Advanced Service Monitoring Configurations with SLA Decomposition and Selection

Howard Foster and George Spanoudakis
Department of Computing, City University London,
Northampton Square, EC1V 0HB, London,
England, United Kingdom
{howard.foster.1,g.e.spanoudakis}@city.ac.uk

ABSTRACT

Service Level Agreements (SLAs) for Software Services aim to clearly identify the service level commitments established between service requesters and providers. The commitments that are agreed however can be expressed in complex notations through a combination of expressions that need to evaluated and monitored efficiently. The dynamic allocation of the responsibility for monitoring SLAs (and often different parts within them) to different monitoring components is necessary as both SLAs and the components available for monitoring them may change dynamically during the operation of a service based system. In this paper we discuss an approach to supporting this dynamic configuration, and in particular, how SLAs expressed in higher-level notations can be efficiently decomposed and appropriate monitoring components dynamically allocated for each part of the agreements. The approach is illustrated with mechanical support in the form of a configuration service which can be incorporated into SLA-based service monitoring infrastructures.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Service-oriented Architecture (SOA)

General Terms

Algorithms, Design, Management, Reliability

Keywords

Service Level Agreements, Service Composition, Monitoring

1. INTRODUCTION

As a key part of monitoring and management, systems developed with the Service-Oriented Architecture (SOA) design pattern should utilise negotiated agreements between service providers and requesters. Typically the result of this negotiation is specified in Service-Level Agreements (SLAs), which are then used to monitor levels of service provided and to optionally specify pre-conditions and actions to take in the event of such levels being violated. The dynamic availability and allocation of the responsibility for monitoring SLAs (and often different parts within them) to different monitoring components is necessary as both SLAs and the components available for monitoring them may change dynamically during the operation of a service based system [4]. The complexity of SLA terms however means that several monitoring components may need to be selected for a single SLA guaranteed term expression (e.g. availability $> 90\%$) since each part of the expression may be reasoned by a physically different provider. Existing work has shown examples of decomposition based upon simple decomposition of expressions [4] but there is also need to consider how variations between different monitors (e.g. trustworthiness or access constraints) can be considered in a dynamic monitoring configuration process.

In this work we show how complex service agreement terms can be decomposed in to manageable monitoring configurations, whilst also including a mechanism to support preferred monitoring component selection requirements. Advanced configuration is supported by a *MonitoringManager* component which mechanically parses an SLA, generates a formal Abstract Syntax Tree (AST) and decomposes the terms of the AST in to expressions for monitoring. Each expression is then used to select appropriate reasoning or service sensor monitoring components. The main contribution of this work is that both the monitoring configurator and the monitoring configuration specification are generic, reusable artifacts, that can be incorporated in to other monitoring frameworks where configuration of monitoring components is required. In the case of the configurator, it is already offered as a reusable service using standard web service protocols and enables the use of replaceable selection criteria for candidate monitors which can be driven from preferences for monitor provider and offered features.

The paper is structured as follows. In section 2 we provide a brief background and related work section discussing how work has progressed on service monitoring frameworks and configuration. In section 3 we present an example monitoring architecture whilst in section 4 we describe the overall approach to monitoring configuration. In section 5 we discuss in detail the parsing and decomposition of SLA specifications and in section 6 we discuss the selection and specification of a monitoring system configuration. In section 7 we briefly discuss an application of the configuration module and in section 8 conclude the paper with a discussion of

SAC'11 21-MAR-2011, TaiChung, Taiwan

present and future work.

2. BACKGROUND AND RELATED WORK

Background and related work in this paper falls within two areas. First, we consider the definition and translation of SLAs and second the runtime monitoring of service based systems based upon monitoring features. Several projects have focused on SLA definitions and provisioning in the context of both Web and Grid services. The Adaptive Services Grid (ASG) project for example, has designed an architecture for establishing and monitoring SLAs in Grid environments [6]. 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 [5] has several similarities with the approach presented in this paper, such as the need to separate SLA from service management. Their focus of work, however, is on statically binding services and monitors, whilst ours is on dynamically allocating monitors to SLA parts, based upon matching the exact terms that need to be monitored and the monitoring capabilities available in different services. For SLA translation, in [17, 9] the authors describe decomposing an SLA of resource requirements (with the purpose of building a system which represents the SLA required). Their approach is more focused on building a system rather than monitoring existing services. However, it also employs techniques to optimise and arrange efficient configurations based upon the SLA expressions stated. In [10], the authors consider evaluating expressions for conditions of properties of services (e.g. response time), however their SLA format appears to offer only single assertions rather than complex expressions.

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. In the case of composite services, this can be done directly in a process engine, by interleaving monitoring code with the process executable code as in [2, 3, 1, 7]. The assessment of monitoring 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 [13, 16, 8] 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 and the monitoring logic remain separate. The approaches cited above for non-intrusive monitoring, however, take for granted the availability of events required for monitoring and cannot cope with dynamic changes in SLAs.

The work described in this paper extends an existing approach on dynamic generation of monitoring system configurations [4]. Specifically, we consider individual guarantee terms within an SLA by decomposition of complex guarantee expressions, a wider spectrum of monitoring components (e.g. effectors) and support complex monitoring configurations that can engage different monitoring components for checking the same SLA term if necessary.

