Implementing the Monitoring of Service Level Agreements

Graham Morgan[1], Simon Parkin[1], Carlos Molina-Jimenez[1], and James Skene[2]
(1)School of Computing Science, University of Newcastle, UK
(2)Department of Computer Science, University College London, UK
{Graham.Morgan, S.E.Parkin, Carlos.Molina }@newcastle.ac.uk, J.Skene@cs.ucl.ac.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. In this paper we describe an implementation that has the aim of providing SLA monitoring services to SLA participants that interact across the Internet. We assume SLA participants may deploy their services using a number of different middleware platforms and may define their SLAs in a number of different ways. Furthermore, as the number of SLAs and associated participants may be large in number, a scalable solution to SLA monitoring is desirable. To avoid the time consuming task of hand coding monitoring software for gathering metric data on a per SLA basis, we automatically generate such software from computer readable SLA specifications.

1. Introduction

Service Level Agreements (SLAs) specify the Quality of Service 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 gain statistical metrics about the performance of a service to determine if the level of Quality of Service (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].

Existing approaches to the monitoring of SLAs by third parties is not well advanced:

- **Ambiguity** – SLAs may appear ambiguous (leading to multiple interpretations) with no indication of how QoS attributes are to be monitored.
- **Lack of generality** – monitoring tends to be platform (middleware) specific and tightly coupled to an SLA language.
- **Poor scalability** – the scalability required to monitor many large numbers of SLAs involving many participants is not properly addressed for many application types (e.g., e-commerce).

Our previous work on the monitoring of SLAs [13] has identified, isolated and reasoned about the basic design issues of monitoring. We 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.

Different contractual requirements between SLA participants have resulted in a number of SLA languages and many applications that require monitoring are deployed using different types of middleware (e.g., Common Object Request Broker Architecture (CORBA), Java 2 Enterprise Edition (J2EE), Web Services). Existing monitoring services are SLA language dependent and middleware dependent, making them unsuitable for deployment over a variety of platforms using a variety of SLA languages. Furthermore, an organisation providing SLA monitoring may be concerned with many hundreds, possibly thousands, of SLAs to ensure a viable business model. This engineering problem of scalability has only been addressed in the context of Internet traffic engineering, and not in the more general case of SLA monitoring associated with inter-organisational issues.

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 is both modular (requires minor tailoring to work with different SLA languages and middleware platforms) and scalable (may scale to satisfy the SLA monitoring requirements of many SLAs and their participants). 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 service provision across the Internet, governed by SLAs described using an existing SLA language, deployed over J2EE and Web Service middleware.

This paper is organized as follows. Section 2 describes background and related work and identifies a number of implementation challenges that, we believe, an SLA monitoring implementation should meet. Section 3 describes our implementation and section 4 provides conclusions and future work.

2. Background & Related Work

This section highlights a number of implementation challenges an organisation will face when delivering SLA monitoring services to clients (providers and consumers). We use a description of our SLA architecture to aid in identifying such challenges and assess how existing implementations are addressing such challenges.

2.1 SLA Monitoring Architecture

The architecture we proposed [13] for monitoring SLAs is shown in Figure 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.

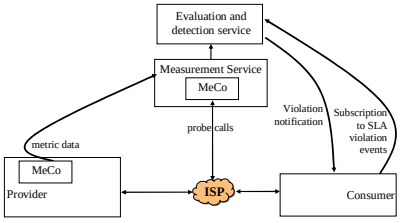

Figure 1 – Architecture for unilateral monitoring of QoS.

The components shown in the diagram that assume responsibility for SLA monitoring are:

- **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** – Determines if SLA violation has occurred via metric data gathered and informs provider/consumer of such violations.

The two MeCos shown in the diagram 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 be disturbed unduly with metric collection responsibilities).

As MeCos directly interact with the observed system they must accommodate whatever middleware platform a provider and consumer are using. Furthermore, MeCos must realise what data to gather. This information is most appropriately drawn from an SLA as it is the SLA that includes all the required information for determining how QoS is related to gathered metric data. This provides our first implementation challenge:

1. *Allow SLA monitoring to occur over a variety of middleware platforms.*

From the viewpoint of an organisation specialising in the provision of SLA monitoring the automated

production of MeCos from SLAs for 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). This provides our second implementation challenge:

2. *Ease the development of a MeCo via automating as much code generation as possible using SLAs as a basis on which such code may be derived.*

MeCos must have a method of communicating the metric data they have gathered to the measurement service. This requires the measurement service to communicate with MeCos that may be geographically distributed across the Internet (e.g., within providers' domain). Assuming monitoring is provided to a number of client organisations, there is a need to utilise an appropriate communication mechanism that is scalable to ensure metric data may be sent to the measurement service in a timely manner and violations may be sent to the appropriate SLA participants. This provides our third implementation challenge:

3. *Ensure metric data and SLA violation notifications may be distributed around the system in a manner that is scalable.*

Once the 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. If this is the case then 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. This provides our fourth implementation challenge:

4. *Allow multiple SLA languages (and associated evaluation and detection service) to be incorporated into a single monitoring implementation.*

We do not state that the four challenges we have identified are the only implementation challenges, but they provide the basis for forming the requirements which we wanted to satisfy in our SLA monitoring implementation (taking the viewpoint of an organisation that delivers SLA monitoring solutions to clients). We continue this section with a discussion of related work and how such work relates to our four implementation challenges.

