductInformation</terms:FieldValue>
       </terms:GTermField>
        <terms:GTermField target=”slo”>
            <terms:FieldName>threshold</terms:FieldName>
            <terms:FieldValue>20</terms:FieldValue>
       </terms:GTermField>
       <terms:GTermField target=”slo”>
            <terms:FieldName>percentile</terms:FieldName>
            <terms:FieldValue>0.95</terms:FieldValue>
        </terms:GTermField>
       </terms:InstantiationFieldList>
</wsag:GuaranteeTerm>
```

Figure 3 – Example of an SLA Guarantee Term

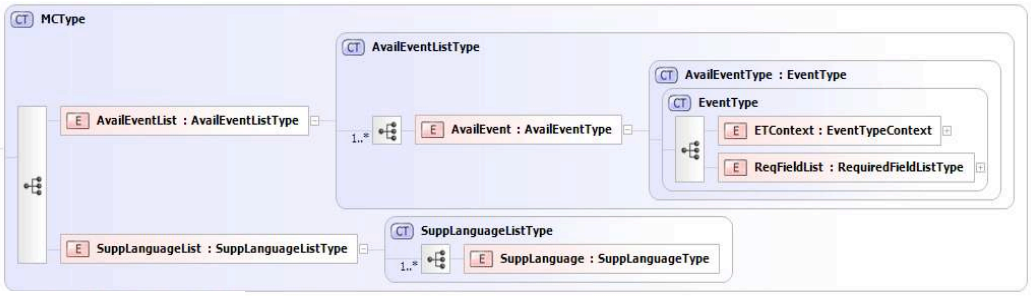

Figure 4 – Monitoring Capability Type (MCType) Description schema

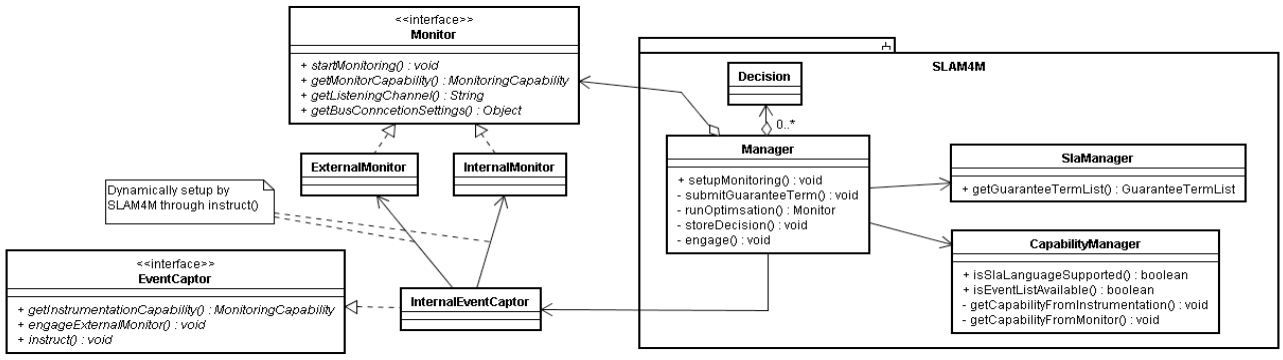

Figure 5 – The architecture of prototype

Figure 3 shows an example of guarantee term in an actual SLA. This example is a fragment of a SLA expressed in WS-Agreement that is used in our prototype implementation. In this example, the namespace *wsag* refers to the WS-Agreement specification while the namespace *terms* refers to the language for expressing guarantee term types. In the example, the guarantee term type refers to the average completion time of a generic service operation call. A Guarantee Term in an SLA can be defined only in terms of the name and id of the corresponding Guarantee Term Type, the fields needed for the instantiation of events, the qualifying condition, and the service level objective. In our example, the events are instantiated with the name of the service and operation which the guarantee term refers to. Also, the guarantee term involves one service level objective about the average completion time of operation invocation (this time should be less than 20ms in 95% of invocations). Using the guarantee term id, SLAM4M can retrieve the guarantee term type description and, then by using the instantiation fields, it can obtain the actual list of events required for monitoring the term.

The XML Schema for describing monitoring capabilities is shown in Figure 4. The schema allows the description of a list of available Event Types and a list of supported Guarantee Term Types languages. The former is used to describe the capabilities of Event Captors, and the latter is used for the description of the capabilities of service monitors.

