Unifying Service Level Management using an MDA-based Approach

Markus Debusmann1, Kurt Geihs2, Reinhold Kröger3

1 Fachhochschule Wiesbaden - University of Applied Sciences
Department of Computer Science, Distributed Systems Laboratory
Kurt-Schumacher-Ring 18, D-65197 Wiesbaden, Germany
{debusmann|kroeger}@informatik.fh-wiesbaden.de

2 Berlin University of Technology
Intelligent Networks and Management of Distributed Systems
Einsteinufer 17, 10587 Berlin, Germany
geihs@ixs.tu-berlin.de

Abstract

Service Level Management (SLM) has gained more and more importance in the past years. One of the biggest challenges is the integration of emerging new technologies such as Web Services. A large number of different SLM approaches has been proposed which complicates an efficient SLM. This paper describes an SLM approach that is based on the Model Driven Architecture (MDA). The first step is to specify abstract reusable SLA patterns which are bound later to a concrete management platform and are finally physically deployed into the managed environment. This approach benefits from the advantages of the MDA and provides a mechanism for transforming SLA patterns for different SLA management platforms. Our approach has been prototypically implemented for the Web Services Level Agreement (WSLA) environment.

Keywords
Service Level Management, Service Level Agreement, Model Driven Architecture, Model Transformation

1 Introduction

The economic pressure on enterprises increases the need for outsourcing and purchasing services from external service providers. A Service Level Agreement (SLA) [Lew01, SMJ00] represents an agreed upon contract between service provider and service customer. SLAs are an important element within the outsourcing business and define Quality of Service (QoS) parameters of the provided service. Typically, an SLA defines SLA parameters that are related to performance and availability,
e.g., a threshold or average for the response time of important functions of the service. During runtime these high-level SLA parameters have to be monitored in order to detect violations.

Over the past years, a large number of different middleware technologies have evolved, such as CORBA, J2EE, and Web Services. Middleware serves as a kind of glue between the components of distributed applications. Especially Web Services have gained increasing attention by science and industry. They are regarded as a key technology for the integration of enterprise applications, for building B2B applications, as well as for Grid Computing [FK99], which addresses highly distributed scientific and commercial applications. This central role of Web Services for many applications requires an efficient management of Web Services especially in the area of Service Level Management (SLM).

Service Level Management has been one of the hottest research topics of the past years which has lead to a large number of SLM approaches. Nevertheless, many problems are still unsolved. [GHH+01, GHH+02] present the MNM Service Model which provides a high level view upon an SLA scenario. The model is mainly used for analysis and specification purposes; for practical SLM the model is too abstract, since it does not define the binding to managed resources. The Web Services Level Agreement (WSLA) approach by IBM [KKL+02] provides an XML-based language for specifying SLAs as well as a set of Web Services (deployment, measurement, condition evaluation) for deploying and monitoring SLAs. In [DK03] the concept of a CIM-based measurement service is presented. The Web Services Management Network (WSMN) [MSvM03] by HP is an alternative approach to Service Level Management of Web Services-based environments. Like WSLA, WSMN also defines a XML-based language for specifying SLAs. WSMN places necessary management functionality (e.g., monitoring and correlation of SLAs) into the Web Services platform itself. In [RL01, Rod03] the notion of a service template is introduced which describes a service from a customer’s point of view as well as from a provider’s point of view. A service template is defined using UML but its mapping to a concrete environment has not been specified so far. Thus, in large enterprises and B2B scenarios the variety of competing SLM approaches requires a unifying approach that provides an abstraction of existing SLM approaches.

In the area of software development, the Model Driven Architecture (MDA), specified by the Object Management Group (OMG), defines a unifying approach. The MDA is a response to the fact that today’s application developers face a continuous change in middleware platforms and computing infrastructures. The core paradigm of MDA is model transformation. With MDA, system construction consists of a sequence of models and transformations between these models. The model-driven approach starts with a platform-independent model (PIM), which is subsequently transformed into a platform-specific model (PSM). The PSM is then transformed to code. Generally, PIM and PSM are defined using OMG’s Unified Modelling Language (UML).

