SLA-driven Management of Distributed Systems using the Common Information Model

Markus Debusmann[*]
FH Wiesbaden - University of Applied Sciences
Kurt-Schumacher-Ring 18
65197 Wiesbaden, Germany
E-Mail: *m.debusmann@computer.org*

Alexander Keller
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA
E-Mail: *alexk@us.ibm.com*

Abstract

We present a novel approach of using CIM for the SLA-driven management of distributed systems and discuss our implementation experiences. Supported by the growing acceptance of the Web Services Architecture, an emerging trend in application service delivery is to move away from tightly coupled systems towards structures of loosely coupled, dynamically bound systems to support both long and short term business relationships across different service provider boundaries. Such dynamic structures will only be successful if the obligations among different providers with respect to the quality of the offered services can be unambiguously specified and enforced by means of dynamic Service Level Agreements (SLAs). In other words, the management of SLAs needs to become as dynamic as the underlying infrastructure for which they are defined.

Our work has shown that Web Services, as a typical example for a service-oriented architecture, can be extended in a straightforward way for defining and monitoring SLAs. On the other hand, SLAs defined for a Web Services environment need to take into account the underlying managed resources whose management interfaces are defined based on traditional management architectures, such as SNMP-based management or the Common Information Model (CIM). As a solution to this problem, the approach presented in this paper addresses the integration problem of how to transform a Web Services SLA so that it can be understood and enforced by a service provider whose management system is based on a traditional management architecture, such as CIM.

Keywords

Service Level Agreements, Web Services, Common Information Model, Inter-Domain Management

[*]Work done while author was an intern at IBM T.J. Watson Research Center.

1 Introduction and Problem Statement

Over the last year, emerging component based service architectures built on top of Web Services [9], such as the *Open Grid Services Architecture (OGSA)* [5, 16], have been gaining increasing acceptance beyond computing-intense scientific and commercial applications: It appears highly likely that the next generation of e-Business systems will consist of an interconnection of services, each provided by a possibly different service provider, that are put together "on demand" to offer an end-to-end service to a customer. Such an environment – referred to as 'Computing Grid' [4] – will be administered and managed according to dynamically negotiated Service Level Agreements (SLA) between service providers and customers [10, 17]. Consequently, systems management will increasingly become SLA-driven and needs to address challenges such as dynamically determining whether enough spare capacity is available to accomodate additional SLAs, the negotiation of SLA terms and conditions, the continuous monitoring of a multitude of agreed-upon SLA parameters and the troubleshooting of systems, based on their importance for achieving business objectives. A key prerequisite for meeting these goals is to understand the relationship between 'high-level' SLA parameters and 'low-level' resource metrics, such as counters and gauges. However, mapping SLA parameters onto metrics that are retrieved from managed resources is a difficult problem [11].

This paper presents our approach for mapping SLAs, defined using the *Web Service Level Agreement (WSLA)* framework (described in section 2), which is based on the Web Services Architecture, onto the Common Information Model (CIM)[2]. Thus, the work described in this paper can be regarded as a precursor to future work on integrating emerging service architectures with traditional enterprise management frameworks. The novelty of our approach lies in the way we address the following key questions; these questions also reflect the structure of this paper:

1. **How can SLA parameters be mapped onto resource metrics?**
   At the core of our approach to this problem is the WSLA language that allows a party involved in the establishment of an SLA to define what is actually meant by an SLA parameter. Instead of merely assigning thresholds to pre-defined SLA parameters, whose semantics vary greatly [1], the WSLA framework, presented in section 2, allows the precise definition of how SLA parameters are supposed to be computed and aggregated.

2. **What SLA monitoring components ought to be implemented as Web Services? For which components is CIM the better answer?**
   Based on an inter-domain SLA management scenario, section 2 breaks down the SLA monitoring process into a set of elementary services needed to enable the management of an SLA throughout the various phases of its lifecycle. Since we are dealing with a service architecture and a resource management architecture, every service may be implemented either as a Web Service or based on CIM. An analysis and an evaluation of the various options is given in section 3.1.

3. **Which parts of the SLA should be modeled in CIM and how does a suitable model look like?**
   While this question is closely related to the previous one, there are a few additional implications a suitable CIM model for SLAs needs to take into account: In particular, the CIM model needs to provide a means for keeping data that relates to the definition aspects of the SLA while being able to measure and store the actual SLA paremeter and metric values at runtime. Stated differently, the measured values need to be tied back to their definitions and to the SLA in which they are defined. Our solution to this problem, based on the CIM Metrics Model, is described in the remainder of section 3.

4. **How can one delegate management functions to an agent in a WBEM/CIM environment?**
   Traditionally, the purpose of CIM subagents (termed "providers") is to make the instrumentation of managed resources accessible to a CIM Object Manager (CIMOM). Providers respond to incoming requests, retrieve the requested management information and return the results to a CIMOM. Thus, they play a passive role in the management process by reacting to requests coming from a CIM client.

Since an SLA is usually associated with a schedule that indicates precisely when and how often the measurements are supposed to be taken, a CIM provider needs to take an active role when carrying out its measurements. Our approach to this classical problem of (statically, in our case) delegating measurement functionality to agents [18], with a specific focus on SLAs and CIM, is described in section 4.

5. **Finally, how can one achieve Interoperability between the Web Services Architecture and CIM?** This is obviously a very broad question, for which a generic approach is likely to be as complex as the well-known approaches for achieving interoperability between traditional management architectures (for an in-depth discussion of this subject, see [14, 15]). Nevertheless, we have designed and implemented a mechanism for deploying SLAs from a Web Service environment into a CIMOM and a way to deliver measurements from a CIMOM back to a Web Service. Our experiences with the proof-of-concept implementation are described in section 5. Our work can be regarded as a precursor to future work dealing with the development of generic mechanisms for integrating Web Services based management with existing management infrastructures.

