SLA-DRIVEN MANAGEMENT OF DISTRIBUTED SYSTEMS USING THE COMMON INFORMATION MODEL

Markus Debusmann[1*], Alexander Keller[2]
[1] *FH Wiesbaden - University of Applied Sciences, Department of Computer Science*
Kurt-Schumacher-Ring 18, 65197 Wiesbaden, Germany
m.debusmann@computer.org
[2] *IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA*
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 of 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 previous 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. However, 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: SLA, Web Services, Common Information Model, Inter-Domain Management

1. Introduction and Problem Statement

Over the last year, emerging component based service architectures built on top of Web Services [12], such as the *Open Grid Services Architecture (OGSA)* [7, 20], 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' [6] – will be administered and managed according to dynamically negotiated Service Level Agreements (SLA) between service providers and customers [13, 21]. Consequently, systems ma-

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

nagement 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 (e.g., Availability, Throughput, Response Time) 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 [14].

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 from resource metrics.

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.

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 second part 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 [22], 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 [17, 19]). Nevertheless, we have designed and implemented a mechanism for deploying SLAs from a Web Services 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, such as CIM.

2. Web Service Level Agreements (WSLA)

This section describes our work towards a flexible SLA monitoring framework, primarily targeted at Web Services, but applicable to any kind of managed resource in a distributed management environment as well. In order to address the requirements of facilitating the SLA definition and the automated configuration of an SLA monitoring infrastructure in inter-domain environments, we have specified and implemented the *Web Service Level Agreement (WSLA)* framework [10]. In [9], 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 of 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 [15] based on the XML schema and a runtime architecture based on several SLA monitoring services, which may be outsourced, either in full or in parts, to third parties to ensure a maximum of accuracy [18]. 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. In addition, the various WSLA monitoring services, described below in further detail, can be configured automatically according to the terms and conditions specified in the SLA. 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 [8].

2.1 SLA Lifecycle in the WSLA 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.

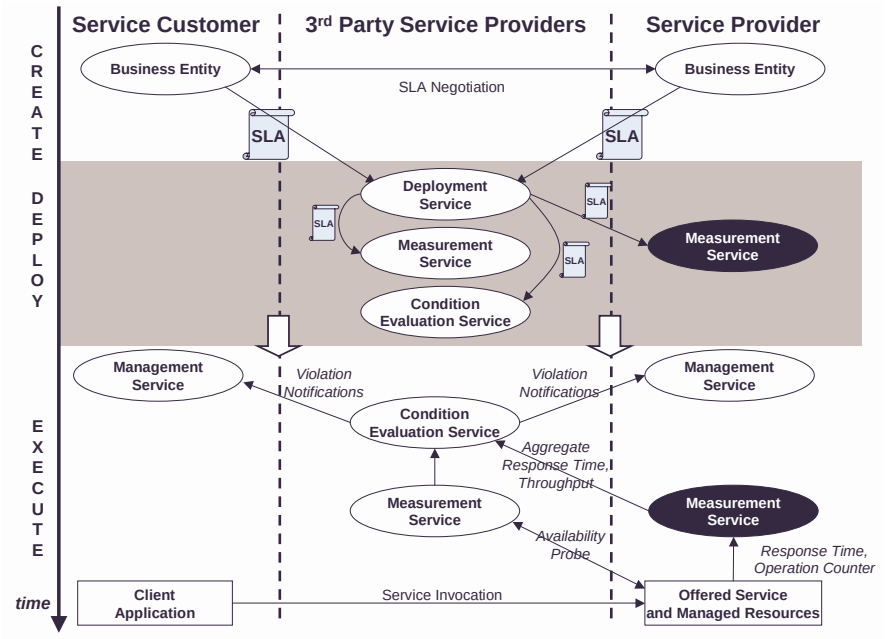

Figure 1. Lifecycle of a Service Level Agreement in a Multi-Provider Environment

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 he 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 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. Keynote Systems, Inc. [11] and Matrix NetSystems, Inc. [16] are real-life examples of such third-party monitoring service providers.

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 (gray shaded area in the middle of Figure 1). The latter is needed to ensure that a supporting service receives only the amount of information it needs to carry out its tasks.