We claim that Service Level Management can benefit tremendously from the MDA paradigm. In analogy to the MDA, definition of abstract SLA patterns, that
can then be transformed and bound to concrete management platforms, is proposed. The MDA-based management approach provides the necessary level of abstraction needed for dynamic on-demand services in large-scale heterogeneous environments. Furthermore, the proposed abstract SLA templates provide added value by being reusable and long-lasting.

The remainder of this paper describes the proposed MDA-based approach for Service Level Management. The paper is structured as follows: Section 2 briefly introduces the Model Driven Architecture. In section 3 the approach as well as the architecture for its realisation are described. In section 4 the UML profile for defining SLA templates including an example demonstrating the application of the profile are presented. Section 5 briefly discusses implementation details. Section 6 concludes the paper and gives an outlook to future work.

2 Model Driven Architecture & UML

The design and implementation of modern software systems is complex and costly due to rapid and continuous technology changes. This requires tremendous efforts to integrate applications, as well as support for developing adaptive and open applications that can keep track with the steady technological progress. The Model Driven Architecture (MDA), defined by the Object Management Group (OMG), aims at solving the integration and development problems. It defines a model-based methodology which separates the application problem domain from the technological specifics of the underlying computing infrastructure [OMG01].

The MDA distinguishes between three models: the Platform-Independent Model (PIM), the Platform-Specific Model (PSM), and the Platform-Specific Code (PSC). The models can be converted into each other by a process called model transformation. Typically, an MDA tool (or a chain of tools) supports the definition and transformation of these models.

The PIM is the core ingredient of the MDA. It specifies a system independent of any platform-specific technology that is used later to implement the system. The PIM focuses on the business functionality and behaviour including the relationships between the components involved. MDA's vision is the specification of computationally complete PIMs. Ideally, all development is done at this modelling level, and these models can be exploited for testing and verification purposes.

The PSM is obtained by transforming the PIM for a given target platform, e.g., Web Services, J2EE, or CORBA. The goal is to capture as much of the platform know-how as possible in the PIM-PSM transformation and to generate as much of the PSM as possible automatically. Thus, the developer can concentrate on the application logic in the PIM and is relieved from the complexities of platform-specific code.

The last step is the generation of the (platform-specific) code from the PSM. Ideally, a MDA tool generates all necessary application code (and associated files) for the chosen programming language and deployment platform.
MDA does not prescribe a strict waterfall model, but explicitly addresses concerns for reengineering and round-trip engineering of application software. Common MDA tools differ substantially when it comes to support these extra features for incremental software development.

For the specification of PIM and PSM, the MDA proposes the Unified Modeling Language (UML) which is also an OMG standard. UML supports various meta-level mechanisms to extend its specification elements, e.g., stereotype extensions and UML profiles. Often, elements in the PIM are specified as components as defined in the EDOC UML profile [OMG02] and the new UML 2 standard [OMG03].

However, this does not necessarily imply a system realisation based on component technology. For the exchange of UML models, e.g. between UML tools, the OMG has defined an XML-based exchange format called XML Metadata Interchange (XMI) [OMG04].

3 MDA-based Approach for Unifying SLM

This section addresses our approach for unifying Service Level Management using the Model-Driven Architecture. First, the general process is presented. After this, the architectural components for realising the approach are discussed.

3.1 Applying MDA to Service Level Management

When applying the MDA to Service Level Management, we need to define the meaning of the different MDA models (PIM, PSM, PSC; see section 2) for the area of SLM first. Figure 1 illustrates the analogy. The PIM corresponds to an abstract SLA which only defines service types and their relations to various other component types, i.e., no concrete instances of components are identified here. The abstract SLA does not identify a management infrastructure such as CIM or SNMP. Furthermore, the Service Level Objectives (SLOs) and their derivation are defined in an abstract manner, i.e., their calculation is defined based on the attributes of the component types. Such an abstract SLA can be reused and applied for various configurations and is therefore referred to as SLA pattern.

