

# MONITORING MIDDLEWARE FOR SERVICE LEVEL AGREEMENTS IN HETEROGENEOUS ENVIRONMENTS

Graham Morgan<sup>1</sup>, Simon Parkin<sup>1</sup>, Carlos Molina-Jimenez<sup>1</sup>, James Skene<sup>2</sup>

<sup>1</sup>*School of Computing Science, University of Newcastle upon Tyne, UK;* <sup>2</sup>*Department of Computer Science, University College London, UK*

**Abstract:** Monitoring of Service Level Agreements (SLAs) is required to determine if the Quality of Service (QoS) provided by a service provider satisfies the expectations of a service consumer. Although tools exist that can generate the software required to evaluate SLAs from the SLA specifications themselves, the code required to gather metric data is still predominantly coded by hand: a time consuming task. In this paper we describe an SLA monitoring implementation that can generate metric data gathering software directly from machine readable SLAs. Assuming that an organisation specialising in SLA monitoring and evaluation may not wish to be tied to any one particular middleware platform and/or SLA language, we aim to provide generic monitoring services that may be suitable for use in heterogeneous environments. We demonstrate the flexibility of our approach by providing monitoring solutions for observed systems implemented using Web Services and Enterprise Java Bean (EJB) middleware using a third party SLA language.

Key words: monitoring, service level agreements, middleware

## 1. INTRODUCTION

Service Level Agreements (SLAs) specify the Quality of Service (QoS) associated with the interaction between the provider of a service and a service consumer. SLAs are gaining in importance as increasing numbers of companies conduct business over the Internet (e.g., banking, auctions), requiring the positioning of SLAs at organisational boundaries to provide a basis on which to emulate the electronic equivalents of contract based business management practices.

Monitoring is required to collect statistical metrics about the performance of a service to determine if the QoS agreed upon between provider and consumer is realised. Third parties may assume responsibility for monitoring SLAs to ensure the results of the evaluation process are trusted by both the provider and consumer [2].

Our previous work on the monitoring of SLAs [13] presented an architecture that covers the fundamental issues of SLA monitoring: SLA specification, separation of the computation and communication infrastructure of the provider, service points of presence, metric collection approaches, measurement service and evaluation & detection service. As a next step, we now turn our attention to the implementation of our architecture. As in our previous work on design, we assume the viewpoint of an organisation that is concerned with the provisioning of third party monitoring for participants of SLAs. If such an organisation is to support SLA monitoring for many different types of clients then an assumption that only a single SLA language will suffice and all technologies are enabled via a single middleware standard may not be realistic.

To facilitate the process of SLA evaluation, metric data must be gathered by software components, possibly within the service provider domain, as specified by an SLA. Hand coding such software on a per SLA basis is a time consuming task, especially if an organisation specialising in SLA monitoring must deal with many thousands of SLAs. The automated parsing of machine readable SLAs by an SLA violation and detection tool-kit can derive the software components required for SLA violation detection [14]. However, deriving the software components required for the monitoring of metric data in a similar manner has not yet been addressed.

Building on our previous work on the design of an SLA monitoring architecture, this paper presents an approach to SLA monitoring that requires minor tailoring to work with different SLA languages and middleware platforms. Our system is capable of deriving the appropriate metric gathering software directly from machine readable SLAs. We demonstrate the suitability of our approach by tailoring our system to work with an application providing services across the Internet, governed by SLAs described using an existing SLA language, deployed over Enterprise Java Beans (EJB) and Web Service middleware.

This paper is organised as follows. Section 2 describes background and related work, section 3 describes our implementation and section 4 provides conclusions and future work.

## 2. BACKGROUND & RELATED WORK

For completeness, and to clarify our approach to SLA monitoring, this section continues with a description of our previous work on the development of an SLA monitoring architecture. Via this description we identify when it may be possible to ease the development of metric gathering middleware via software automatically generated directly from SLAs. We then present a discussion in which we determine the suitability of other works in providing a general purpose SLA monitoring service for heterogeneous environments (existence of different SLA languages and middleware platforms).