3. ARCHITECTURE

The overall architecture for our Monitoring System is illustrated in Figure 1. The Planning and Optimization Component (POC) is a local executive controller for a ServiceManager. It is responsible for assessing and customizing SLA offers, evaluating available service implementations and planning optimal service provisioning and monitoring strategies. The POC generates a suitable execution plan for monitoring (based upon a configuration obtained from the MonitoringManager component) and passes this to the Provisioning and Adjustment Component (PAC). The PAC collects information from the Low Level Monitoring System, analyzes the incoming events and decides if a problem has occurred or it is about to occur, identifies the root cause and if possible decides and triggers the best corrective or proactive action. In case the problem cannot be solved at a local level, PAC escalates the issue to a higher level component, namely the POC. In case of an SLA violation, Adjustment can trigger re-planning, re-configuration and/or alerting to higher-level SLA. These capabilities are considered to be important in order to guarantee best user perception preserving underlining resources.

The MonitoringManager (MM) coordinates the generation of a monitoring configuration of the system. It decides, for an SLA specification instance it receives, which is the most appropriate monitoring configuration according to configurable selection criteria. A monitoring configuration describes which components to configure and how their configurations can be used to obtain results of monitoring Guarantee States. The Low Level Monitoring Manager is a central entity for storing and processing monitoring data. It collects raw observations, processes them, computes derived metrics, evaluates the rules, stores the history and offers all this data to other components (accessible through the ServiceManager). It implements the monitoring part of a ProvisioningRequest, containing constraint based rules (time and data driven evaluations) and ServiceInstance specific Sensor related configurations. It is general by design, so capable of supporting Infrastructure, Software and Services and other use cases.

There are three types of Monitoring Features in the Monitoring System. First, *Sensors* collect information from a service instance. Their designs and implementations are very much domain-specific. A sensor can be injected into the service instance, e.g., service instrumentation, or it can be outside the service instance intercepting service operation invocations. A sensor can send the collected information to the communication infrastructure or other components can request (query) information from it. There can be many kinds of sensors, depending on the kind of information they want to collect, but all of them should implement a common interface. The interface provides methods for starting, stopping, and configuring a sensor. Second, *Effectors* are components for configuring service instance behaviour. Their designs and implementations are very much domain-specific. An effector can be injected into a service instance, e.g., service instrumentation, or can interface a service configuration interface. There can be many kinds of effectors, depending on the service instance to be controlled, but all of