In our scenario, we assume that a part of the SLA monitoring and supervision activities is delegated to third party service providers. Their runtime interactions are depicted in the lower part of Figure 1. 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, mean, median etc.). This is usually being done by a Measurement Service 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). In sections 4 and 5, we will describe our approach to designing and 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 at runtime 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(s) 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 take 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 the WSLA Language

In this section, we provide a brief overview over the parts of the WSLA language that relate to the definition of SLA parameters and the way they are monitored. For a detailed discussion of the entire WSLA language, the reader is referred to [10].

The **Parties** section identifies the contractual parties and contains the technical properties of a party, i.e., their addresses and interface definitions (e.g., the ports for receiving notifications).

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 or from a system acting as a proxy for the resource. Typical examples of measurement directives are the uniform resource identifier of a hosted computer program, a protocol message (e.g., an SNMP GET message), the command for invoking scripts or compiled programs, or a query statement issued against a database or data warehouse. **Functions** are the measurement algorithm that specifies how a composite metric is computed. Examples of functions are formulas of arbitrary length containing mean, median, 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 retrieve and 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. For the sake of brevity, we have omitted their description here; further details and usage examples can be found in [10].

3. Integrating the WSLA and CIM Environments

Considering an SLA management environment as shown in Figure 1 raises the question how the five services can be integrated in the most efficient way. Today, the Business Entity is usally a human being. 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) are best implemented as a Web Service and which services ought to be implemented with CIM? Three alternatives can be considered:

1 The first approach is to implement the services entirely in a Web Services environment, as demonstrated by the WSLA Compliance Monitor of the Web Services Toolkit [8]. This obviously simplifies 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 managed resources. However, if certain tasks are to be delegated to third party providers, this solution makes assumptions about their management infrastructure 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 facilitates the ties between high-level SLA parameters and low-level resource metrics as well as the integration with managed

resources. Making the Condition Evaluation Service a Web Service yields the flexibility of delegating its tasks to third party providers. The Deployment Service is the gateway between both worlds by having a Web Services interface. Its backend acts as a CIM Client and is thus able to communicate with the CIM based Measurement Service for setting up the measurements defined in the SLA.

3.1 Representing SLA Definitions in CIM

Obviously, the straightforward way of implementing an SLA model in CIM is to reuse existing classes of the CIM Core and Common Schemas. However, CIM does not yet provide explicit classes for defining SLAs and SLOs. The CIM Policy Schema consists of various classes representing policies (`CIM_Policy`), which aggregate conditions (`CIM_PolicyCondition`) and actions (`CIM_PolicyAction`). Consequently, these classes could serve as a basis for defining the CIM equivalent of WSLA Obligations. However, obligations are evaluated by the Condition Evaluation Service, which we decided to keep outside the scope of our CIM implementation. Furthermore, the CIM Policy Schema does not provide an explicit representation of conditions and actions, which is the case of WSLA. On the other hand, we were able to reuse the class `CIM_PolicyTimePeriodCondition`, which maps to the WSLA Schedule concept and therefore serves as base class of `IBM_Schedule`. It should be noted that recent work in the Policy Working Group aims at extending the Policy Schema by a mechanism to express SLAs in CIM.

Information relating to the definition of SLA parameters and their associated metrics is found in another CIM Common Schema: The CIM Metrics Schema defines, among other, two classes that allow the representation of metric definitions (`CIM_BaseMetricDefinition`) and – once a metric instance is created – its value (`CIM_BaseMetricValue`). This matches the WSLA philosophy very well, since the metric definitions (and the way they are computed) are part of an SLA and therefore – in a first step – deployed to the Measurement Service in charge of computing them. Once the monitoring process starts, a Measurement Service obtains and computes values for these metric definitions and thus creates new instances of the class `CIM_BaseMetricValue` (or more specific subclasses). We were therefore able to fully take advantage of the CIM Metrics Model. Distinguishing between metric definitions and values 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,

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

Figure 2 depicts the CIM SLA model, which is equivalent to WSLA in terms of its expressiveness. As a result of the decision to implement the Measurement Service in CIM, the model reflects only those parts of an SLA that define the relationships between SLA parameters and metrics, i.e., the aggregation functions and the schedule for their retrieval. The central element of the model is the `IBM_SLA` class, which ties together all the other elements comprising the SLA. It is derived from `CIM_ManagedElement`. Several instances of `IBM_SLA` can exist in parallel, thereby representing SLAs of one individual or several different customers.