4. THE FRAMEWORK FOR SLA MONITORING MANAGEMENT

4.1 Overall Design

The architecture of our prototype is shown in Figure 5. Before discussing the details of the modules within SLAM4M, we introduce the interfaces that must be implemented by the event captor(s) and the monitor(s) of a managed service to allow the dynamic setup of the service monitoring infrastructure.

More specifically, event captors expose the following methods:

- *getInstrumentationCapability()*: This method is called by SLAM4M to retrieve the monitoring capability of the Event Captor;
- *instruct():* This method is called by SLAM4M to set the end point reference of the Monitor to which events will need to be sent. In case of local monitors, the Event Captor can directly start sending events at the Monitor end point reference; in case of external monitors, the end point reference is required by the Event Captor to start the engagement of the external monitor;
- *engageExternalMonitor():*This method is called to obtain the settings for connecting to the Event Bus and the name of the channel to which send events during the engagement of an External Monitor.

Also, monitors expose the following methods:

- *getMonitorCapability()*: This method is called by SLAM4M to retrieve the Monitor capability;

- *submitGuaranteeTerm():*This method is called by SLAM4M to submit a Guarantee Term to be monitored; the invocation of the monitor triggers the generation of monitoring properties by the monitor engine, i.e. Event Calculus or RTML rules in our prototype implementation;
- *getConnectionSettings():*This method is called by event captors to get the settings for connecting to the Event Bus during the engagement protocol;
- *getListeningChannel():*This method is called by event captors during the engagement protocol to get the name of the channel on which to send events.

SLAM4M and its internal modules are shown on the right part of Figure 5. The Manager represents the outer layer of SLAM4M and exposes the operation *setupMonitoring()*. This operation is invoked by external components to dynamically setup the monitoring infrastructure and receives an SLA expressed in WS-Agreement as argument. The Manager stores also the decision taken during the setup of the monitoring infrastructure, for each Guarantee Term in an SLA. For a specific SLA Guarantee Term, this decision can be that: (a) the term can be monitored and its monitoring is delegated to the local monitor of the service that should adhere to the term, (b) the term can be monitored and its monitoring is delegated to an external monitor, or (c) the term cannot be monitored. The SLAManager exposes functionality for parsing SLAs expressed in WS-Agreement and extracting their guarantee terms. The CapabilityManager retrieves the capabilities of service monitors and event captors and the description of guarantee term types. The monitoring capabilities are matched to detect whether the list of required events to monitor a guarantee term is available from a given event captor and whether the guarantee term language is supported by a given monitor. Note that the list of required events for a guarantee term is derived according to the instantiation of required fields described in Section 3.

4.2 Algorithm for dynamic monitoring setup

The algorithm for checking the monitorability of SLA terms and setting up a service monitoring infrastructure for a given SLA is shown in Figure 6 (the algorithm constitutes the internal implementation of the operation *startMonitoring()* of SLAM4M Manager in Figure 5).

1. Extract the list of Guarantee Terms `gTermList` from the SLA
2. Get the capabilities of the monitor (`languageList`) and event captor (`availEventList`) of the service
3. FOR EACH Guarantee Term `gTerm` (`gTerm` ∈ `gTermList`)
   3.1. Get the list of required Events (`requiredEventTypeList`) of the term
   3.2. Get the required language (`GTermTypeLanguage`) of the term
   3.3. IF `requiredEventTypeList` ⊆ `availEventList` AND `GTermTypeLanguage` ∈ `languageList`
      3.3.1. Select monitor for the term and submit `gTerm` to it
      3.3.2. Instruct Event Captor with reference to Monitor
      3.3.3. Record that `gTerm` is delegated to `INT_MONITOR`
   3.4. IF `requiredEventTypeList` ⊆ `availEventList` AND `GTermTypeLanguage` ∉ `languageList`
      3.4.1. Look for suitable External Monitor and submit gTerm
      3.4.2. IF External Monitor NOT found
         3.4.2.1. Record that `gTerm` is not monitorable
      3.4.2. ELSE Run the engagement protocol between instrumentation and External Monitor
      3.4.3. Record that `gTerm` is delegated to `EXT_MONITOR`
   3.5. IF `requiredEventTypeList` ⊄ `availEventList`
      3.5.1. Record that `gTerm` is not monitorable

Figure 6 –Monitoring infrastructure set up algorithm

The algorithm gets as input an SLA and the service that it refers to and outputs a list of decisions on how the different guarantee terms of the SLA will be monitored. The algorithm first extracts the *Guarantee Terms* of the SLA, through the *SLAManager*, and then retrieves the capabilities of the service event captor and monitor (i.e. the list of available events and the list of supported Guarantee Term languages, respectively) through the *CapabilityManager* (lines 1-2). Then, for each Guarantee Term, it first gets the required event list and the required language by matching the Guarantee Term type description with the actual Guarantee Term extracted from the SLA (3.1–3.2). Then, the algorithm assesses whether the monitoring of the term can be performed locally (Case 1, line 3.3) or delegated to a suitable external monitor (Case 2, line 3.4). If the local event captor does not provide the required event, then the monitoring of the term is not possible (line 3.5).