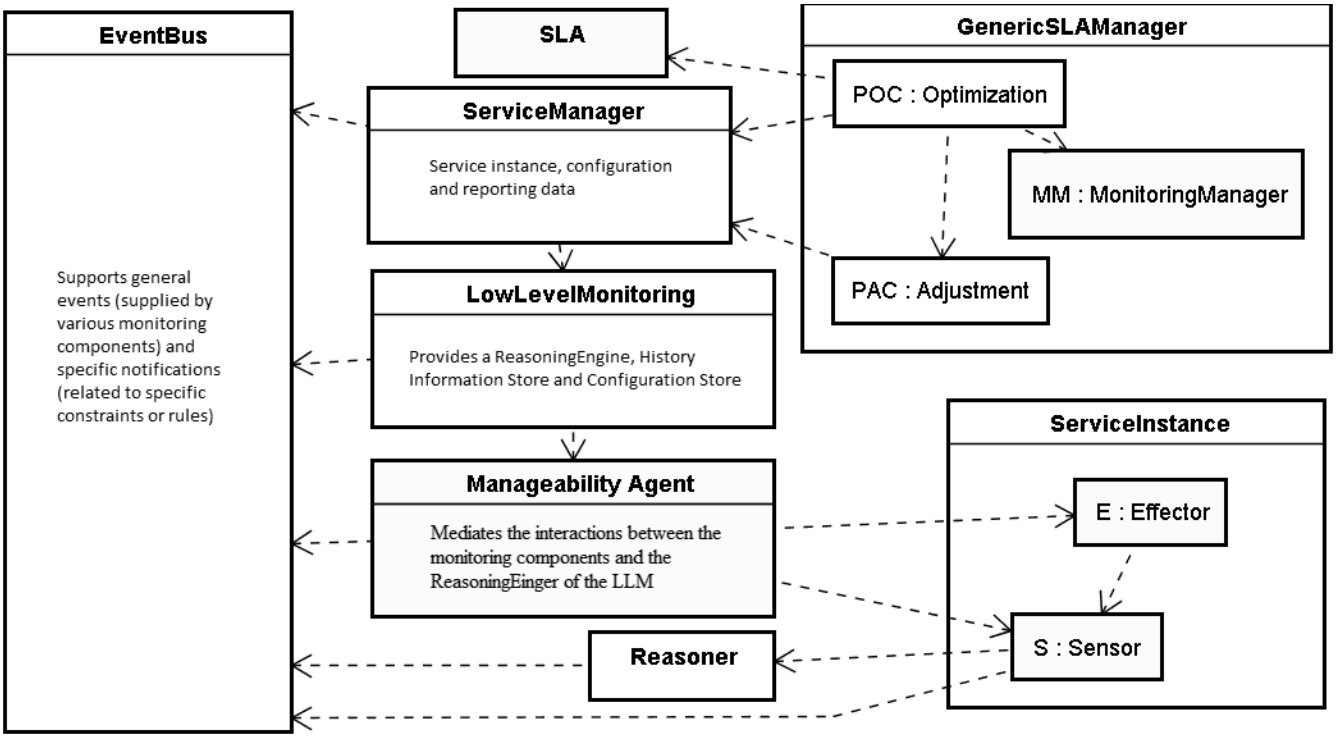

Figure 1: A Core Monitoring Architecture

them should implement a common interface. The interface should provide methods for configuring a service. The third type of monitoring feature is a *Reasoning Component Gateway (RCG)*. An RCG provides the interface for accessing a Reasoning Engine. A Reasoning Engine (or short name as Reasoner) performs a computation based upon a series of inputs provided by the events or messages sent from a sensor or an effector. An example RCG may provide a function to *compute the average completion time* of service requests. In this case the RCG accepts events from sensors detecting both request and responses to a service operation. RCGs also provide access to generic runtime monitoring frameworks such as EVEREST [12].

3.1 SLA Specification

The main input to the monitoring system is an instance of an SLA specification. In this work we reuse the model as defined in the European Union SLA@SOI project [11]. This SLA specification provides an abstract syntax for expressing the content of SLA agreements. The specification describes:

- The various parties to the agreement, in particular the service provider and consumer,
- The functional properties of offered services - i.e. service descriptions (operations etc).
- The terms of one or more agreements - i.e. the various obligations that parties commit to. These obligations are encoded in the form of guarantees, which come in two basic flavours:
- Guaranteed states; expressed as constraints over non-functional (QoS) service properties, and
- Guaranteed actions; concerning, for example, requirements for reporting SLA status, or for the payment of penalties in case of SLA violation.

A Standard Terms vocabulary (of which a partial list is given in Table 1 and also provided in [11]), defines tokens to be used by concrete instantiations of the abstract SLA syntax. These terms include constraint expressions, logical and comparison operators, arithmetic functions, time-series and scheduling functions and common QoS Metrics (such as completion-time, availability etc). All the standard terms are defined as URIs, such that different namespaces may be employed to signify local variations. Domain-specific applications can define their own additional terms as required.

An example instance of an SLA specification is illustrated in Figure 2. Here the SLA instance represents one *AgreementTerm*. Within the AgreementTerm (AG1) are variable declarations (*VariableDeclr*) and a number of obliged *Guaranteed* states. The variable VAR1 defines a reference variable for the effectiveDate of the AgreementTerm. The obligations of the service operation (identified in a complete SLA specification) are one or more Guaranteed states, which specify constraint expressions as a pair of values and domains. The value is either a function or another expression (or indeed a variable substitution). The domain refers to the operator of the expression and the value to be evaluated. Hence, a Guaranteed state can be built from an expression tree of constraint expressions and their domain values. For example, AG1 in Figure 2 has a Guaranteed state (ID as Availability) which specifies that the availability of the service should be greater than 60 percent. Using this form of specifying SLAs, complex Agreement Terms and Guaranteed states can be built.

Table 1: Sample list of QoS Terms as defined in SLA@SOI project

QoS Terms	
Numerical Measures: S-Service	
availability(S)	probability service S is running
accessibility(S)	probability that S is accessible
arrival_rate(S)	operation invocation rate in S
data_volume(S)	data volume processed per request
throughput(S)	max. arrival rate for operations in S
completion_time(S)	operation completion time in S
mttr(S)	mean-time-to-repair
mttf(S)	mean-time-to-failure
Standards Conformance: S-Service	
non_repudiation(S)	possibility to deny use of S
supported_standards(S)	set of standards supported by S
regulatory(S)	set of regulations supported by S
Security: S-Service	
authentication(S)	proper methods used by S
auditability(S)	logs are maintained by S
Infrastructure: R-Resource	
vm_cores(R)	number of cores of R
cpu_speed(R)	processing speed of R
memory(R)	amount of memory of R
persistence(R)	vm image of R is persisted
vm_image(R)	virtual image for R