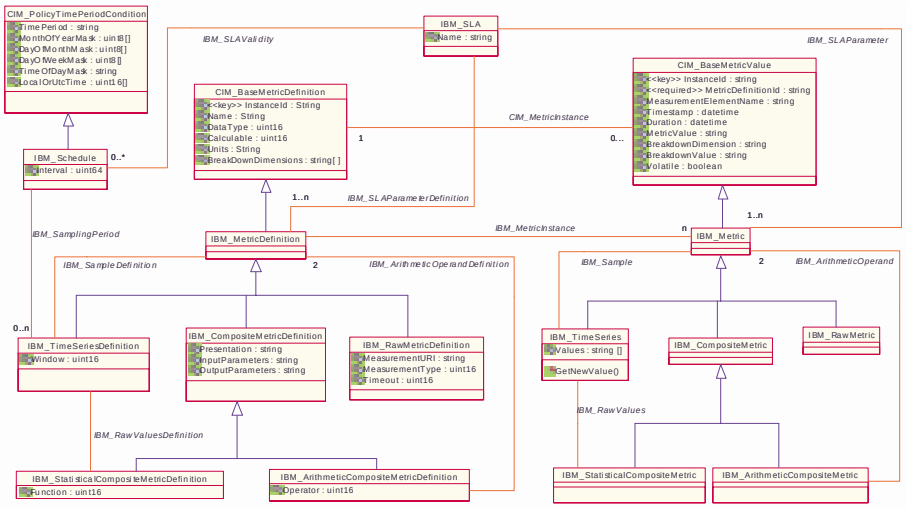

Figure 2. CIM Model of Service Level Agreement definitions and values

`IBM_MetricDefinition` is the base class of the various SLA metric definitions; it inherits from the `CIM_BaseMetricDefinition` class and is refined as follows: As discussed in section 2.1, WSLA distinguishes between raw metrics, composite metrics, and time series. Consequently, the model comprises three corresponding classes `IBM_RawMetricDefinition`, `IBM_CompositeMetricDefinition` and `IBM_TimeSeriesDefinition` that hold the definitions for these metric types. Composite metric definitions are further subclassed because they represent complex metrics that are computed by the Measurement Service. The `IBM_ArithmeticCompositeMetricDefinition` class represents the arithmetic operator (e.g., +,-,*,/) which aggregates two `IBM_MetricDefinitions` by following the association `IBM_ArithmeticOperandDefinition`. The `IBM_StatisticalCompositeMetricDefinition` class captures the definitions of statistical functions that apply to times series. The latter are stored using the `IBM_TimeSeriesDefinition` class, which holds metric values sampled in regular intervals, e.g., according to an `IBM_Schedule`. `IBM_TimeSeries` instances may be used as input for any number of statistical composite metrics. This reduces redundancy and ensures the integrity of measurement data by providing a shared basis for statistical calculations. The intervals during which metrics are collected and placed into a time series (cf. the association `IBM_SamplingPeriod`) are represented by the class `IBM_Schedule`, which extends the class `CIM_PolicyTimePeriodCondition` by a property 'Interval'.

The three metric types, along with the function definitions and their schedules allow the definition of arbitrarily complex SLA parameters, such as the average utilization of network interfaces or the maximum reponse time within the last hour. Note that all the classes discussed until now – depicted in the left part of Figure 2 – are used to represent the definitions within an SLA — and not the actual measurement values. They are instantiated whenever a new SLA is deployed to the CIM based Measurement Service. Once the definition classes are instantiated, the Measurement Service uses these definitions to retrieve and compute the actual values.