### 2.1 SLA Monitoring Architecture

The architecture we proposed [13] for monitoring SLAs is shown in Fig. 1. For sake of simplicity, we assume that the provision of the service is unilateral, that is, the service flows only from the provider to the service consumer, as opposed to bilateral provisioning where the two interacting parties provide services to each other; bilateral provisioning is a more general scenario and may be represented by two complimentary unilateral deployments. With unilateral service provisioning we need to monitor the observance of only two contractual obligations: (i) the provider's obligations, dictating that the service must satisfy certain QoS requirements; and (ii) the service consumer's obligations, which dictate how the service consumer is expected to use the service.

We assume that calculations relating to QoS are specified explicitly (e.g., maximum latency) in a computer readable format, allowing automated SLA evaluation and violation detection.

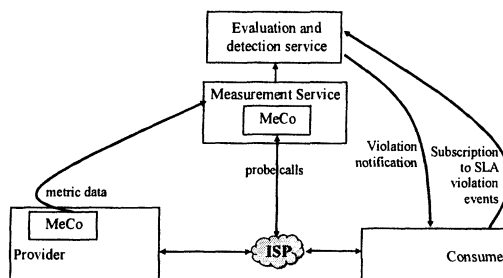


Fig. 1. Architecture for unilateral monitoring of QoS

The components shown in Fig. 1 assume responsibility for SLA monitoring and evaluation:

- *Metric collector (MeCo)* – Gathers metric data associated with the performance and usage of the observed system.
- *Measurement service* – Measures a given list of metrics at specified intervals.
- *Evaluation and violation detection service* – Inspects gathered metric data to determine if SLA violation has occurred and informs provider/consumer of such violations.

The MeCos shown in the Fig. 1 gather metric data relating to the provider's obligations (MeCo in measurement service) and the consumer's obligations (MeCo in service provider). This scenario assumes a probing style approach to service monitoring. That is, synthetic load is generated by a simulated client (provided by measurement service) to determine if the provider is satisfying SLAs [3] [9]. An alternative to probing would be to have a MeCo co-located with the consumer and gather metric data associated with actual client calls. We consider only the probing approach in this paper as it may not be possible to deploy monitoring at the consumer side (as consumers may not always agree to accept metric collection responsibilities).

To ease the development costs of monitoring middleware solutions, the automated production of MeCos from SLAs for use over a variety of middleware platforms would be welcome. This is analogous to the production of client/server stubs for easing the implementation of *remote procedure call* (RPC) code: an interface specification is parsed to produce the required code to enact communications across process space (possibly between nodes on a network).

Once metric data has been received by the measurement service, the data must be prepared in a suitable format for handling by the evaluation and detection service. This should be straightforward as the SLA specifies exactly what data is required and in what form. However, an organisation specialising in SLA monitoring may utilise a number of SLA languages for satisfying the different requirements found in a variety of application domains. In this situation the measurement service must be capable of interfacing with the evaluation and detection service via a number of different SLA language standards, even though the measurement service's basic functionality remains unaltered. Therefore, an appropriate approach to implementation would be to allow the measurement service to work with arbitrary SLA languages with only the minimum of tailoring. As the SLA identifies the types of metric data to be evaluated, the ability to automatically generate code that provides translation of metric data to a format suitable for

processing by an SLA evaluation tool (which is SLA language dependent) is required.

We may summarise opportunities for automated code generation to ease implementation in the following ways:

1. Ease the development of a MeCo using SLAs to automatically derive metric gathering software for a number of varying middleware platforms.
2. Ease the development of software for enabling SLA language integration into the measurement service by automatically deriving such software from the SLAs themselves.

## 2.2 Implementation/Deployment Issues & Related Work

An approach to MeCo deployment is via the use of middleware interceptors (e.g., [8]). Interceptors are middleware components that can be placed between application components to provide additional functionality (e.g., security, redirection). Interceptors provide an opportunity to implement SLA monitoring with the minimum of modification to an observed system. Popular implementations of middleware standards (i.e., CORBA, EJBs, Web Services) provide interceptor type mechanisms. Therefore, the use of interceptors is widely advocated as the appropriate way of providing SLA monitoring for distributed applications. However, existing implementations of MeCo type interceptors are middleware dependent (e.g., CORBA [5] [7], Web Services [1] [4] [6]), making a single implementation unfit for deployment over a number of middleware platforms. This homogeneous approach makes existing metric data gathering solutions difficult to use in heterogeneous environments (requiring a single implementation to be substantially modified or combining different implementations).