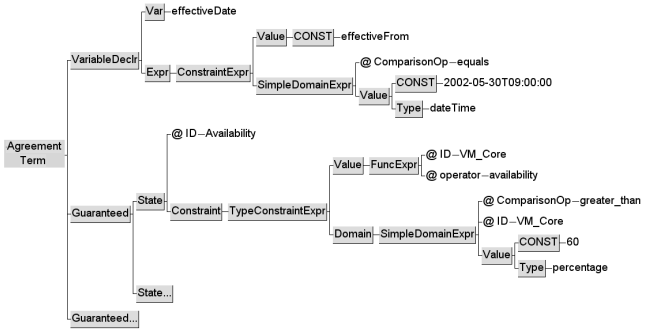

Figure 2: Partial SLA as XML Elements

3.2 Monitoring Features Specification

In addition to an SLA specification, the MonitoringManager requires a set of monitoring features specifications for monitoring feature types (introduced at the beginning of this section). Component Monitoring Features are specified for a type of monitoring component and offered for a type of service. A ComponentMonitoringFeature specification has two instance variables. The *type* variable holds the type of the component. The permitted types are: SENSOR, EFFECTOR, and REASONER. A sensor provides information about a service, an effector changes the properties of a service, whilst a reasoner processes information to produce a monitoring result, e.g. it consumes information provided by sensors and reports whether an SLA is violated or not. The *uuid* variable uniquely identifies the component that has the monitoring features. This variable has the same value as the service UUID. Furthermore, a Feature contains a list of MonitoringFeatures. MonitoringFeatures are constructed as one of two types, being basic or a function.

Basic monitoring features are used to distinguish between event and primitive monitoring features. It has a single parameter *type* for the type of the basic monitoring feature. In the case of Primitive monitoring features allowed types correspond to the Java primitive types, e.g., Long, Boolean, String. In the case of an Event monitoring feature allowed types are currently REQUEST, RESPONSE and COMPUTATION (result of a function). A basic monitoring feature with a sub-type of *primitive* is used to advertise abilities to report about primitive service information, e.g., cpu_load, logged_users, available_memory. Sensors are the typical components exposing this kind of feature. A primitive feature has two instance variables: *type*: holds the variable type. It can be, for instance, one of the Java standard primitive type. It can also be any other type defined in an SLA standard vocabulary. *unit*: holds the monitoring feature unit of measurement, e.g., mt, km, kg. *Event* monitoring features are used to advertise abilities to report about service interactions or service state, e.g., service operation requests and responses, service failures. Sensors and Reasoner are the typical components exposing this kind of feature. An Event basic sub-type has one instance variable *type*, which holds the event type as one of REQUEST, RESPONSE, COMPUTATION. Domain specific event types can also be defined and used here.

Function monitoring features are used to advertise abilities to perform a computation and report its result, e.g., availability, throughput, response_time. Reasoners are the typical components exposing this kind of feature. The class Function has two instance variables: *input*: holds the list of the function input parameters, *output*: holds the output parameter. Reasoner features are described by a type (the term or operator performed), one or more input parameters and one output.

The example in Figure 3 illustrates the features of an example service. A sensor feature has two monitoring features; one for events reporting *cpu-load* and one the number of *logged-users*. The example also illustrates a reasoner feature with two monitoring features. One providing a *greater-than* comparison of two input parameter numbers. The other providing an *MTTR* (Mean-Time-To-Repair) computation output based upon request and response input events.