2 The Web Service Level Agreement (WSLA) Framework

This section describes our work towards a flexible SLA monitoring framework, targeted at Web Services. We have defined and implemented the *Web Service Level Agreement (WSLA)* framework [7] for defining and monitoring SLAs in inter-domain environments. In [6], we have described the concepts behind WSLA. Although it is not the purpose of this paper to describe WSLA in detail, we need to provide a brief overview over WSLA and its principles to set the stage for our CIM based SLA model detailed in section 3 and the architecture of our solution described in section 4.

Our approach to enable SLA-driven Management of distributed and highly dynamic systems, WSLA, consists of a flexible and extensible language [12] based on the XML schema and a runtime architecture based on several SLA monitoring services, which may be outsourced to third parties to ensure a maximum of accuracy. WSLA enables service customers and providers to unambiguously define a wide variety of SLAs, specify the SLA parameters and the way how they are measured, and tie them to managed resource instrumentations. A Java-based prototype implementation of the WSLA framework, termed *SLA Compliance Monitor*, is included in the current version 3.2 of the publicly available IBM Web Services Toolkit [1].

2.1 SLA Lifecycle in the WSLA Runtime Architecture

Figure 1 depicts the typical lifecycle of an SLA in a multi-provider environment. The lifecycle consists of the following straightforward phases: SLA creation, SLA deployment, SLA execution, and SLA termination. For the sake of brevity, the latter is not depicted in the figure.

The SLA creation process involves the negotiation and signing of an SLA by both a service provider and service customer. During this process, a customer retrieves the metrics offered by a provider, aggregates and combines them into various SLA parameters, defines service levels for every SLA parameter, and submits the SLA to the service provider for approval. On the side of every *signatory party* (a party that signs an SLA) a **Business Entity** carries out the negotiation: It embodies the business knowledge, goals and policies of a party. Such knowledge enables the business entity to decide which service levels should be specified in the SLA to ensure compliance with its business goals. A typical example for such a decision on the service customer side is to define thresholds for response times or throughput, depending on the price the customer is willing to pay. On the provider side, typical business actions are to decide if the SLA is acceptable as a whole or whether the customer-specified thresholds are too restrictive. Once agreement on the main elements of the SLA is reached, customer and provider may define third parties (which we call *supporting*

[1]The IBM Web Services Toolkit can be downloaded from http://www.alphaworks.ibm.com/tech/webservicestoolkit.

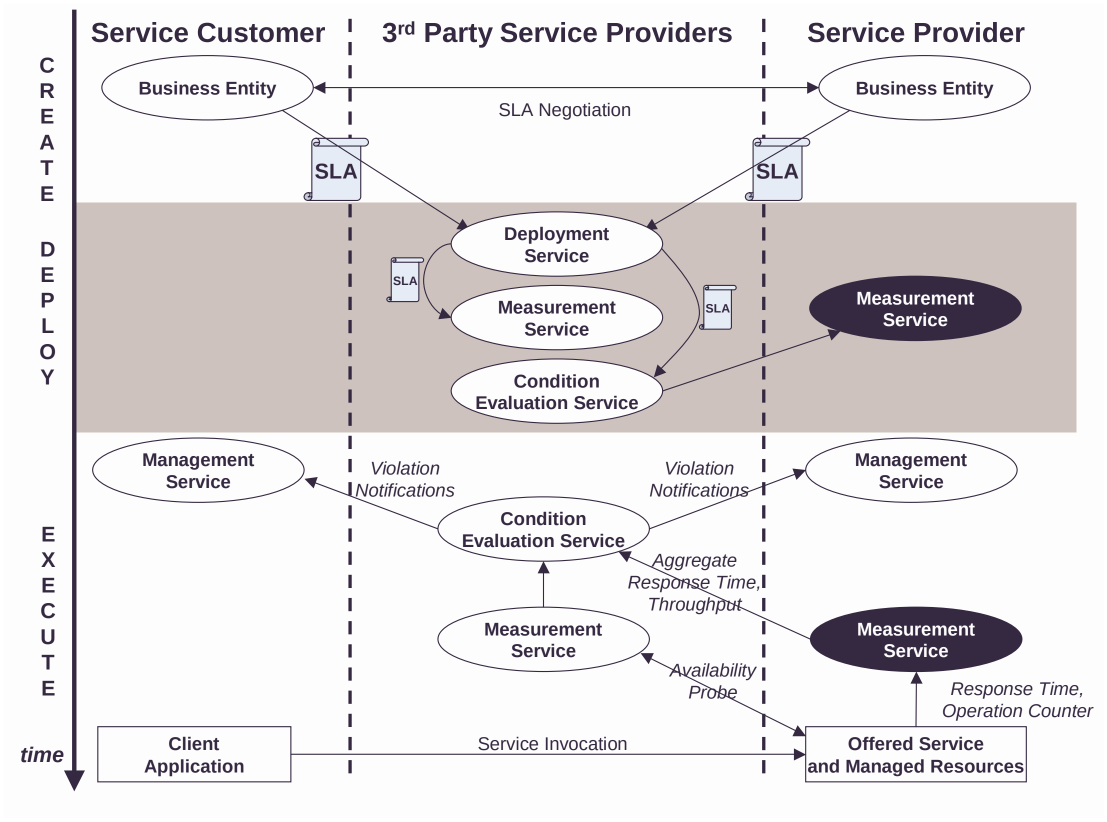

Figure 1: Lifecycle of a Service Level Agreement in a Multi-Provider Environment

parties in the WSLA context), to which SLA monitoring tasks may be delegated. Supporting parties come into play when either a function needs to be carried out that neither service provider nor customer wants to do, or if these signatory parties do not trust their counterparts to perform a function correctly.