There are a number of SLA languages proposed by the literature (e.g., Web Service Level Agreements (WSLA) [4], Quality Description Languages (CDL) [5], Service Level Agreement Language (SLAng) [11]). Unfortunately, no existing implementation allows the use of multiple SLA languages.

The process of automated code generation from SLAs for the purposes of SLA evaluation has been demonstrated (e.g., [6] [11]). However, using an SLA to generate a MeCo (or equivalent) for gathering appropriate metric data has not yet been realised. The related work that comes closest to automated MeCo generation from SLAs is [6]. Via the use of *business management platform* (BMP) agents the work presented in [6] concentrates on the automation of SLA monitoring for Web Services. The distributed nature of the approach described in [6] provides an opportunity to manage

metric data collection at observed systems with the minimum of human involvement. However, this peer-to-peer approach is not suitable for all application types, and not suitable for an organisation delivering SLA monitoring services using our architecture.

As demonstrated by [7] (QoS monitoring associated with network traffic engineering), scalability may be a requirement for a practical deployment of SLA monitoring. When delivering SLA monitoring services (even in an e-commerce environment) scalability of message dissemination is desirable (especially to a third party monitoring service that may have hundreds, or thousands, of clients). [7] highlights the usefulness of *message oriented middleware* (MOM) as an appropriate message dissemination medium for metric data. An alternative to MOM would be to use a client/server approach (e.g. RPC).

The client/server model requires clients and servers to record references to each other to enable the initiation of bi-directional information flow. The scalability of such a model is difficult to maintain when the number of interconnected clients and servers may be appropriately measured in hundreds or thousands. Furthermore, when using RPC the processing of messages must be handled as and when messages are received by clients and servers. The MOM model is considered suitable for large-scale data dissemination as it tackles these two problems by presenting a weakly coupled message passing environment. In the MOM model, information flow is not based on the referencing of the sender and receiver, as in client/server, instead information flow is based on the properties of a message. Evidence provided by [7] indicates that propagation of metric data and SLA violation notifications can be best served via the use of MOM technologies.

In summary, the monitoring of SLAs in an environment consisting of different SLA languages and different middleware platforms is not possible using existing approaches. Furthermore, the automated generation of code specifically for metric data gathering, although desirable and progressed by [6] [7], is not realised. In addition, providing a messaging infrastructure using MOM technologies is shown to be beneficial [7] when developing a scalable metric gathering solution. However, this scalability issue appears only to have been addressed in traffic engineering (as opposed to inter-organisational middleware) solutions.

In the remainder of the paper we describe the implementation of our monitoring architecture. The primary focus of the paper is the easing of the development of SLA monitoring and evaluation software for heterogeneous environments. A description of how we use MOM as a basis for our messaging services to allow for scalability is provided for completeness.

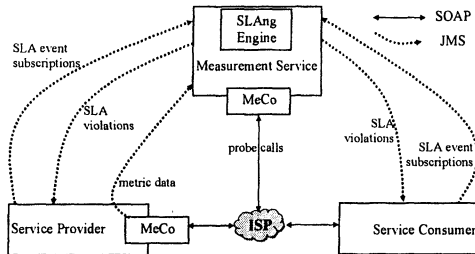
### 3. IMPLEMENTATION

As already mentioned in Section 2, our approach to SLA monitoring is based on our earlier work described in [13], culminating in the architecture shown in Fig. 1. For our SLA language we use SLAng [11]. SLAng represents the product of work carried out at University College London (UCL).

SLAng meets the needs of an SLA language to support the construction of distributed systems and applications with reliable QoS characteristics. The *Unified Modeling Language* (UML) is used to model the language, producing an abstract syntax. This language model is embedded with an object-oriented model of services, service consumers and their behaviour. Constraints are defined formally using the *Object Constraint Language* (OCL), providing the semantics. This approach permits natural and economical modeling of design and analysis domains and the relationships between them, supporting both manual and automatic analysis.

The monitoring system we have constructed uses metric collection as defined in SLAng and uses the SLAng engine for automating SLA evaluation. From an SLA defined using SLAng it is possible to automate the production of the appropriate software components needed for SLA evaluation (incorporated into the SLAng engine). It is worth noting that the SLAng engine only checks a limited number of system performance metrics, notably those related to request latency, service availability and percentage of service usage (i.e., how many requests service consumers are issuing over a period of time). We have developed a formal notation for describing conventional contracts by means of Finite State Machines (FSMs) for representing more application dependent QoS [17]. However, for brevity and to demonstrate our work we only consider metrics as described using SLAng.