In Case 1, the SLAM4M Manager can choose to delegate the monitoring of the term to the local monitor or search for a suitable external monitor. This decision can be taken by a running a selection algorithm (line 3.3.1), which determines the best suitable (local or external) monitor (this corresponds to the operation *runOptimsation()* of the Manager). In the current implementation, this selection always returns the local monitor. In future implementations, however, we are planning to investigate and implement more sophisticated selection methods taking into account factors such as the current workloads of different monitors, their efficiency for the given term type or their trustworthiness (e.g. monitor reliability, availability etc).

In Case 2, the Manager looks for a suitable external monitor that could support the language. The *searchExternalMonitor()* method performs an exhaustive search of the monitors that are listed in the external monitor registry until a monitor that supports the language is found. If a suitable monitor cannot be found, then the guarantee term monitoring is recorded as non monitorable. When a suitable external monitor is found, the Manager triggers the engagement protocol between the local event captor and the external monitor by calling the *engageExternalMonitor()* method exposed by the event captor. The protocol realized by this method is shown in Figure 7. More specifically, the event captor sets a reference to the external monitor and then, it retrieves the settings for connecting to the Event Bus from this monitor and the unique name of the channel (token) on which the external monitor will be listening for events. Finally, the event captor connects to the Event Bus and publishes events to the channel. Before sending the channel name to the event captor, the external monitor registers the new channel on the Event Bus.

1. Set a reference to the External Monitor
2. Get Event Bus connection settings from External Monitor
3. Get channel name `channel` from External Monitor
4. Connect to Event Bus and subscribe to channel `channel`

Figure 7 – Engagement protocol

Figure 8 shows examples of: (i) a required event in the list of required events in a guarantee term type description (e.g. the time stamped service operation call), (ii) an event reporting capability (i.e., an available event representing time stamped calls of the operation *GetProductInformation* of the service *InventoryService*), and (iii) a monitoring capability. Note that the available event type can be positively matched against the required events in the guarantee term type only after the latter has been instantiated with the values of *serviceName* and *operationName* fields as in the actual guarantee term shown in

Figure 3. Also, the monitoring capability can be positively matched with the language for expressing the guarantee term type declared in the guarantee term of Figure 3.

5. IMPLEMENTATION AND EVALUATION

A prototype of our framework has been implemented in Java, using an instance of the EVEREST toolkit for service monitoring presented in [4] as the local Monitor. The list of external monitors includes additional instances of the EVEREST toolkit as well as instances of the ASTRO monitoring engine [1]. Both these monitors have been already used and evaluated in the context of event-based service and composite service process monitoring. To become compliant with SLAM4M, EVEREST and ASTRO have been extended so as to provide implementations of the Monitor and Event Captor interfaces described in Section 4.1. Our prototype uses also the open source implementation *wsag4j*[1] (WS-Agreement for Java) to parse WS-Agreement SLAs and the open source Openfire[2] implementation of the XMPP PubSub[3] specifications to realise its Event Bus.