Once the SLA is finalized, both service provider and service customer make the SLA document available for deployment. The **Deployment Service** is responsible for checking the validity of the SLA and distributing it either in full or in appropriate parts to the supporting parties. The latter is needed to ensure that a supporting party service receives only the amount of information it needs to carry out its tasks.

In the scenario, we assume that a part of the SLA monitoring and supervision activities is delegated to third party service providers. Typical services that may be outsourced to third parties fall into two categories:

- **Measurement Service:**
  The Measurement Service maintains information on the current system configuration, and runtime information on the metrics that are part of the SLA. It measures SLA parameters such as availability or response time either from inside, by retrieving raw metrics directly from managed resources, or outside the service provider's domain, e.g., by probing or intercepting client invocations. A Measurement Service may measure all or a subset of the SLA parameters. Multiple measurement services may simultaneously measure the same metrics, e.g., a measurement service may be located within the provider's domain while another measurement service probes the service offered by the provider across the Internet from various locations. For our discussion, we call metrics that are retrieved directly from managed resources **Raw Metrics**. **Composite Metrics**, in contrast, are created by aggregating several raw (or other composite) metrics according to a specific algorithm, such as averaging

one or more metrics over a specific amount of time or by breaking them down according to specific criteria (e.g., top 5%, minimum, maximum, average etc.). This is usually being done within a service provider's domain (depicted in figure 1 as the oval having a black background), but can be outsourced to a third-party measurement service as well (measurement service with white background). Keynote Systems, Inc. [8] is an example of such an external measurement service provider. In sections 4 and 5, we will describe our approach to implementing an internal measurement service (black oval in the figure) in CIM and how it accesses managed resource instrumentation.

- **Condition Evaluation Service:**
  This service is responsible for monitoring compliance of the SLA parameters with the agreed-upon Service Level Objective (SLO) by comparing measured parameters against the thresholds defined in the SLA and notifying the management services of the service customer and provider. It obtains measured values of SLA parameters from the Measurement Service and tests them against the guarantees given in the SLA. This can be done each time a new value is available, or periodically.

Finally, both service customer and provider have a **Management Service**: Upon receipt of a notification, the management service (usually implemented as part of a traditional management platform) will retrieve the appropriate actions to correct the problem, as specified in the SLA. The main purpose of the management service is to execute corrective actions on behalf of the managed environment if a Condition Evaluation Service discovers that a term of an SLA has been violated.

2.2 Expressing SLAs in WSLA: The main Elements of the WSLA Language

In this section, we provide a brief overview over the WSLA language. For a more detailed discussion, the reader is referred to [13].

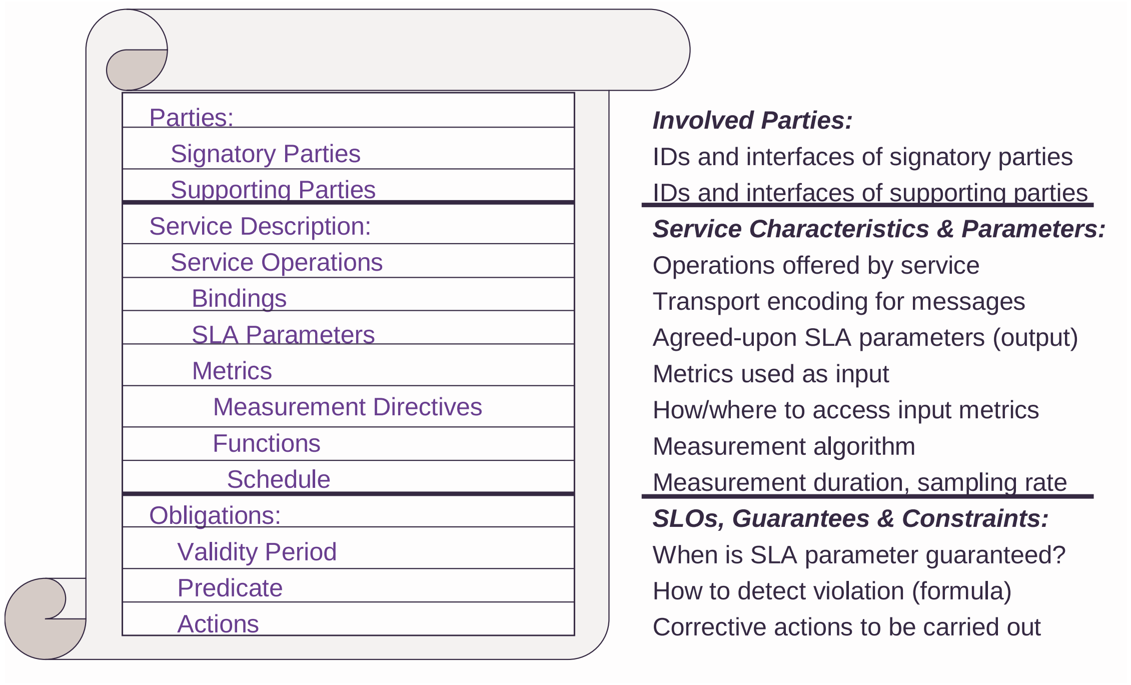
Parties:
Signatory Parties
Supporting Parties
Service Description:
Service Operations
Bindings
SLA Parameters
Metrics
Measurement Directives
Functions
Schedule
Obligations:
Validity Period
Predicate
Actions
Involved Parties:
IDs and interfaces of signatory parties
IDs and interfaces of supporting parties
Service Characteristics & Parameters:
Operations offered by service
Transport encoding for messages
Agreed-upon SLA parameters (output)
Metrics used as input
How/where to access input metrics
Measurement algorithm
Measurement duration, sampling rate
SLOs, Guarantees & Constraints:
When is SLA parameter guaranteed?
How to detect violation (formula)
Corrective actions to be carried out

Figure 2: Typical Structure of an SLA

Figure 2 illustrates the typical elements of an SLA with signatory and supporting parties. Clearly, there are many variations of what types of information and which rules are to be included and, hence, enforced in a specific SLA. The **Parties** section, consisting of the signatory parties and supporting parties fields identify all the contractual parties. **Signatory Party** descriptions contain the identification and the technical properties of a party, i.e., their interface definition and their addresses. The definitions of the **Supporting Parties** contain, in addition to the information contained in the signatory party descriptions, an attribute indicating the sponsor(s) of the party.