We assume that the communications that are required to be monitored are enacted over middleware technologies that support message interception. This is a valid assumption as all major middleware vendors provide a mechanism for message interception in their technologies (e.g., interceptors in CORBA, handlers in SOAP, interceptors in EJB containers).



*Fig. 2.* SLAs monitoring architecture with message oriented middleware

The architecture shown in Fig. 2 alters the architecture shown in Fig. 1 to accommodate our approach to implementation. For completeness (some of the descriptions deviate little to those presented in section 2) we provide descriptions of the components in Fig. 2:

- *Service provider MeCo* - Intercepts service consumer requests (and associated outgoing responses) and records measurements based upon a service consumer's usage of the service provider's platform. These measurements aid in determining if a service consumer is violating an SLA by using a service inappropriately (excessively in our case study).
- *Measurement service MeCo* - Observes the performance of a service provider by assuming the role of a service consumer. Periodic probing of the service provider is enacted by the measurement service MeCo to gain measurements relating to the performance of a service provider as viewed by a service consumer. These measurements aid in determining if a service provider is satisfying service consumers as specified in an SLA.
- *Measurement service* - Responsible for collecting the measurements gathered from MeCos and informing SLA participants of SLA violations.
- *SLAng engine* - A sub-system of the measurement service that is responsible for detecting SLA violations given metric data supplied by the measurement service.
- *Messaging service* - Provides communication platform across which metric data and SLA violation notifications are propagated.

The measurement service is within the domain of a trusted third party, ensuring that service provider and consumer may abide by the decisions on SLA violations generated by the SLAng engine.

In the following sections we describe the implementation of each component and how different components collaborate to provide SLA monitoring and SLA violation notification. Our implementation is based on SLAng, EJB and Web Services. We state the type of tailoring that may be



required to enable other SLA languages, including SLA engines, and middleware platforms to work with our Java implementation.

### 3.1 Metric Collectors (MeCos)

MeCos are responsible for gathering metric data and propagating such data to the measurement service for evaluation. Service providers have a MeCo within their organisational domain for monitoring service consumer usage. MeCos are suitable for use with arbitrary middleware platforms (and associated protocols). Different middleware platforms may be supported with the use of *MeCo hooks*. Irrelevant of middleware platform, MeCo hooks determine what metric data to gather from loading classes (*metric data classes*) from the class repository (classes generated directly from SLAs). A wrapper class (*platform wrapper*) is required to allow integration of the metric data classes into a specific platform (a MeCo hook is the combination of platform wrapper classes and metric data classes).

A MeCo hook (specifically the platform wrapper component) is middleware dependent and is responsible for the interception of consumer request/reply messages and passing such messages through the MeCo. So far, we have demonstrated the use of MeCo hooks for supporting Web Services using SOAP and Enterprise Java Beans (EJBs) using Java Remote Method Invocation (Java RMI). This combination was chosen as these two approaches are combined in many vendor middleware products that provide implementations of *Java 2 Enterprise Edition (J2EE)*, a well known architecture designed to ease the development of enterprise computing solutions.

The specification of J2EE defines a platform for developing Web-enabled applications using Java Server Pages (JSPs), Servlets and EJBs. Application servers for Java components (also called J2EE servers) are expected to provide a complete implementation of J2EE. Web Services provide a presentation of services for inter-organisational communications with the back end application logic implemented using EJBs. We used the JBOSS application server [10] for our J2EE implementation.

Our SOAP MeCo hook implementation is based on *Apache eXtensible Interaction System (Axis)* [15]. Axis provides handlers (*Axis Handlers*) that may be chained together to provide a mechanism for interception, and possible alteration of a SOAP message (e.g., add/remove headers, manipulate the body), at different points during traversal of the protocol stack (i.e., before request is processed by server side logic or before reply is received by a client). Axis handlers provide an appropriate opportunity to redirect SOAP messages to a MeCo (via MeCo hooks) for metric gathering. The addition of Axis handlers does not require alterations to the application

logic, therefore the introduction of monitoring at the service provider may be achieved in a transparent manner. We use JBoss interceptors to implement MeCo hooks suitable for interception of Java RMI invocations.