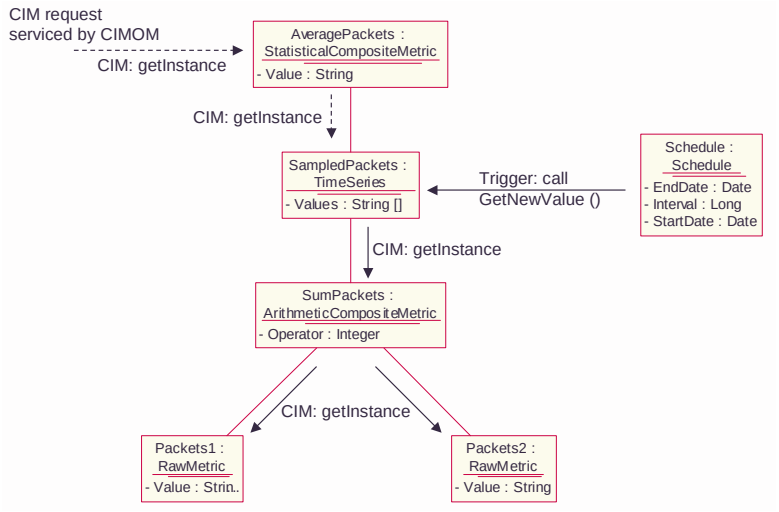

Figure 3. Aggregating the 3 metric types: raw metrics, composite metrics and time series

3.2 Computation and Aggregation of Metric Values

The computation of SLA parameters requires the automatic retrieval of metric values by the Measurement Service. During runtime, instances of `IBM_Schedule` are used to trigger the retrieval of current metric values and to perform metric computation and aggregation. The input metrics and the (intermediate or final) results are represented by the classes we will discuss below; they are depicted in the right half of Figure 2.

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

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

4. CIM based SLA Measurement Service

Figure 4 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 3.1. This model has to be loaded once into the CIMOM (1) and appropriate

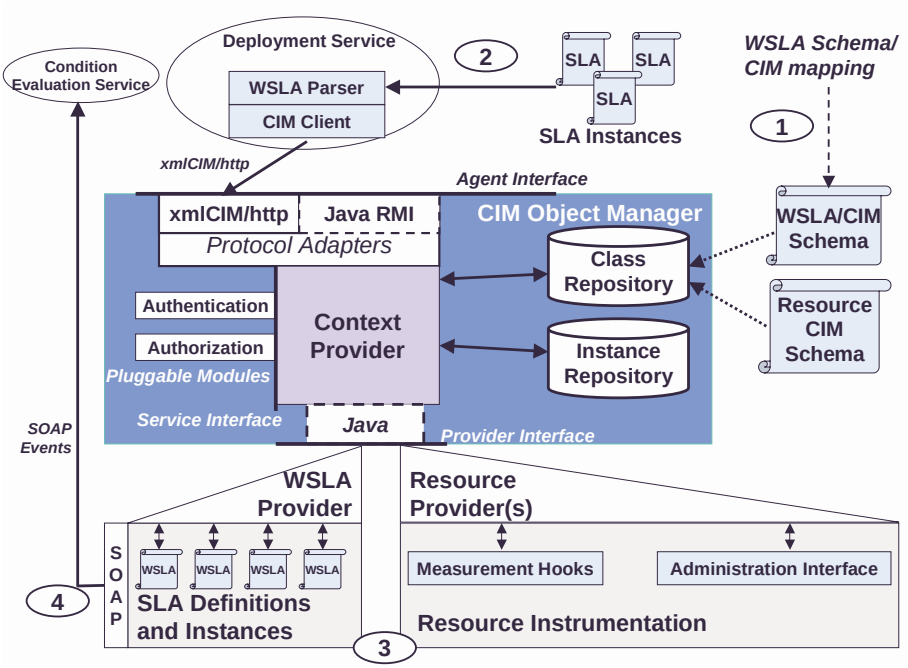

Figure 4. Architecture of the CIM based SLA Measurement Service

providers implementing this information model have to be developed. Since all the SLAs share a common structure, this CIM SLA model is stored in the class repository.

After this WSLA/CIM schema 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). 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 requests to the CIMOM to create the CIM instances and associations representing the content of the SLA. This leads to the instantiation of all the definition classes, which were described in section 3.1.

We distinguish between two types of providers: A WSLA provider, implementing the SLA model (maintaining the SLA definition and carrying out the computations) and one or more resource providers. The resource providers are responsible for exposing metrics from managed resources. In the next step (3), the classes of the WSLA provider that relate to metric values at runtime (described in section 3.2) calculate the SLA parameters based on the raw metrics obtained from one or more resource providers. The computation is either triggered by `IBM_Schedule` instances or CIM requests issued by an external CIM client, respectively. Finally, in step (4), the computed SLA parameters are forwarded as SOAP events to the Condition Evaluation Service (implemented as Web Service) through a SOAP library embedded in the WSLA

provider. This mechanism is detailed in section 4.3. Note that since CIM providers are implemented on a per-class basis, the term *WSLA provider* is used as a shorthand for *the set of CIM instance and association providers implementing the SLA model*.