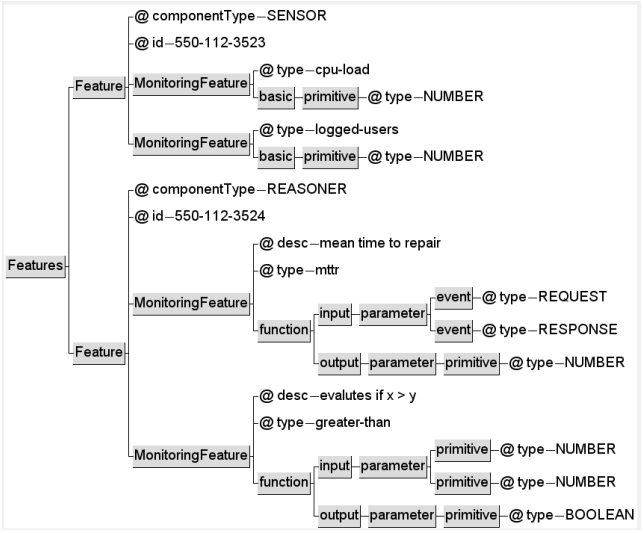

Figure 3: Component Monitoring Features as XML Elements

4. APPROACH TO CONFIGURATION

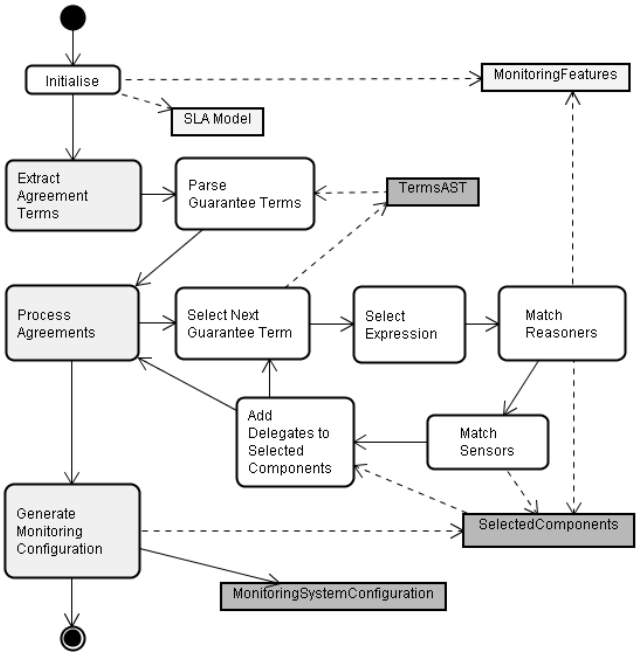

Figure 4: SLA Monitoring Configuration Activities

Our approach to dynamic configuration of monitoring infrastructures is based on the process illustrated in Figure 4. The process takes as input an SLA specification and a series of Monitoring Features. Given these inputs, initially the MM extracts the Guarantee States from Agreement Terms of the SLA specification. The terms are in turn parsed in to a formal Abstract Syntax Tree (AST) for the expressions of the states. The AST is then used as input to select each Guarantee State expression (by traversal of the AST) of each state and match each left-hand side (lhs), operator and right-hand side (rhs) of the expression with appropriate monitoring features. The matching algorithms are discussed in section 6. Following selection, the delegate components form a SelectedComponents list, which in turn, is used to generate a complete Monitoring System Configuration (MSC) result for an agreement. If no suitable monitoring configuration can be formed (i.e. not all monitoring requirements could be matched) then an empty configuration is returned for a particular guarantee term. The approach can be used for two perspectives; first, to configure the monitoring system when a new SLA needs to be monitored, and second to perform adjustments to an existing configuration when requirements change or violations are detected. However, main focus in this paper is the first, in that we assume that a new SLA is to monitored and therefore do not consider how this would affect the current state of monitoring. The end result of the configuration process is an MSC representing the configuration of selected Monitoring Components which reason or provide events to monitor each Agreement Term of the input SLA.

5. SLA TERM DECOMPOSITION

The Monitoring Manager abstracts the Guaranteed States (a guarantee made by one of the parties involved in the agreement) that a certain state of affairs will hold for the service. We abstract these states from the Agreement terms and parse the terms using a grammar which is based upon the Backus Normal Form (BNF) specification of the SLA specification [11]. The grammar for the parser is currently only based upon the Agreement Term and Guaranteed State expressions. A sample part of the grammar is listed in Figure 5.

```
1  /*********************************************
2  * SLA: The Specification of AgreementTerms
3  *********************************************/
4  void SLA() : {} {
5    AgreementTerm()* }
6  /*********************************************
7  * Agreement: AgreementTerms in SLA Model
8  *********************************************/
9  void AgreementTerm() : {} {
10   GuaranteeTerm()((TermOperator())
11   GuaranteeTerm())* }
12 /*********************************************
13 * GuaranteeTerm: A Guaranteed State expression
14 *********************************************/
15 void GuaranteeTerm() : {} {
16  <QUOTED_STRING> (Term())(Comparator())(Term())}
17 /*********************************************
18 * Term: One or more Term functions or Identifier
19 *********************************************/
20 void Term() : {} {
21   LOOKAHEAD(TermFunction())TermFunction()
22   | <STRING> | <QUOTED_STRING> }
23 /*********************************************
24 * Comparator: Operators in term expression
25 *********************************************/
26 void Comparator() : {} {
27  ( <EQUALS> | <NOTEQUAL> | <LTHAN> | <GTHAN> |
28    <LEQUAL> | <GEQUAL> | <ISEQUALTO> ) }
```

Figure 5: Partial JAVACC Grammar for SLA Term Decomposition

The grammar is used as input to the Java Compiler Compiler (JAVACC) [14] which generates compiler source code to accept and parse source files specified in a defined grammar language. The resulting AST is built to represent the SLA specification terms and expressions. Beginning with the SLA declaration (lines 4-5) one or more AgreementTerms are parsed. Each AgreementTerm (lines 9-11) is parsed as one or more GuaranteeTerms, separated by a comparison operator. Each GuaranteeTerm (lines 15-16) is then parsed as an identifer (which holds the id label of the GuaranteeTerm), and a basic Term followed by a comparison operator and then followed by another basic Term. Each basic Term (lines 20-22) is represented by either one or more TermFunctions (similiar to a normal function call syntax), a string identifier (representing a variable of the SLA specification. The JAVACC function *LOOKAHEAD* informs the parser to check whether the next symbol to parse is a function or string. Finally, the comparator operators (lines 26-28) list the acceptable types of operators that can be used between terms.

Since the term decomposition is based upon a generated parser, other SLA specification formats may generate their own parsers and transform their SLA specification to the AST input required by the MonitoringManager. In this way, the implementation of the configurator is generic and reusable. In addition, the generated AST compiler can be reused by Monitorability Agents (which accept the monitoring system configuration as a result of matching monitoring

components) who can translate the SLA terms in to their own language specification. As an example, we have already performed such a translation for the EVEREST monitoring language [12] that is based on Event Calculus, which is used to reason about the expressions in the SLA@SOI project SLA examples.

6. MONITORING CONFIGURATION

6.1 Monitoring Terms

The main configuration algorithm (illustrated in Figure 6) is responsible for selecting all the term expressions from the prepared SLA term tree (TermAST), obtaining a match for the expression terms with available monitoring component features and then building a suitable monitoring system configuration. The algorithm begins by selecting the root of each Agreement Term expression, which in turn holds one or more Guarantee State expressions.