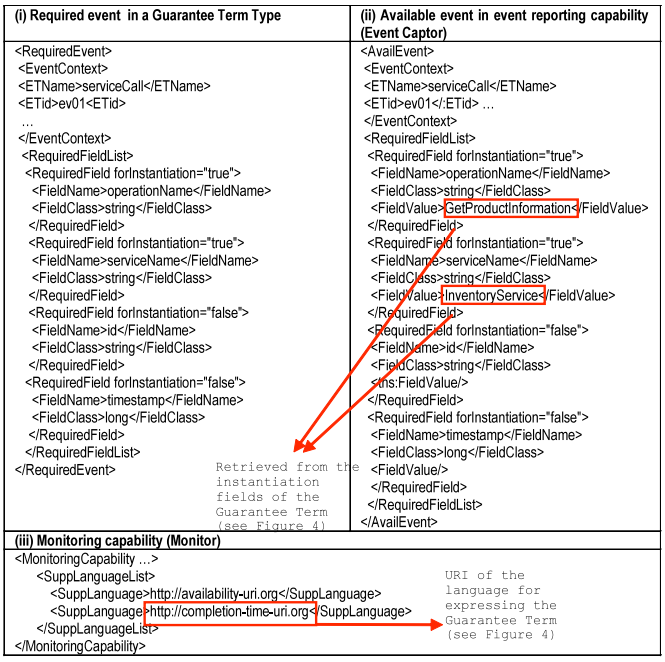

Figure 8 – Examples of Guarantee Term Type and capabilities specification

For a preliminary experimentation with the prototype, we used a retail SBS for selling products in a supermarket. This SBS involves an atomic service, for inventory management, and a composed service (BPEL process) for managing the payment of purchased goods. In the tests, SLAM4M run on an Intel Core Duo (2.33GHz) machine with 2GB of RAM. We run some experimental tests using sample WS-Agreement SLAs for the inventory and the payment service having two Guarantee Terms each. To test the ability of SLAM4M to deal with different monitor instances, in our experimental setup we used an instance of EVEREST as the local monitor for the Guarantee Terms in the inventory service SLA and an instance of the ASTRO toolkit as external monitor for guarantee terms of the payment service. In the experiments, the external monitor registry was populated with 10 different monitor instances (5 instances of EVEREST, 5 instances of ASTRO).

Table 1 shows the average completion time of the whole monitoring infrastructure setup process and the main individual phases of the algorithm in Figure 7. For each of the two cases handled by the algorithm in Section 4.2 (i.e., the selection of local and external monitors), the average completion time has been computed over 300 different invocations of the *setupMonitoring()* method in the SLAM4M Manager. On average, the whole process starting from the submission of the SLA to SLAM4M to the creation of monitorable rules in the monitor and the engagement of event captors, took from 0.9 to 2.9 seconds. The time required for setting up the monitoring infrastructure, i.e., searching for a Monitor and engaging (if necessary) the external Monitor, is comparable with the time required by the Monitor to generate Event Calculus or RTML rules, i.e. the time required to execute the *submitGuaranteeTerm()* method. This demonstrates that the overhead introduced by the dynamic setup of the monitoring infrastructure is in the same order of magnitude as the time required for setting up the monitor statically (i.e., the time required by a monitor to create monitoring rules from the terms of the SLA). Moreover, our preliminary tests show that future work should investigate methods for reducing the time required for searching and engaging the external monitor, which now represents the largest share of the overhead introduced by the dynamic setup of the monitoring infrastructure.

Table 1 – Average Times for processing a Guarantee Term

Avg. Execution Time	*EVEREST toolkit (local monitor)*	*ASTRO toolkit (selected external monitor)*
setupMonitoring() [whole process]	0.91s	2.9s
searchExternalMonitor()	Not required	0.47s
engageExternalMonitor()	Not required	1.4s
runOptimsation()	0.01s	Not required
Instruct()	0.05s	Not required [internal to engageExternalMonitor()]
submitGuaranteeTerm() [generation of monitoring rules]	0.85s (EVEREST rules)	0.92s (RTML rules)

6. RELATED WORK

Work on runtime monitoring of service based systems has developed different types of monitors. These monitors realise either intrusive or event-based monitoring.

Intrusive monitoring relies on weaving the execution of monitoring activities at runtime within the code that realises the service itself or the orchestration process of an SBS. In the case of composite services, this can be done directly in the BPEL engine, by interleaving monitoring code with the process executable code as in [2,8,10,13]. The assessment of the monitorability of service properties required by SLAs can not be easily achieved through this paradigm, since the properties to be monitored and the actions required for monitoring must be interleaved with service execution code and, therefore, known a priori by the system designer. Event-based (aka non-intrusive) monitoring [4,5,11] 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 and the monitoring logic remain separate. The approaches cited above for

[1] http://packcs-e0.scai.fraunhofer.de/wsag4j/index.html
[2] http://www.igniterealtime.org/projects/openfire/index.jsp
[3] http://xmpp.org/extensions/xep-0060.html

non-intrusive monitoring, however, take for granted the availability of events required for monitoring and cannot cope with dynamic changes in SLAs and/or the constitution of an SBS. The work in [12] presents an approach to SBS monitoring based on diagnosis models, where the behaviour of the SBS is checked at runtime in order to discover the reasons of faults that may occur in the system. Even in this case, however, the service diagnosis infrastructure is statically defined and cannot be modified dynamically during service execution.