4.1 Active CIM Providers

One of the major novelties of our approach is the use of active CIM providers. While active management agents are known in the network management community for some time (e.g., the OSI Event Report Management Function or the Summarization Function), CIM providers are, until now, stateless resource providers. They are passive and only surface information from managed resources without providing advanced processing capabilities. Normally, the retrieval of information is initiated by a management system acting as a CIM client. Stateless providers may cause a considerable overhead by requiring periodic polling for new values.

In our implementation, the WSLA provider actively monitors the SLA by autonomously retrieving metric values from managed resources and calculating SLA parameters without being triggered by an external client. Instead, the retrieval of new metrics is automatically requested by the provider implementing the `IBM_Schedule` class of the SLA model (see section 3.2 for details). Having this delegated management functionality executed autonomously eliminates the need for polling and thus reduces the overhead significantly.

4.2 Recovery Mechanism

SLA management requires the continuous monitoring of SLA parameters and metrics. Since the CIM providers are not loaded automatically, but rather on demand, there is the potential problem that providers are not reactivated after a restart of the Measurement Service. Since the monitoring is triggered by the provider implementing the `IBM_Schedule` class, one needs to make sure this provider is properly reactivated.

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 automatically, 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 the CIMOM restart case, providers are not loaded automatically, but only when a request for their data comes in. Consequently, a recovery procedure needs to be in place to guarantee that the `IBM_Schedule` provider is always loaded, since it issues requests to the providers of other classes (e.g., `IBM_TimeSeries`) of the SLA model, which forces the CIMOM to load them if they are not activated. In other words, the `IBM_Schedule` provider is used to bootstrap the other providers of the SLA model. This can be achieved by having a simple CIM client enumerate the `IBM_Schedule` instances once the CIMOM has started. Such a command can be included in the startup script of the CIMOM.

4.3 Event Forwarding

In addition to collecting resource metrics and computing SLA parameters, the Measurement Service needs to forward newly computed SLA parameters by means of asynchronous events to the WSLA Condition Evaluation Service (cf. Figure 4).

CIM defines an event model [3] that enables CIM clients to subscribe to events published as CIM objects by the CIMOM and its providers, respectively. The CIM event model distinguishes between `ClassIndication` (creation/deletion/modification

of CIM classes), `InstIndication` (creation/deletion/modification/read of CIM instances as well as the invocation of methods), and `ProcessIndication` (not CIM-specific alerts caused by managed resources, e.g., SNMP traps). The client can apply filters to limit the number of events being generated. Events are delivered by a handler and are currently limited to the XML-encoded CIM operations. Other handler types are intended for the future, but not specified yet. In this approach, a CIM client would need to act as a gateway to forward CIM events into a non-CIM environment.

Since a generic CIM/Web Services interoperability solution would require the definition of mappings between the different information models and communication protocols (a non-trivial task comparable to generic management gateways, as described in [17]), we chose a more pragmatic approach that is not based on the CIM event model. Instead, we enable our WSLA provider to emit events directly to a WSLA Condition Evaluation Service. This implies bypassing the CIMOM and, thus, the CIM event model. Our task is facilitated by the fact that the provider is already aware of the SLA parameters that are to be sent to each party, because – in WSLA – this event subscription information is already part of the SLA. By doing so, we are able to augment the provider with a SOAP library, which allows the sending of SOAP notifications to the WSLA Condition Evaluation Service. This principle can be applied to notification mechanisms of other architectures (CORBA events, SNMP traps, etc.) as well.

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 (instance or association) provider. This facilitates the adaptability of a model if it is subject to changes and reduces the complexity of individual providers.