In the next step, an SLA pattern ("the PIM") is transformed into a concrete SLA ("the PSM"), which we call SLA instance, by binding the pattern to concrete configuration information. This includes the binding of the abstract component types

![Diagram of MDA-based Approach for Unifying SLM](image)

**Figure 1:** Applying MDA to Service Level Management
to concrete component instances as well as the identification of a concrete management framework, e.g. CIM, that is used for managing the SLA. An SLA instance contains all information necessary for configuring the management infrastructure for the given SLA.

The third step is the actual deployment of the SLA instance (“the PSC”). The result of this step is the actual configuration of the management infrastructure enabling it to autonomously manage the conformance of the SLA.

3.2 Architecture for Realising the MDA-based Approach

This section describes the different models and the corresponding architecture in more detail. In all figures information items are displayed as dark gray rectangles with rounded edges, while components processing these information items are displayed as light gray rectangles.

3.2.1 Defining an SLA pattern

The starting point for Service Level Management is the signing of an SLA contract between a service customer and a service provider. This signed SLA is a legal contract. In our approach, the formalisation of the contract is done in UML by an administrator using an UML tool and an appropriate UML profile (see section 4) which is tailored for SLA definition. The result of the formalisation is the SLA pattern which is stored in the SLA repository for further use. An SLA pattern specifies the relations and dependencies of service types in an abstract manner, i.e., the SLA pattern identifies the types rather than concrete instances of dependent services and components. For example, a service for providing HTML content might depend on a web server, a web container, a database, some machines hosting the components, and some network connections between them. Thus, an SLA pattern is very similar to a pattern that identifies interrelations between services and components. The beauty of the approach is that for a certain type of service a pattern has to be defined only once and then can be re-used and instantiated multiple times.

![Figure 2: Definition of the Platform Independent Model](image_url)

Although UML tools offer the opportunity to export the UML model in the XML Metadata Interchange (XMI) [OMG04] format, which can be processed by other tools, we cannot recommend using XMI here. From our experience, its processing
3.2.2 Generating an SLA instance

For actually managing an SLA, a corresponding SLA pattern is retrieved from the SLA repository and bound to a concrete configuration. This is a rather complex task, because all real instances of the types identified in the SLA pattern have to be known. The necessary information is usually retrieved from a configuration repository which contains all relevant information concerning the dependencies between services and components. The transformation of the SLA pattern into an SLA instance is performed by the binding service (see figure 3). As explained above, such an SLA instance is constructed for a certain management infrastructure, e.g., CIM.

Figure 3: Generation of the Platform Specific Model

The SLA repository stores the UML models in the original format as exported by the UML tool. This is necessary in order to load an SLA pattern from the repository back into the UML tool for further modifications of the pattern.

For the subsequent processing phases with other tools it is not necessary to use the complete UML model of the SLA pattern. This would be rather cumbersome because such models can become rather complex. In addition, a lot of information is included which is not needed for generating the SLA instance. Thus, the first task of the binding service after having retrieved an SLA pattern is its transformation into an intermediate format which only contains the information necessary for creating the SLA instance. This processing step is not depicted in figure 3; it is assumed that the processing is done inside the binding service. Clearly, the format of the SLA instance (e.g., a XML document, source code, etc.) highly depends on the management infrastructure used to manage the SLA.

3.2.3 Deploying an SLA instance

The final step in the MDA is the generation of the platform-specific code. For Service Level Management this corresponds to the physical deployment of the SLA instance generated in the previous phase.

The SLA instance is passed to an SLA deployment service (see figure 4) which is specific for the target management infrastructure. The SLA deployment service
configures the infrastructure specific services that are responsible for managing the SLA conformance. For example, when using WSLA as management infrastructure, the WSLA deployment service will configure WSLA’s measurement service and condition evaluation service. After their configuration, these services are set up to autonomously manage the deployed SLA.