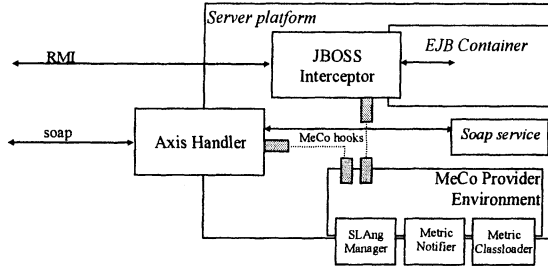


Fig. 3. Service Provider use of MeCos

Fig. 3 shows the architecture of MeCo deployment in the service provider. The MeCo provider environment contains a number of components that cumulatively satisfy the metric collection and dissemination (back to the measurement service) requirements of our monitoring system (Fig. 2):

- *SLAng Manager* – Examines an SLA (as used by SLAng engine) to determine the metric data that the MeCo is to observe. The product of parsing an SLA is a Java class (*metric data class*) that may be used for gathering the appropriate metric data. This metric data class is stored in a class repository for later use. As there may be many SLAs that a MeCo is responsible for monitoring at any one observed site, streamlining of the monitoring may occur by avoiding duplicate monitoring requests. For example, if  $SLA_1$  and  $SLA_2$  describe the upper bound latency for a client invocation  $C_1$ , then the message interception associated with  $C_1$  by a single MeCo hook may satisfy the monitoring requirements of both  $SLA_1$  and  $SLA_2$ .
- *Metric Notifier* – Based on the deduction of what to monitor made by the SLAng manager, the metric notifier assumes responsibility for managing the appropriate message passing between MeCo and measurement service. This requires the lifecycle management of message channels over which metric data will travel.
- *Metric Classloader* – Loads the metric data classes for implementing the monitoring of the required data as specified by the SLAng manager. Metric data classes are loaded from the class repository. Each class represents a metric type as specified by an SLA used by the SLAng engine (e.g., response time).

The MeCo provider environment was developed in a modular fashion so the minimum of tailoring is required to make a MeCo work with different middleware platforms, and different SLA languages. The MeCo hooks, as already discussed, allow different protocols and associated middleware platforms to be supported (only the platform wrapper parts of the MeCo hooks require tailoring on a per-middleware/protocol basis). For each SLA language a different SLAng manager and class repository is required as SLA parsing (by the SLAng manager) and different mechanisms for metric data monitoring are required. This approach has the benefit of allowing SLA language extensions to be incorporated into a MeCo as and when required.

The MeCo in the measurement service differs from the MeCo located in the service provider in that the measurement service MeCo is employed to periodically probe the service provider. Probing in this manner is carried out to gain metric data relating to how service providers appear to be performing as viewed by a service consumer (e.g., response time of service provider). A tool suitable for producing synthetic load may be used (e.g., JMeter [16]), to simulate the clients and implement the desired probing strategy. Alternatively, a basic probing strategy may be created and enacted automatically by the MeCo by parsing the appropriate SLAs. The probing strategy enacted by the MeCo is sufficient for determining SLA violations. Configuration relating to the probing of a service provider is located in a Web Service Descriptor Language (WSDL) file. WSDL files are used to describe how to communicate with a Web Service, and as such can be used to configure the probe to send messages to the target server. The (Java) classes required to enact probing are created via the parsing of additional extensibility elements defined in the given WSDL file. These elements also provide a realistic set of parameters to supplement this approach to probing. As with the platform wrapper class in the service provider MeCo, a platform wrapper class is used for implementing the probing for a specific middleware platform (EJB/RMI or Web Services/SOAP).

Once requests have been created and sent as part of a probing strategy, they are intercepted by the measurement service MeCo in the manner described previously (via MeCo hooks) with metric data passed from the MeCo to the measurement service.

### **3.2 Messaging Service**

The messaging service is responsible for passing metric data from the service provider MeCo to the measurement service and passing SLA violation detection messages from the measurement service to interested parties of an SLA. The *Java Messaging Service* (JMS) [12] was chosen as the message platform.

The JMS specification does not indicate how the underlying system implementation is achieved, resulting in a number of varying solutions available from different vendors. A number of solutions that attempt to provide scalability have been proposed (e.g., [18]). Therefore, our scalability concerns are related to the way we use the standard JMS API (not the underlying messaging implementation itself).