Function:	Given an agreement, select the most appropriate monitoring components.
Input(s):	1) TermAST - An AST of the Guaranteed Agreement Terms. 2) Features - a list of service monitoring features.
Output(s):	A set of monitoring components with configurations.
Algorithm:	Given the TermAST and a set of Monitoring Features 1) **select** root of AST and extract *expressions* 2) **extract** lhs, rhs, operation and **select** input-types 3) **set** M1 to **MonitorConfig(*lhs*)** 4) **if** *node.lhs* is **expression** then (a) **set** M2 to **MonitorConfig(*rhs*)** **otherwise set** M2 to *rhs.value* 5) **set** RM to **SelectMonitor(*input-types*,*operation*,*Features*)** 6) **store** delegate for expression

Figure 6: Algorithm for MonitorConfig

The Agreement Term expression is pre-defined as a set of boolean expressions (where all must be true for the Agreement Term to be upheld). Each Guarantee State has a left and right-hand side term and an operator. From the terms a set of input-types is determined. Two term monitors (M1 and M2) are set to reason about the terms and a reasoner monitor is set to reason about the main expression. If the left-hand side of the expression is itself an expression then the second monitor (M2) is recursively configured using the same algorithm (MonitorConfig). If it is not, then the value of the right-hand side of the expression is used as the monitor. Furthermore, a reasoner monitor is assigned to the selection of an appropriate monitor for the input-types, operation and with the list of Features supplied.

6.2 Monitor Selection

The MonitorConfig algorithm uses a SelectMonitor algorithm (Figure 7) to match the required types and operations (or term names) to the monitoring component features. The algorithm begins by iterating through the monitoring component features available and building an appropriate feature list (FeaturedMonitors) by selecting the monitors that match the type of term or operator. Each FeaturedMonitor is then selected and checked for appropriate input-types. For example, the operator < (less than) can be provided for boolean, numbers or other data types. If the feature and types match then the FeaturedMonitor is added to a list of selected monitors (SelectedMonitors). In the current implementation of the work we simply select the first monitor matched as an appropriateness measure. It is envisaged for an enhanced implementation to use some optimisation algorithm (at step 3. of the algorithm) based upon a criteria passed by the user (or indeed, specified as part of the overall SLA). This could also include assessing using the same provider of features to reduce financial cost, optimise messaging and to group related monitors.

Function:	Given a set of input-types and a monitor term, select the first monitor that matches the term or event types required.
Input(s):	1) Input-Types - A set of types (e.g. Number, Event) etc. 2) Term - A term or operation to be monitored (e.g. completion-time or < (operator). 3) Features - a list of service monitoring features.
Output(s):	A monitoring component offering the types and operation/term.
Algorithm:	Given the Input-Types, MonitoringFeatures and Term 1) **for each** MonitoringFeature in Features **do** (a) **select** FeaturedMonitors where *type* **equals** the Term 2) **for each** Monitor in FeaturedMonitors **do** (a) **for each** *type* in Input-Types **do** (i) **if** Monitor has *Type* then (ia) **add** Monitor to SelectedMonitors 3) **select** the first Monitor in SelectedMonitors *(*replaceable selection criteria)* 4) **return** *SelectedMonitor*

Figure 7: Algorithm for SelectMonitor

6.3 System Configuration

As briefly discussed in section 4, the MSC defines an entire configuration for monitoring an SLA within the monitoring system. An example MSC is illustrated in Figure 8 showing a REASONER component (for monitoring a Guaranteed State) and then a set monitoring feature components for each part of the Guaranteed State expressions. The MSC contains a list of components representing Sensor, Effector or Reasoners selected to support the Guarantee Terms of agreements in an SLA.

Each Component in an MSC contains one or more component configurations for each of the different components. For example, an MSC can contain a Reasoner component which has component configurations for two Sensor components and one additional Reasoner component. A Sensor component configuration contains a MonitoringFeature (that which was advertised in selection for the Sensor component) and one or more OutputReceiver(s). An output receiver is another component which expects the result (as an event or value) to perform its own function. A Reasoner component configuration also specifies one or more OutputReceivers but a Specification replaces the MonitoringFeature with a list of

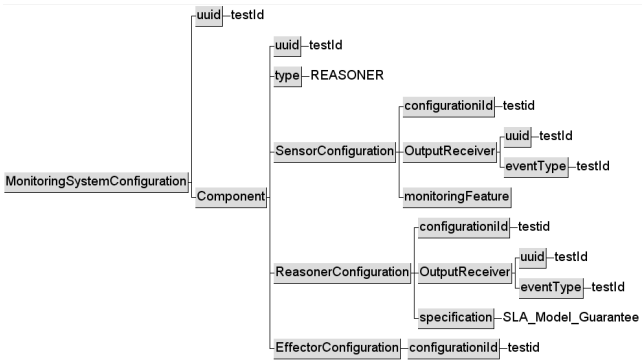

Figure 8: A Monitoring System Configuration

Guarantee States required for reasoning by the component.