The **Service Description** section of the SLA specifies the characteristics of the service and its observable parameters as follows: For every **Service Operation**, one or more **Bindings**, i.e., the transport encoding for the messages to be exchanged, may be specified. In addition, one or more **SLA Parameters** of the service may be specified. Examples of such SLA parameters are *service availability*, *throughput*, or *response time*. Every SLA parameter refers to one **Metric**, which, in turn, may aggregate one or more other (composite or raw) metrics, according to a measurement directive or a function. Examples of composite metrics are *maximum response time of a service*, *average availability of a service*, or *minimum throughput of a service*. Examples of raw metrics are: *system uptime*, *service outage period*, *number of service invocations*. **Measurement Directives** specify how an individual raw metric can be obtained from a managed resource. Typical examples of measurement directives are the uniform resource identifier of a hosted computer program, a protocol message, or the command for invoking scripts or compiled programs. **Functions** are the measurement algorithm, or formula, that specifies how a composite metric is computed. Examples of functions are formulas of arbitrary length containing average, sum, minimum, maximum, and various other arithmetic operators, or time series constructors. For every function, a **Schedule** is specified. It defines the time intervals during which the functions are executed to compute the metrics. These time intervals are specified by means of *start time*, *duration*, and *frequency*. Examples of the latter are *weekly*, *daily*, *hourly*, or *every minute*.

Obligations, the last section of an SLA, define the SLOs, guarantees and constraints that may be imposed on the SLA parameters: First, the **Validity Period** is specified; it indicates the time intervals for which a given SLA parameter is valid, i.e., when the SLO may be applied. Examples of validity periods are *business days*, *regular working hours* or *maintenance periods*. The **Predicate** specifies the threshold and the comparison operator (greater than, equal, less than, etc.) against which a computed SLA parameter is to be compared. The result of the predicate is either *true* or *false*. **Actions**, finally, are triggered whenever a predicate evaluates to *true*, i.e., a violation of an SLO has occurred. Actions are e.g., *sending an event to one or more signatory and supporting parties*, *opening a trouble ticket or problem report*, *payment of penalty*, or *payment of premium*. Note that, as stated in the latter case, a service provider may very well receive additional compensation from a customer for exceeding an obligation, i.e., obligations reflect constraints that may trigger the payment of credits from any signatory party to another signatory or supporting party. Also note that these actions may be individually specified for every SLA parameter.

3 Integrating WSLA and CIM

This section describes the integration of WSLA and CIM. After discussing different integration alternatives, we present our CIM model for representing SLAs and the principles of metric computation and aggregation.

3.1 Implementing SLAs in a Web Services and CIM Environment: Design Considerations

Considering a SLA management environment as shown in figure 1 raises the question how the five services are integrated in the most efficient way. Today, the business entity is normally a human being and therefore not subject to implementation. The management service represents the management platform run by the service provider and customer. Thus, the key question for integrating WSLA and CIM is: Which of the remaining services (deployment, measurement, condition evalution) should be implemented as a Web

Service and which services ought to be implemented in CIM? Three alternatives can be considered:

1. The first approach is to implement the services entirely in a Web Services environment. This obviously simplifies the integration of the components and the delegation of services to third party providers; however, the integration with a management platform and today's managed resources is challenging as none of them have a management interface based on Web Services. Therefore, a pure Web Services based solution is highly unlikely.

2. Implementing all services on a CIM basis is the other extreme. This simplifies the integration with the managed resources. However, if certain services are delegated to third party providers, this solution makes assumptions about the management infrastructure of these service providers and thus limits the flexibility of the overall SLA management system.

3. For maximum flexibility, it is crucial to find the right balance between those two extremes. In our solution we chose the measurement service to be CIM based (depicted as a black oval in figure 1), which highly simplifies the binding between high-level SLA parameters and low-level resource metrics as well as the integration with management platforms. Making the condition evaluation service a Web Service offers high flexibility to delegate this service to third party providers. The deployment service is the gateway between both worlds by offering a Web Services interface for deploying the SLA. The backend side the deployment service is able to communicate with the CIM based measurement service in order to setup the measurements defined in the SLA.

3.2 Representing SLAs in CIM

For SLA-driven management, a formal representation of the agreed-upon definitions in a Service Level Agreement is required such that a management system is able to automatically monitor and, in a subsequent step, enforce the SLA.

Figure 3 depicts the CIM model to represent Service Level Agreements conforming to WSLA (see section 2). As a result of the decision to implement the measurement service in CIM, the information model reflects only the parts of a WSLA document that define the relationships of SLA parameters to low-level resource metrics, the defined functions and the schedule for their retrieval.

The central model element is the `IBM_SLA` class which ties together all the other elements comprising the SLA. `IBM_SLA` itself is inherited from `CIM_ManagedElement`. Several instances of `IBM_SLA` can exist in parallel, thereby representing SLAs of different customers or even several SLAs of one individual customer.

The model strictly distinguishes between metric definitions and metric instances as defined in the CIM Metrics Schema. This has the following advantages:

1. keeping both definitions and values together and thus linking information from the deployment and runtime stages,

2. leveraging the power of CIM queries for SLA retrieval, e.g., retrieve all SLA parameters for a given SLA instance,

3. enabling the service provider to develop a collection of common-off-the-shelf service metric definitions that can be reused for different customers.