Several projects have also focused on SLA definition, establishment, and provisioning in the context of both Web and Grid services. Adaptive Services Grid (ASG), for example, has designed architecture for establishing and monitoring SLAs in Grid environments [14]. In this architecture, the monitoring rules and parameters as well as the architecture for SLA monitoring are statically defined and cannot be updated at runtime. The TrustCOM project has also produced a reference implementation for SLA establishment and monitoring [15]. This implementation, however, does not involve the dynamic setup of monitoring infrastructures. The SLA Monitoring and Evaluation architecture presented within the Gridipedia project [16] has several similarities with the approach presented in this paper, such as the need to separate SLA from service management and the adoption of a publish-subscribe infrastructure for connecting managed services to remote monitors. The binding between services and monitors, however, is statically defined and cannot be established or altered dynamically. Moreover, the dynamic assessment of SLA monitorability is not supported.

The novelty of SLAM4M with respect to the above approaches is the introduction of a distributed monitoring architecture that can support the dynamic execution of monitorability checks and setup of monitoring infrastructures that may incorporate different types of monitors and event captors. A preliminary investigation of the specification of service monitoring capabilities has been presented by the authors in [9].

7. CONCLUDING REMARKS

In this paper, we have introduced a framework supporting the dynamic assessment of the monitorability of the terms in a given SLA following changes in the services deployed by an service based system and the dynamic establishment of monitoring infrastructures to support SLA monitoring. A prototype implementation of this framework has been developed and presented in the paper along with a preliminary evaluation of the framework using the implemented prototype. Whilst in the paper, we have assumed that monitorability assessment and the dynamic set up of monitoring infrastructures supported by the framework occur at the same time, in principle the two activities can be separated and SLA monitorability could be assessed in conjunction with service discovery and/or SLA negotiation.

Currently, we are developing an algorithm to select the best available monitor in cases where more than one monitor that can support the monitoring of SLA guarantee terms can be found whilst setting up monitoring infrastructures. We are also planning a further evaluation of our approach, focusing on its scalability with respect to the size of the SLAs that need to be monitored (number of guarantee terms) and the number of available monitors. Finally, we are extending our framework to support scenarios of SBSs organised in complex hierarchies of business, software and infrastructure services with hierarchical SLAs.

8. ACKNOWLEDGMENTS

We express our thanks to Michele Trainotti and Miguel Angel Rojas Gonzalez for their support on the ASTRO toolkit and the wsag4j library, respectively. This research has been supported by the EU Commission under the Framework 7 project SLA@SOI (grant n. 216556).

9. 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] O. Moser, F. Rosenberg, and S. Dustdar, Non-intrusive monitoring and service adaptation for WS-BPEL, Proc. WWW 2008.

[4] G. Spanoudakis, K.Mahbub, Non Intrusive Monitoring of Service Based Systems, Int. J. of Cooperative Information Systems, 15(3):325–358, 2006.

[5] 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.

[6] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, Service-Oriented Computing: State of the art and research challenges. IEEE Computer, 11:38–45, 2007.

[7] Web Services Agreement Specification (WS-Agreement), http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf.

[8] D. Bianculli and C. Ghezzi. Monitoring Conversational Web Services. Proc. IW-SOSWE'07, 15–21, 2007.

[9] M. Comuzzi and G. Spanoudakis, Describing and Verifying Monitoring capabilities for SLA-driven Service-Based Systems, Proc. CAiSE Forum 2009.

[10] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea and P. Spoletini. Validation of web service compositions. IET Software, 1(6):219–232, 2007.

[11] K. Mahbub and G. Spanoudakis, Run-time Monitoring of Requirements for Systems Composed of Web Services: Initial Implementation and Evaluation Experience, Proc. of ICWS 2005, 257–265, 2005.

[12] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan, Fault Tolerant Web Service Orchestration by Means of Diagnosis, 3rd Eur. Work. on Software Architecture, 2006.

[13] A. Lazovik, M. Aiello and M. Papazoglou, "Planning and monitoring the execution of web service requests", Int. J. of Digital Libraries 2006

[14] K. Jank, Reference Architecture. Adaptive Services Grid Deliverable D6.V-1, 2005.

[15] The TrustCOM project. Deliverable 64: Final TrustCoM Reference implementation and associated tools and user manual. June 2007 (v3.0).

[16] Gridipedia, SLA Monitoring and Evaluation Technology Solution http://www.gridipedia.eu/sla-monitoring-evaluation.html