However, the principle of implementing every CIM provider on a per-class basis could not be held up for the implementation of the `IBM_ArithmeticCompositeMetric`, `IBM_StatisticalCompositeMetric`, and `IBM_TimeSeries` classes because they have cyclic dependencies (cf. Figure 2). Consider, e.g., the case when the CIMOM is restarted after a failure and when persistent instances have to be reloaded (no persistent storage mechanism of CIM instances is standardized yet, thus every CIM implementation handles this differently) in order to monitor 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 a `IBM_StatisticalCompositeMetric`, the attempt to resolve the reference to this instance would result in a request to the corresponding provider without having finished the initialization of the `IBM_TimeSeries` instances (cf. Figure 5). During the initialization of the other providers, a single reference to an `IBM_TimeSeries` instance would entail an attempt to initialize the `IBM_TimeSeries` provider again, thus resulting in a loop.

Combining the initially three separate providers into a single provider – responsible for handling all three classes – solves this deadlock situation because all metric instances are jointly restored from persistent storage. By doing so, the provider has all information available internally to resolve the references without having to rely on

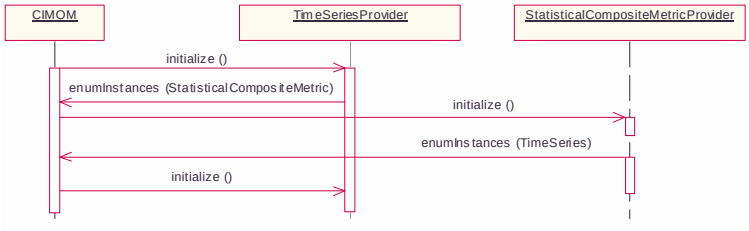

Figure 5. Deadlock between different CIM provider classes

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.

The CIM based Measurement Service and the WSLA Condition Evaluation and Deployment Services are new extensions to an existing prototype for discovering quantitative models for service level management (see [5] in this book for more details). The CIM based Measurement Service, described in this paper, is used to carry out SLA measurement tasks and sends its results through the WSLA Condition Evaluation Service both to the AutoTune Manager and a Managed Probe. The Measurement Service obtains its data from the IBM DB2 UDB database management system, which has been extended by a CIM based management instrumentation.

6. Conclusions and Outlook

We have presented a novel approach for SLA-driven management of distributed systems using CIM. It uses the WSLA framework to define and formally represent SLAs as XML documents. To access existing managed resource instrumentations, these documents need to be transformed into a representation compatible with typical management architectures. Here, we assume a CIM based management environment.

Our approach to this problem consists of developing a CIM based model for SLAs that preserves the expressiveness of WSLA, and an SLA Measurement Service, implemented as a CIM provider. The model 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. It acts as a CIM client and is able to perform the mapping of the SLA into a CIM based environment. During deployment, the Measurement Service instantiates CIM objects that represent the SLA definitions. During the runtime phase, new metric data is collected from managed resources and subsequently aggregated into SLA parameters and forwarded to one or more Condition Evaluation Services, according to the schedule defined in the SLA. This implies that the data collection must be triggered from within the CIM Object Manager by means of so-called active CIM providers. This concept, available e.g., in the Summarization Function of the OSI/TMN management framework, goes beyond the capabilities of today's 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 some work, since typical CIMOM implementations are not geared towards active providers. In particular, we had to address the problem of ensuring a 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.

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] CIM Event Model White Paper. Version 2.6, Distributed Management Task Force, March 2002.

[4] 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.

[5] Y. Diao, F. Eskesen, S. Froehlich, J.L. Hellerstein, A. Keller, L.F. Spainhower, and M. Surendra. Generic On-Line Discovery of Quantitative Models for Service Level Management. In G.S. Goldszmidt and J. Schönwälder, editors, *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management*. Kluwer Academic Publishers, March 2003.

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

[7] 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.

[8] IBM Corporation. *Web Services Toolkit version 3.2*, August 2002. Available at: http://www.alphaworks.ibm.com/tech/webservicestoolkit.

[9] 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.

[10] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management, Special Issue on E-Business Management*, 11(1), March 2003.

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

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

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

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

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

[16] Matrix NetSystems, Inc. *Beyond Mere Latency*, 2002. http://www.matrixnetsystems.com.

[17] 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.

[18] C. Overton. On the Theory and Practice of Internet SLAs. *Journal of Computer Resource Measurement*, 106:32–45, April 2002. Computer Measurement Group.

[19] 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.

[20] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid Service Specification. Draft 3, Global Grid Forum, July 2002.

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

[22] 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.