The `IBM_MetricDefinition` is the central class for defining metrics of an SLA. This class inherits from the `CIM_BaseMetricDefinition` class.

We distinguish between raw metrics, composite metrics, and time series. As discussed in section 2.1, raw metrics are simple resource metrics that are directly retrieved from the managed resources, e.g., this are counters retrieved by SNMP, or values retrieved from a Web Service. Composite metrics are complex metrics computed by the SLA measurement service itself. The `IBM_ArithmeticMetric` class connects two

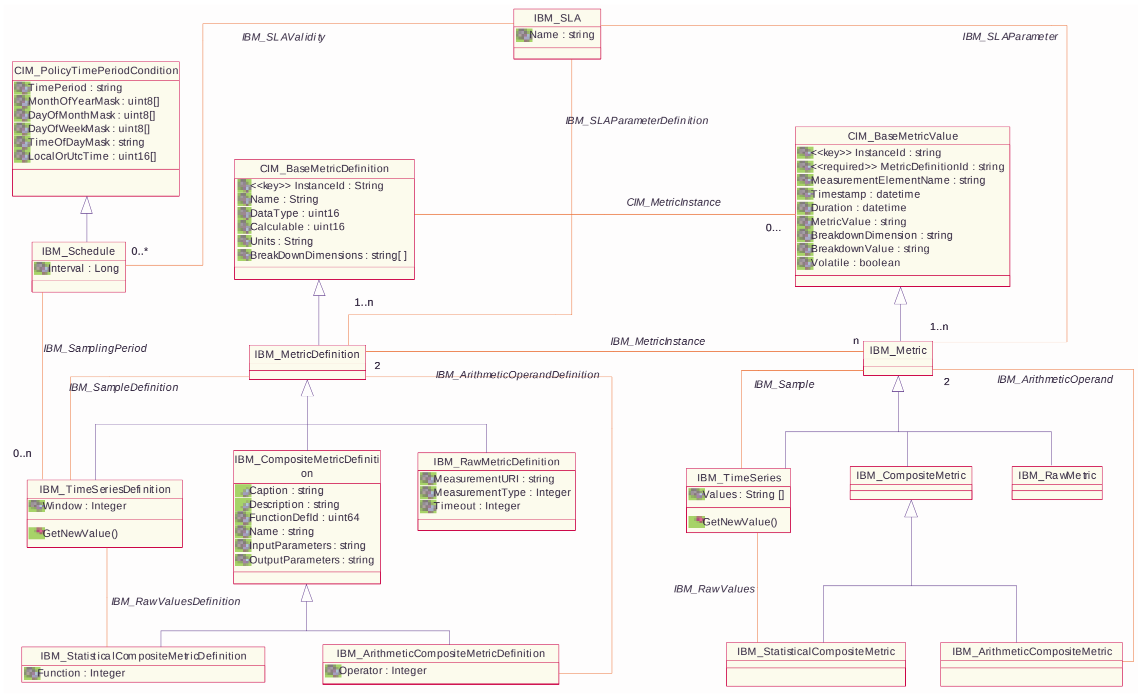

Figure 3: CIM Model of a Service Level Agreement

metrics through an arithmetical operator (+,-,*,/). The `IBM_StatisticalMetric` class performs basic statistical functions, like average, min, max, etc., on collections of metric values. Metric collections are stored using the `IBM_TimeSeries` class, which holds metric values sampled in regular intervals. `TimeSeries` instances may be used as input for several statistical metrics and thus ensure the integrity of data by providing a shared basis for statistical calculations and reduce redundancy. These metric types allow the definition of complex service parameters, such as the average utilization of network interfaces or the maximum reponse time within the last hour.

To efficiently measure SLA parameters, metric values have to be retrieved automatically. The frequency of the sampling is defined as part of the SLA and represented by the `IBM_Schedule` class with properties representing a start date, an end date, and a sampling interval in which a new metric value is stored in the time series (cf. CIM association `IBM_SamplingPeriod`). During runtime, instances of `IBM_Schedule` actively trigger the retrieval of metric values (see section 3.3 and 4.2). The other metrics are calculated on demand when they are requested either from a CIM client or during the calculation process described in the following section.

3.3 Metric Computation and Aggregation

Figure 4 depicts the runtime relationships between metric instances comprising a simple SLA. Two separate activation mechanisms for calculating the metrics exist: timer-triggered (solid arrows) and request-triggered (dashed arrows). In regular time intervals, a `Schedule` instance initiates the collection of a new metric value for a `TimeSeries` object by invoking its `GetNewValue()` CIM method. This causes the collection of the `ArithmeticCompositeMetric` associated with the `TimeSeries`, which is done by means of the CIM operation `getInstance` defined in [3]. Before the `ArithmeticCompositeMetric` instance can be calculated, its associated `RawMetrics` have to be retrieved. After the calculation is done the result is given back and finally stored within the `TimeSeries` object.