2.2 Existing Approaches to Implementation

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 , J2EE, 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 (our first implementation challenge).

The process of automated code generation from SLAs for the purposes of SLA evaluation has been achieved (e.g., [6] [11]). However, deriving code that will implement a MeCo suitable for deployment on a specific middleware platform is not yet fully realised. The related work that comes closest to our second implementation challenge is presented in [6] (automated SLA monitoring for web services). Via the use of business management platform (BMP) agents [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. As our third implementation challenge indicates, when delivering SLA monitoring services (even in an e-commerce environment) scalability of message dissemination is desirable. [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, 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 our third implementation challenge may be best served via the use of MOM technologies.

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 meets implementation challenge 4 as all existing SLA monitoring implementations are SLA language specific.

From our discussion of existing approaches to SLA monitoring we may determine that there exists no single implementation that meets our four implementation challenges. However, attempts at automated SLA monitoring [6] and scalable metric data dissemination [7] do provide evidence that at least two of our challenges may be satisfied by existing implementations (albeit confined to specific application domains).

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 figure 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 construction of distributed systems and applications with reliable QoS characteristics. The Unified Modelling 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 clients and their behaviour. Constraints are defined formally using the Object Constraint Language (OCL), providing the semantics. This approach permits natural and economical modelling 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 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 (e.g., how many requests clients are issuing over a period of time). We have developed a way of 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 [11].

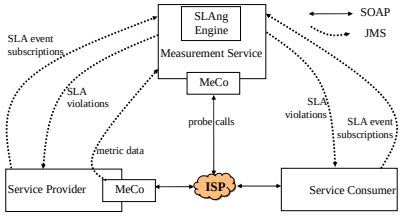

Figure 2 –SLAs monitoring architecture with message oriented middleware.

We assume 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).

The architecture shown in figure 2 alters the architecture shown in figure 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 our implementation influenced diagram shown in figure 2:

- **Service provider MeCo** - This MeCo (metric collector) 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.
- **Measurement service MeCo** – This MeCo observes the performance of 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 collating 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 throughout the system.

The measurement service may be 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. When appropriate, we identify how our implementation attempts to meet the implementation challenges described in section 2. Although our implementation is based on SLAng, J2EE 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 implementation. Our implementation is in Java.

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*. Only the code within the MeCo hooks has to be tailored for specific middleware platforms. MeCo hooks are middleware dependent and are 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 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 Enterprise Java Beans (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 implementing such services achieved using EJBs. We used the JBOSS application server [10] to support EJBs.

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 [12] to implement MeCo hooks suitable for interception of Java RMI invocations. JBoss presents an implementation of the J2EE architecture.

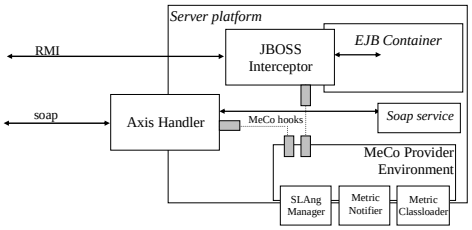

Figure 3 – Service Provider use of MeCos.

Figure 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 (shown in figure 2):

- **SLAng Manager** – Examines an SLA contract file (as used by SLAng engine) to determine the metric data that the MeCo is to observe. The product of parsing an SLA contract is a Java class that may be used for gathering the appropriate metric data. This Java 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 creation of message channels over which metric data will travel.
- **Metric Classloader** – Loads the appropriate classes for implementing the monitoring of the required data as specified by the SLAng manager. Classes are loaded from the class repository. Each class represents a metric type as specified by the SLA language used by the SLAng engine (e.g., response time).

The MeCo provider environment was developed in a modular fashion so the minimum of tailoring was required to make the 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 MeCo hooks require tailoring on a per-middleware basis). For each SLA language a different SLAng manager and class repository is required. This is because such a language must be parsed (by the SLAng manager) and appropriate mechanisms for metric data monitoring realised (by class repository). This approach has the added benefit of allowing our system to be extendable in that any extensions that may be added to an SLA language over time may be incorporated into the MeCo.

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. However, it is perceivable that an organisation specialising in SLA monitoring may wish to make use of complex probing strategies allowed by a tool like JMeter (why we allow this choice of probing strategy creation).

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 etc.) with metric data passed from the MeCo to the measurement service in the same manner as the metric data generated at the service provider MeCo (via messaging channels).

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) was chosen as the message platform.