6.4 Configuration Deployment

To dynamically setup the whole monitoring environment many components need to be configured. The MSC specification generated in section 6.3 structures all the needed information into a coherent inter-level representation. An MSC instance can be created for two main purposes: to configure the monitoring system when a new SLA needs to be monitored, or to perform adjustments to an existing configuration. Although the purpose of this paper is not to define how the configuration is actually used to execute the configuration of the monitoring environment we briefly outline this to aid the reader understand how the output is leveraged in the environment.

As illustrated in Figure 1, a generated MSC is passed to a *Service Manager* which links a service instance with a service *Managability Agent*. The Manageability Agent exposes a method to accept a configuration and then, on behalf of the service under agreement, starts dependent components to monitor the service activities and to generate any violations as part of that agreement. For example, each Agreement-Term has a reasonser (the sum of evaluating all Guarantee States in the Agreement). Each Guarantee State also has a reasonser (to evaluate the expression of each Guarantee State). Once the service Managability Agent is initialised, each reasoner is configured with the appropriate part of the MSC (e.g. for a cpu_load evaluation). The results generated by the reasoners and sensors in this configuration will be monitored by the Managability Agent and appropriate routed from the Event Bus. For further details of the architecture, the reader is invited to refer to [reference SLA@SOI deliverable].

7. TOOL SUPPORT

The approach and modules described in this paper are supported by a number of implementations. In particular the *MonitoringManager* component is available as a JAVA package, and is also available to be accessed as a Web Service. Since it can be hosted, we have developed a test Web application which accepts the inputs of the SLA specification (as described in section 3.1) with Component Monitoring Features (as described in section 3.2) and automatically generates the Monitoring System Configuration as described in section 6.3. The Web application is illustrated in Figure 9. The Web application provides a useful test harness to check whether the configurations can be successfully generated. Possible errors which can be logged include whether the specifications are appropriately defined, or whether there are sufficient features described to fulfil each of the monitoring term requirements. Thus, it is a useful interactive design tool but also can be accessed at runtime through the Web Service interface. The MonitoringManager will be integrated as part of a SLA@SOI project platform showcase and will be available from: **http://www.sla-at-soi.org/**. A full evaluation of the approach and results are being carried out in-line with the SLA@SOI project functional integration tests.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have described an approach to advanced configuration of service systems, in particular, ones in which an SLA agreement has been established and has services requiring monitoring. The work aims to provide a generic module applicable not only to the architecture illustrated but also to other architectures (although still based upon SLAs and Monitoring Component Features). Our work will be extended to cover further elements of the SLA specification (such as Guaranteed Actions, which are not presently considered) and also including preferential selection of monitoring components. Preferential selection of components is useful where there are multiple monitoring components offered for the same term. Preferences could be based upon monitoring cost (both in computing resource or financially) or non-functional requirements. The existing implementation is already part of the wider SLA@SOI project monitoring platform, providing integration and validation testing, and we seek to find other environments to test it within.

9. ACKNOWLEDGMENTS

The research reported in this paper has been supported by the EU Commission as part of the F7 Integrated Project *SLA@SOI* (grant agreement n. 216556).

10. REFERENCES

[1] L. Baresi, D. Bianculli, and C. Ghezzi. Validation of Web Service Compositions. *IET Software*, 1(6):219–232, 2007.

[2] L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *International Conference on Service-Oriented Computing (ICSOC)*, 2005.

[3] D. Bianculli and C. Ghezzi. Monitoring conversational webservices. In *2nd International Workshop on Service Oriented Software Engineering (IW-SOSWE)*, 2007.

[4] M. Comuzzi and G. Spanoudakis. Dynamic set-up of monitoring infrastructures for service-based systems. In *25th Annual ACM Symposium on Applied Computing, Track on Service Oriented Architectures and Programming (SAC 2010)*, Sierre, Switzerland, 2010. ACM.

[5] Gridipedia. *SLA Monitoring and Evaluation Technology Solution.* Available from: http://www.it-tude.com/?id=gridipedia, 2009.

[6] K. Jank. *Reference Architecture: Adaptive Services Grid Deliverable D6.V-1.* Available from: http://asg-platform.org/twiki/pub/Public/ProjectInformation, 2005.