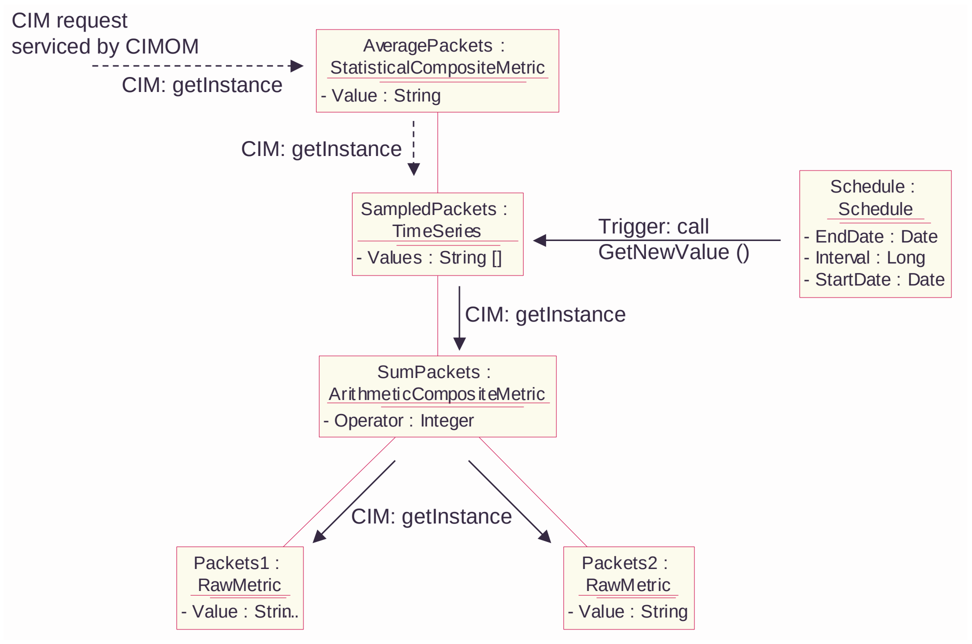

Figure 4: Example of aggregating raw metrics into composite metrics

The second possible activation mechanism is a CIM request from a CIM client. In our example, a request for the `StatisticalCompositeMetric` is handled, thus the associated `TimeSeries` instance has to be retrieved. After that, the average value is calculated based on the values retrieved from the `TimeSeries` object.

4 Architecture of a CIM based SLA Measurement Service

This section addresses the architecture of our CIM based SLA measurement service. First, an overview of the components is given; in the subsequent sections, the concept of active providers and the recovery mechanism, which is needed to reactivate the providers after a restart of the measurement service, are described in detail.

4.1 Overview

Figure 5 depicts the architecture of our CIM based SLA measurement service. Since the CIMOM is the central component responsible for realizing the measurement service, the SLA structure has to be mapped first onto a CIM model, as described in section 3.2. This model has to be loaded once into the CIMOM (1) and appropriate providers implementing this information model have to be developed. Since all SLAs have a common structure, this CIM model is the basis for all SLAs deployed to the CIMOM and is stored in the class repository.

After the SLA model is loaded, SLAs of different customers can be deployed (2). In our implementation, the deployment of SLAs is realized as a custom-based solution that fits the WSLA environment (cf. section 3.1). Therefore, the deployment service offers a Web Services interface for receiving new WSLA documents, i.e., agreed-upon SLAs signed by the signatory parties. After receiving an SLA, the Web Service backend uses an XML parser to analyse and process the SLA. If the document is valid, the backend sends a series of CIM

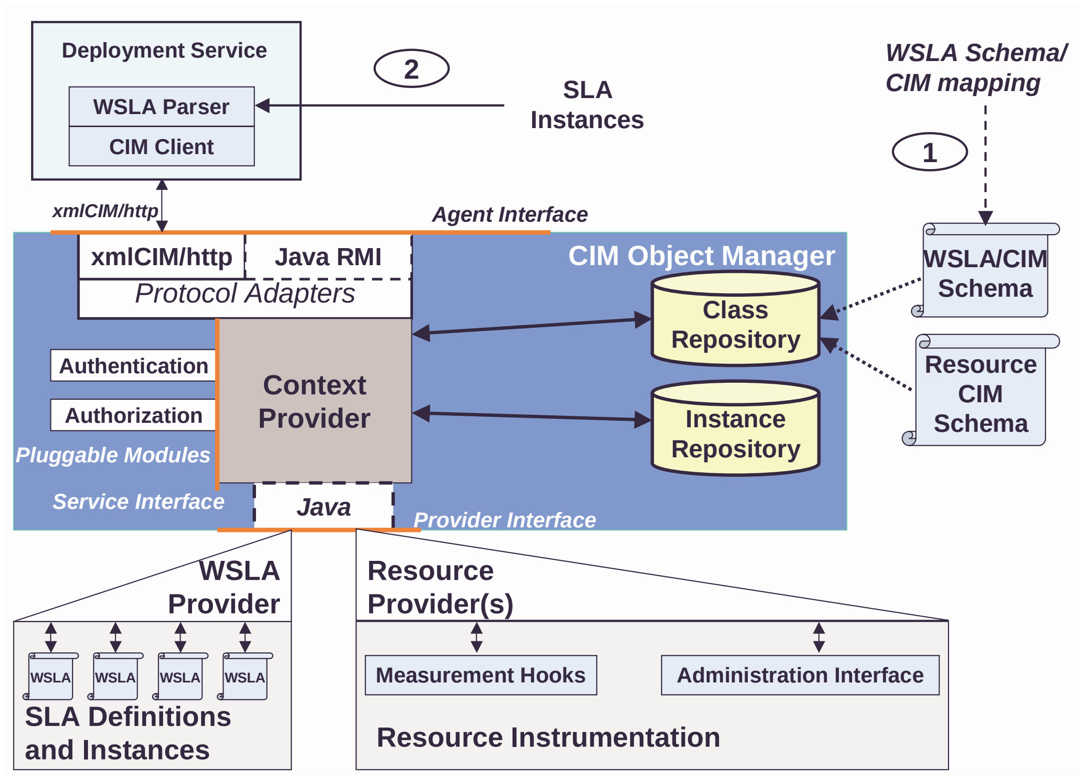

Figure 5: Architectural Overview of the CIM based SLA Measurement Service

requests to the CIMOM to create the CIM instances and associations representing the structure of the SLA. This leads to the instantiation of all the "definition" classes, depicted on the left side of figure 3.