JMS supports point-to-point and publish/subscribe models of interaction. Point-to-point is based on the notion of queues, with a queue identified as an asynchronous mechanism for passing messages from suppliers to consumers. Publish/subscribe is based on topics, with clients publishing and subscribing to well defined topics. The topic acts as a mechanism for gathering and distributing related messages (as perceived by an application) to clients and allows subscribers and publishers to be unaware of each other's existence.

The topic approach was chosen with the measurement service creating a topic on a per operation basis (e.g., the name of a method associated with an operation). We call such topics *metric topics* (in our approach each metric topic relates to a clause in an SLA).

A MeCo disseminates metric data by publishing such data on a metric topic. We found that this approach provided an opportunity to allow multiple SLA engines (checkers) to be employed. A problem with existing SLA engines is their lack of scalability when faced with checking increasing numbers of SLAs [14]. Therefore, employing additional engines (via additional measurement services) and so provide an opportunity to improve scalability is desirable in an SLA monitoring implementation. Via this method we may also allow different SLA languages to be used. The introduction of additional measurement services (and associated SLA engines) in this manner is straightforward: a measurement service registers as a consumer for the metric data they are interested in (to enable SLA violation detection). Additional measurement services may be added with minimum disruption to the overall function of the monitoring infrastructure (via subscription to appropriate metric topics). This approach may support multiple third party measurement services: a service provider may provide services to multiple consumers, with such consumers requiring different third parties to govern their SLA violation detection mechanisms (requiring different measurement services).

Propagating an SLA violation to SLA participants is achieved via a JMS topic (SLA topics). Such topics are created on a per SLA basis, with organisations assuming responsibility for registering as subscribers on the SLAs they participate in. An SLA topic message consists of a metric ID (associated with the metric that was violated) and the value that caused such a violation.

### 3.3 Measurement Service

The measurement service evaluates metric messages received from metric topics and notifies organisations, via SLA topics, of SLA violations. The measurement service contains a number of components (Fig. 4):

- *SLAng Message Manager* – Examines an SLA and determines which metric and SLA topics are required. Metric and SLA topics are created when required by the SLAng message manager. In addition, when an SLA is withdrawn from use the SLAng message manager deletes the appropriate SLA and metric topics (after determining that the metric topics flagged for deletion are no longer required by other, active, SLAs).
- *Metric Listener* – Subscribes to the appropriate metric topics as instructed by the SLAng message manager and assumes responsibility for consuming metric topic messages and translating such messages to a format suitable for acceptance by the SLAng engine.
- *SLAng Engine* – Receives messages from the metric listener and issues SLA violation notification messages.
- *Violation Notifier* – Subscribes to the appropriate SLA topics as instructed by the SLAng message manager and assumes responsibility for translating violation notification messages received from the SLAng engine to JMS messages and issuing such messages on SLA topics.
- *Metric Manager* – Generates appropriate Java classes for implementing SLA language specific functions (e.g., providing metric data in suitable format for evaluation by SLAng engine).

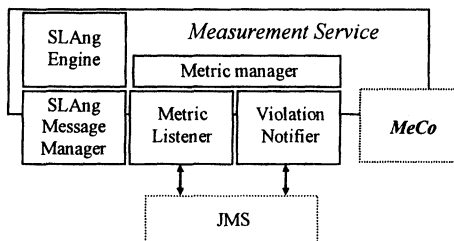


Fig. 4. Measurement service

The metric listener must translate the metric data it receives from metric topics into a suitable format for submission to the SLAng engine. This requires a service usage message to be created. A service usage message is a description of how a service was used and relates to the SLA clauses governing service/consumer interaction. The SLAng engine examines service usage messages to determine if SLA violation has occurred or if

service usage has been enacted within acceptable bounds. The violation notifier includes in the violation message details relating to what caused the SLA violation in the message issued to the appropriate SLA topic.

The service usage message is SLA language/engine dependent. However, a class repository is used (in a manner similar to how a MeCo realises what metric data to gather), to maintain a collection of Java classes that produce service usage messages as and when required. Therefore, as the metric manager is responsible for creating such classes, then a metric manager must be developed on a per SLA-language basis. In addition to creating service usage messages, there exists classes in the class repository that provide the appropriate interface code required to communicate with an SLA engine.