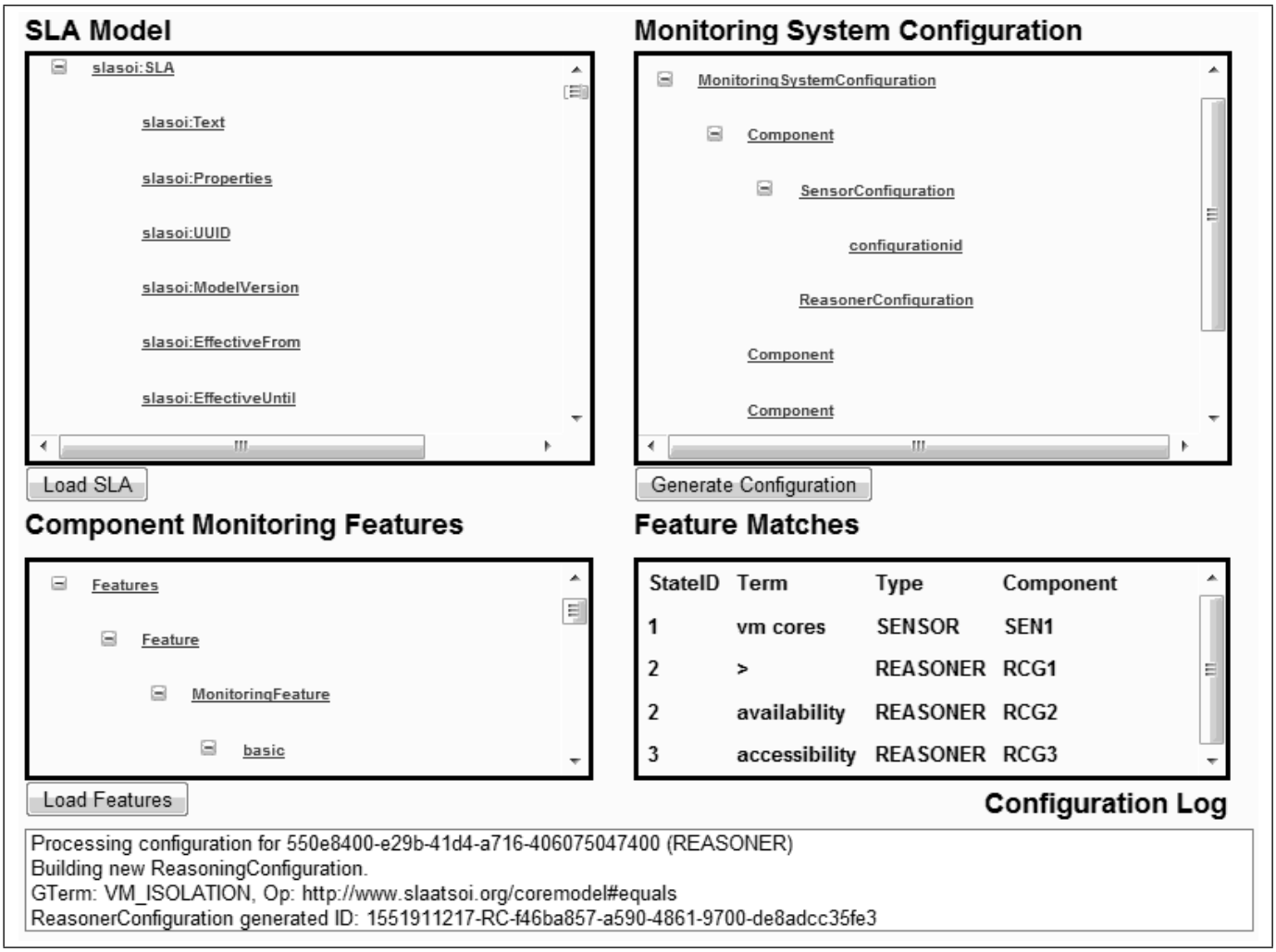

Figure 9: Prototype Monitoring System Configuration Explorer

[7] A. Lazovik, M. Aiello, and M. Papazoglou. Planning andMonitoring the Execution of Web Service Requests. *International Journal of Digital Libraries*, 2006.

[8] K. Mahbub and G. Spanoudakis. Run-time monitoring ofrequirements for systems composed of web services:initial implementation and evaluation experience. In *International Conference on Web Services (ICWS)*. Springer, 2005.

[9] J. Richter, C. Baruwal, R. Kowalczyk, B. Quoc Vo, M. Adeel Talib, and A. Colman. Utility Decomposition and Surplus Redistribution in Composite SLA Negotiation. In *IEEE International Conference on Services Computing*, 2010.

[10] A. Sahai, V. Machiraju, M. Sayal, L. J. Jin, and F. Casati. Automated SLA Monitoring for Web Services. In *IEEE/IFIP DSOM*, pages 28–41. Springer-Verlag, 2002.

[11] SLA@SOI. *Deliverable D.A1a: Framework Architecture*. Available from: http://sla-at-soi.eu/publications/deliverables, 2009.

[12] G. Spanoudakis, C. Kloukinas, and K. Mahbub. The serenity runtime monitoring framework. In *Security and Dependability for Ambient Intelligence,Information Security Series*. Springer, 2009.

[13] G. Spanoudakis and K. Mahbub. Non Intrusive Monitoring ofService Based Systems. *International Journal of Cooperative Information Systems*, 15(3):325–358, 2006.

[14] Sun Microsystems. *The Java Compiler Compiler (JAVACC)*. Available from: https://javacc.dev.java.net/, July 1999.

[15] TrustCOM. *Deliverable 64: Final TrustCoM Reference Implementation and Associated Tools and User Manual*. Available from: http://www.eu-trustcom.com/, 2007.

[16] W. Van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek. Conformance checking of Service Behavior. *ACM TOIT*, 8(3), 2008.

[17] C. Yuan, S. Iyer, X. Liu, D. Milojicic, and A. Sahai. SLA Decomposition: Translating Service Level Objectives to System Level Thresholds. In *Fourth International Conference on Autonomic Computing (ICAC)*, 2007.