The management information of an SLA is offered by the WSLA provider. Logically, the CIMOM distinguishes between two types of providers: A WSLA provider and one or more resource providers. The WSLA provider implements the CIM classes responsible for representing the structure of an SLA. These CIM instances – like most instances in today's CIM environments – are not subject to change after their deployment. The resource providers are responsible for retrieving raw metrics from managed resources. In the next step, the classes of the WSLA provider that relate to runtime metrics (depicted on the right side of figure 3) calculate the SLA metrics. They are either triggered by `IBM_Schedule` instances or CIM requests, respectively.

4.2 Active CIM Providers

One of the major novelties of our approach is the use of active CIM providers. Until now, CIM providers are stateless resource providers, i.e., they are passive and only surface information from managed resources without providing sophisticated processing capabilities. Normally, the retrieval of information is initiated by the management system. Stateless providers may cause a considerable overhead, e.g., by requiring polling of new values from management applications.

In our implementation, the providers actively monitor the SLA by autonomously retrieving metric values from managed resources and calculating high-level SLA parameters without being triggered by a management application, whose request is then dispatched by the CIMOM to the appropriate provider. The retrieval of new metrics is automatically requested by the provider implementing the `IBM_Schedule` class of the information model (see section 3.3). Making use of this delegated management functionality the overhead caused by polling can be decreased significantly.

4.3 Recovery Mechanism

SLA management requires the continuous monitoring of SLA parameters and metrics. Thus, it must be ensured that the providers remain active in order to perform their measurements. Since the CIM providers are not loaded automatically, but rather on demand, there is the potential problem that the providers are not reactivated after a restart of the measurement service. Since the monitoring is triggered by the provider implementing the `IBM_Schedule` class, it has to be ensured that the provider implementing this class is properly reactivated.

In particular, two different cases have to be considered: First, the deployment of a new SLA and, second, the restart of the CIMOM after a failure. When a new SLA is deployed to the CIMOM, the `IBM_Schedule` provider is loaded, since every SLA contains one or more instances of the `IBM_Schedule` class. Thus, the first case is not a cause for concern. However, in case of a CIMOM restart the providers are not loaded automatically because in typical CIMOM implementations, providers are only loaded when an (external) management application requests data from the CIMOM that needs to be supplied by the specific provider. Thus, a certain recovery procedure has to be applied to ensure that the `IBM_Schedule` provider is loaded and the instances are activated to trigger the data collection for the `IBM_TimeSeries` instances. This can be achieved by forcing the CIMOM to load the provider by having a simple CIM client that enumerates the `IBM_Schedule` instances once the CIMOM has started. Such a command can easily be integrated into the startup script of the CIMOM and thus is no limitation of our approach.

5 Prototype Implementation

The prototype has been implemented using the SNIA (Storage Networking Industry Association) CIMOM v0.7 and the Java Development Kit (JDK) v1.3.1. In principle, every CIM class – as well as the associations – is implemented by a separate provider. This ensures a high flexibility if the information model is undergoing changes and reduces the complexity of individual providers. Encapsulating basic provider functionality in a base class minimizes the implementation effort for provider developers.

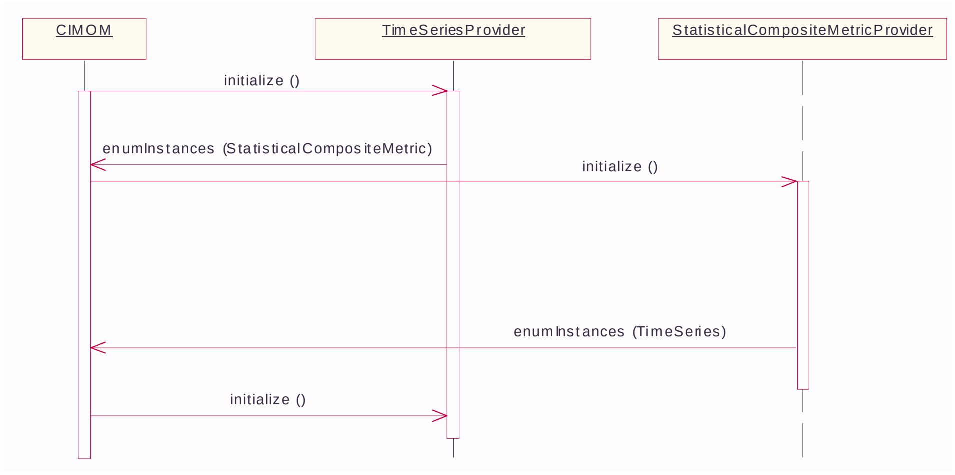
CIMOM
TimeSeriesProvider
StatisticalCompositeMetricProvider
initialize ()
enumInstances (StatisticalCompositeMetric)
initialize ()
enumInstances (TimeSeries)
initialize ()

Figure 6: Deadlock between different CIM provider classes

However, the principle of implementing every CIM provider on a per-class basis could not be hold up for the implementation of the `IBM_ArithmeticCompositeMetric`, `IBM_StatisticalCompositeMetric`, and `IBM_TimeSeries` classes because of cyclic interdependencies between them (cf. fig. 3). Consider the case when the CIMOM is restarted after a crash and the persistent instances have to be reloaded in order to actively monitor the conformance of the deployed SLA(s). Each CIM instance of these metric classes is associated with a calculation object that performs the retrieval of raw metric values and the metric calculation. These calculation objects have to be initialized with the references to their associated objects. Assuming an `IBM_TimeSeries` instance has to be initialized, its associated class can be either of type `IBM_RawMetric`, `IBM_ArithmeticCompositeMetric`, or `IBM_StatisticalCompositeMetric`. If an operand is either an `IBM_ArithmeticCompositeMetric` or an `IBM_StatisticalCompositeMetric`, the attempt to resolve the reference to this instance would result in a request to the corresponding provider without finishing the initialization of the `IBM_TimeSeries` instances first (cf. fig. 6). During the initialization of the instances of the other provider, a single reference to an `IBM_TimeSeries` instance would result in an attempt to initialize the `IBM_TimeSeries` provider again, thus ending up in an infinite recursion.