JMS provides an Application Programming Interface (API) that allows Java developers to integrate MOM into their applications. 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]). As the JMS API is standard, we can use any of these solutions. 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. A client may get all its messages delivered to a queue, allowing a queue to contain a variety of different message types. 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*. The measurement service consumes messages as and when they are published on the metric topics. This is a desirable scenario when we consider a large scale deployment of our monitoring architecture. Assuming we have multiple service providers, there is no need for each service provider MeCo to create a direct communication channel to the messaging service. Requiring a messaging service to manage communication links to hundreds or thousands of service providers is not scalable. A MeCo disseminates metric data by publishing such data on an appropriately named 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 (checkers) is their lack of scalability when faced with checking increasing numbers of SLAs [14]. Therefore, an opportunity to employ additional engines (via additional measurement services) and so improve scalability is desirable in an SLA monitoring implementation. Via this

method we also allow different SLA engines and measurements services (possibly using different SLA languages) to be used in our implementation, meeting one of our implementation challenges.

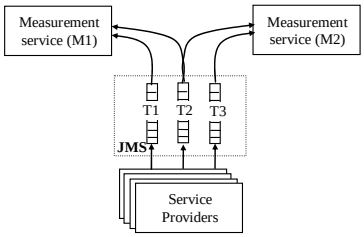

Figure 4 – Message scenario using metric topics in JMS.

Figure 4 illustrates a scenario with multiple service provider MeCos publishing metric data on metric topics T_1, T_2 and T_3. Measurement services M_1 and M_2 are consuming metric data from these metric topics and are responsible for identifying SLA violation. The set of SLAs M_1 is responsible for is different than the set of SLAs M_2 is responsible for, allowing M_1 and M_2 to share the processing load associated with SLA violation for a number of clients. The introduction of additional measurement services 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). As the use of JMS provides loosely coupled communications between MeCos and measurement services, additional measurement services to be added with minimum disruption to the overall function of the system (via subscription to appropriate metric topics by measurement services). 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).

A metric topic message contains the metric ID (unique identifier associated with a particular metric), values monitored (metric data), client ID (unique identifier associated with service consumer) and server ID (unique identifier associated with service provider). The contents of such a message is middleware/SLA language dependent (but could easily be applied to other middleware/SLA solutions).

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 (figure 5):

- **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).

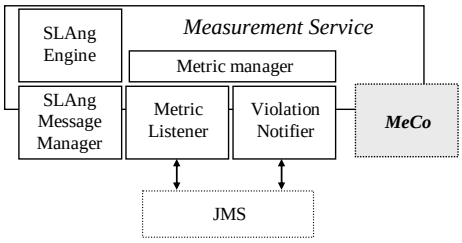

Figure 5 – Measurement service.

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 Java classes, then a metric manager must be developed on a per SLA-language basis. In addition to creating service usage messages, the 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 limited tailoring, provides an opportunity to monitor service provision over a number of different middleware platforms using different SLA language specifications. The software components required to gather metric data may be, partially, automatically derived from SLAs, reducing the need to hard code such components on a per-SLA basis. We have demonstrated our implementation using a third party SLA language and evaluation tool and gathered metric data from EJB and Web Service components. MOM has been used as a basis on which to create scalable SLA monitoring implementations. The design of our system provides an opportunity to utilise multiple SLA engines to gain scalable processing resources suitable for evaluating SLAs.

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., games), where peer-to-peer environments are predominant as opposed to the client/server architectures that we have considered so far.

5. References

[1] Markus Debusmann, Alexander Keller, "SLA-Driven Management of Distributed Systems Using the Common Information Model", in Proceedings of the 8th IFIP/IEEE IM, 2003

[2] Chris Overton, "On the Theory and Practice of Internet SLAs", Journal of Computer Resource Measurement 106, 32-45, Computer Measurement Group, 2002

[3] Ahsan Habib, Sonia Fahmy, Srivinas R. Avasarala, Venkatesh Prabhakar, Bharat Bhargava, "On Detecting Service Violations and Bandwidth Theft in QoS Network Domains", Computer Communications, Elsevier, Vol. 26 Issue 8, Pages 861-871, 2003

[4] Alexander Keller, Heiko Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", IBM Research Report, 2002

[5] Richard Schantz, John Zinky, David Karr, David Bakken, James Megquier, Joseph Loyall, "An Object-Level Gateway Supporting Integrated-Property Quality of Service", ISORC '99, 1999

[6] Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Li Jie Jin, Fabio Casati, "Automated SLA Monitoring for Web Services", HP-Labs Report HPL-2002-191, 2002

[7] Abolghasem Asgari, Panos Trimintzios, Mark Irons, Richard Egan, George 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] Jim Pruyne, "Enabling QoS via Interception in Middleware", HP-Labs Report HPL-2000-29, February 2000

[9] Keynote Systems, http://www.keynote.com, as biewed November 2004

[10] JBoss project, http://www.jboss.org, as viewed September 2004

[11] James Skene, D. Davide Lamanna, Wolfgang 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 (2003), "Model Driven Performance Analysis of Enterprise Information Systems", Electronic Notes in Theoretical Computer Science, 82(6)

[15] R. Irani, S. J. Basha, "AXIS: Next Generation Java SOAP", Peer Information; 1st edition, 2002.

[16] Keld 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