#### **4. CONCLUSION**

We have described an implementation of SLA monitoring that, with tailoring, provides an opportunity to monitor service provision over a number of different middleware platforms with the possibility of using different SLA languages. The software components required to gather metric data may be automatically derived from SLAs, reducing the need to hand code such components on a per-SLA basis from scratch. We have demonstrated our implementation using a third party SLA language and evaluation tool and gathered metric data from EJB and Web Service components. The way in which MOM may be used as a basis on which to create scalable SLA monitoring implementations is described.

Our future work, in the short term, is concerned with engineering tasks: extending our system to cover additional middleware platforms (e.g., CORBA, .NET) and the inclusion of a variety of SLA languages. In the long term we are seeking to extend our scope of applications to cover interactive media (e.g., online games).

#### **Acknowledgements**

This work is part-funded by the UK EPSRC Grant GR/S63199: "Trusted Coordination in Dynamic Virtual Organisations" and by the European Union under Project IST-2001-34069: "TAPAS (Trusted and QoS-Aware Provision of Application Services)".

## 5. REFERENCES

1. M. Debusmann, A. Keller, "SLA-Driven Management of Distributed Systems Using the Common Information Model", in Proceedings of the 8th IFIP/IEEE IM, 2003
2. C. Overton, "On the Theory and Practice of Internet SLAs", Journal of Computer Resource Measurement 106, 32-45, Computer Measurement Group, 2002
3. A. Habib, S. Fahmy, S. R. Avasarala, V. Prabhakar, B. Bhargava, "On Detecting Service Violations and Bandwidth Theft in QoS Network Domains", Computer Communications, Elsevier, Vol. 26 Issue 8, Pages 861-871, 2003
4. A. Keller, H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", IBM Research Report, 2002
5. R. Schantz, J. Zinky, D. Karr, D. Bakken, J. Megquier, J. Loyall, "An Object-Level Gateway Supporting Integrated-Property Quality of Service", ISORC '99, 1999
6. A. Sahai, V. Machiraju, M. Sayal, L. J. Jin, F. Casati, "Automated SLA Monitoring for Web Services", HP-Labs Report HPL-2002-191, 2002
7. A. Asgari, P. Trimintzios, M. Irons, R. Egan, G. Pavlou, "Building Quality-of-Service Monitoring Systems for Traffic Engineering and Service Management", Journal of Network and Systems Management, Vol. 11, No. 4, 2003
8. J. Pruyne, "Enabling QoS via Interception in Middleware", HP-Labs Report HPL-2000-29, February 2000
9. Keynote Systems, <http://www.keynote.com>, as viewed November 2004
10. JBoss project, <http://www.jboss.org>, as viewed September 2004
11. J. Skene, D. Lamanna, W. Emmerich, "Precise Service Level Agreements", Proceedings of the 26th International Conference on Software Engineering, Pg. 179 – 188, 2004
12. Sun Microsystems, Java Message Service (JMS) Specification, <http://java.sun.com/products/jms>, Version 1.1, 2002
13. C. Molina-Jimenez, S. Shrivastava, J. Crowcroft, and P. Gevros, "On the Monitoring of Contractual Service Level Agreements", In Proceedings of the IEEE Conference on Electronic Commerce CEC\04, San Diego, 2004
14. J. Skene and W. Emmerich, "Model Driven Performance Analysis of Enterprise Information Systems", Electronic Notes in Theoretical Computer Science, 82(6), 2003
15. R. Irani, S. J. Basha, "AXIS: Next Generation Java SOAP", Peer Information; 1st edition, 2002.
16. K. H. Hanse, "Load Testing your Applications with Apache JMeter", Java Boutique Internet, <http://javaboutique.internet.com/tutorials/JMeter/>, as viewed November 2004
17. C. Molina-Jimenez, S. K. Shrivastava, E. Solaiman, J. P. Warne, "Contract Representation for Run-time Monitoring and Enforcement", In Proceedings of the IEEE International Conference on E-Commerce (CEC 2003), California, USA, 24-27 June 2003
18. Arjuna Technologies, "Arjuna Messaging Service", <http://www.arjuna.com/products/arjunams/index.html>, as viewed November 2004.