Combining the initially three separate providers into a single provider, now responsible for handling all three CIM classes, solves this deadlock situation because all CIM instances are restored together from the persistent storage. By doing so, the provider has all information available internally to resolve the references without having to rely on other CIM providers. In case of deploying a new SLA, this problem does not occur because the deployment service creates the instances in the correct order.

6 Conclusions and Outlook

We have presented a novel approach for SLA-driven management of distributed systems using CIM. Our approach uses the WSLA framework to define and formally represent Service Level Agreements as XML documents. To be able to connect to managed resources, these documents need to be transformed into a representation compatible with typical management architectures. In our case, we assume a CIM based management environment.

Our approach to solving this problem consists in developing a CIM based model for SLAs that preserves the expressiveness of WSLA, which is populated by a deployment service whenever a new SLA needs to be monitored by our system. The deployment service is implemented as a Web Service and is able to perform the mapping of the SLA into a CIM based environment. As a result of the deployment, the CIM based measurement service instantiates CIM objects representing the SLA definitions. During the runtime phase, new metric data is collected from the managed resources and subsequently aggregated into higher-level SLA parameters, according to the schedule defined in the SLA. This implies that the data collection is triggered from within the CIM based environment by means of so-called active CIM providers. This new concept goes beyond common CIM providers, which are passive and need to be triggered by external management applications. The move from traditional stateless CIM providers to active CIM providers required the development of new mechanisms, since typical CIMOM implementations are not geared towards active providers; in particular, we had to address the problem of ensuring the smooth recovery after a CIMOM failure. Other problems were caused by the interdependencies between different CIM providers, for which we were able to find a solution.

While our work shows that it is possible to implement more advanced functionality – such as data collection capabilities – with CIM object managers, a more general approach to CIM provider initialization is needed to avoid potential deadlocks. In addition, the customized mapping of WSLA documents into the CIM schema needs to be expanded to support the full interoperability of CIM with a Web Services environment. Recent OASIS work on a new management protocol may be a first step to address this problem. However, past experiences in the network management community with work on achieving interoperability between management architectures suggest that the real issue consists in finding a mapping between the various information models that preserves their semantics.

Acknowledgments

The authors express their gratitude to Bob Moore and Lee Rafalow of IBM for helpful discussions and ongoing advice and for their help to making this work succeed.

References

[1] ASP Industry Consortium. *White Paper on Service Level Agreements*, 2000.

[2] Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999. http://www.dmtf.org/standards/cim_spec_v22/.

[3] Specification for CIM Operations over HTTP, Version 1.1. Specification, Distributed Management Task Force, May 2002. http://www.dmtf.org/standards/documents/WBEM/DSP200.html.

[4] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[5] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Service Architecture for Distributed Systems Integration. Draft, Globus Project, July 2002. http://www.globus.org/research/papers/ogsa.pdf.

[6] A. Keller, G. Kar, H. Ludwig, A. Dan, and J.L. Hellerstein. Managing Dynamic Services: A Contract based Approach to a Conceptual Architecture. In R. Stadler and M. Ulema, editors, *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS'2002)*, pages 513–528, Florence, Italy, April 2002. IEEE Press.

[7] A. Keller and H. Ludwig. Defining and Monitoring Service Level Agreements for dynamic e-Business. In A. Couch, editor, *Proceedings of the 16th System Administration Conference (LISA'02)*, Philadelphia, PA, USA, November 2002. The USENIX Association.

[8] *Keynote – The Internet Performance Authority*. http://www.keynote.com.

[9] H. Kreger. *Web Services Conceptual Architecture 1.0*. IBM Software Group, May 2001.

[10] L. Lewis. *Managing Business and Service Networks*. Kluwer Academic Publishers, 2001.

[11] L. Lewis and P. Ray. On the Migration from Enterprise Management to Integrated Service Level Management. *IEEE Network*, 16(1):8–14, January 2002.

[12] H. Ludwig, A. Dan, R. Franck, A. Keller, and R.P. King. *Web Service Level Agreement (WSLA) Language Specification*. IBM Corporation, July 2002.

[13] H. Ludwig, A. Keller, A. Dan, and R.P. King. A Service Level Agreement Language for dynamic electronic Services. In M. Bichler, editor, *Proceedings of the 4th International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (WECWIS'02)*, Newport Beach, CA, USA, June 2002. IEEE Computer Society.

[14] S. Mazumdar. Inter-Domain Management between CORBA and SNMP. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, L'Aquila, Italy, October 1996.

[15] N. Soukouti and U. Hollberg. Joint Inter-Domain Management: CORBA, CMIP and SNMP. In A. A. Lazar and R. Saracco, editors, *Proceedings of the 5th International IFIP/IEEE Symposium on Integrated Management (IM)*, pages 153–164, San Diego, USA, May 1997.

[16] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid Service Specification. Draft 3, Global Grid Forum, July 2002. http://www.gridforum.org/ogsi-wg/GS_Spec_draft03_2002-07-17.pdf.

[17] D. Verma. *Supporting Service Level Agreements on IP Networks*. Macmillan Technical Publishing, 1999.

[18] Y. Yemini, G. Goldszmidt, and S. Yemini. Network Management by Delegation. In I. Krishnan and W. Zimmer, editors, *Proceedings of the Second International Symposium on Integrated Network Management*, pages 95–107. Elsevier Science Publishers B. V. (North Holland), April 1991.