![Figure 4: Generation of the Platform Specific Code](image)

4 Information Model for SLAs

This section describes the information model for defining SLAs. First, the required UML extension mechanisms are introduced, and then the information model is presented. Finally, an example illustrates our approach.

4.1 UML Extension mechanisms

UML is a general purpose modelling language. Not surprisingly, it does not provide special support for specifying SLAs. However, for unambiguously specifying SLAs, such specific modelling elements are needed.

The UML standard offers basically three extension mechanisms: stereotypes, tagged values and profiles [Fra03]. A **stereotype** can extend any UML metamodel element. A stereotyped element has a special meaning in a certain application domain, e.g., the UML **Classifier** element can be stereotyped with **SLA** to express that it represents a Service Level Agreement. Stereotypes can be used as input for code generators and therefore play a very important role in the MDA since code generators can generate specific code for each (stereotyped) class. **Tagged values** are used for extending model elements by adding tags which are comparable to attributes. The values of the tags are defined in the instances of the extended model elements. Stereotypes and tagged values can also be combined, i.e., stereotyped elements can also be extended by tagged values. For example, a class can be stereotyped to be a metric that computes a result from two values. A tagged value can then be used to specify the arithmetical operator for the computation. The third kind of extension
mechanism is the UML profile, also called virtual metamodel. A UML profile is a collection of stereotypes, tagged values, and extended metamodel elements. We have created a new UML profile for SLA specification using stereotypes and tagged values.

4.2 UML SLA Profile

As described in section 3, an appropriate information model for specifying SLA patterns (the PIM) is needed. The UML profiling mechanism was chosen for extending and customising UML for SLM. The following section discusses the ingredients of this UML profile for SLA specifications.

The analysis of different approaches to specifying SLAs ([LKD+03], [MSvM03], [NO97], [SMJ00]) revealed that these approaches more or less use the same abstract specification elements and patterns. Therefore, we chose the WSLA [LKD+03] approach as a starting point for the definition of our profile. In the following figures, all classes of the UML metamodel are shaded grey; the extension classes of our profile are drawn in white.

4.3 Basic Service Relations

Figure 5 depicts the metamodel for defining basic service relations. The core element is the SLA which is agreed upon different Actors. Possible Actors are the service customer, the service provider, and third party providers (called supporting parties in WSLA) which may additionally monitor the SLA conformance. Within an SLA, a Service is specified which has to be managed. From the UML perspective, a Service can be modelled as a component\(^1\) which offers a number of ports. These ports represent ServiceFunctions which offer the interfaces provided by the component. The ServiceFunctions are related to SLAParameters which

---

\(^1\) This does not imply any assumptions about the physical implementation of the service

Figure 5: Metamodel for describing the basic service relations
define characteristics of the service such as response time, availability, etc. The Schedule associated with an SLA parameter specifies points in time when a SLA parameter is calculated.

### 4.4 Calculation of SLA parameters

Figure 6 depicts the metamodel for defining the calculation of SLA parameters as well as associated metrics. An SLAParameter is associated with a Metric element which represents the value of the SLAParameter. The Metric itself is abstract and serves as base class for specialised metrics classes. The RawMetric class represents a simple metric that is retrieved from a managed resource. The retrieval is specified as a MeasurementDirective. How a RawMetric is retrieved via a MeasurementDirective highly depends on the kind of managed resource. The MeasurementDirective class depicted in figure 6 is abstract and serves as a base class for specialised classes for retrieving metrics using CIM, SNMP, Web Services, etc. These specialised classes are not depicted in figure 6.

The ArithmeticMetric is used for performing a simple arithmetic computation (+, −, ×, ÷) based on the values of two metrics. A StatisticalMetric performs statistical calculations such as mean, min/max values, average, and standard deviation. The calculation is based on the values stored in an associated Time-Series element which is responsible for storing the last n values of a metric.

![Figure 6: Metamodel for calculating SLA parameters and metrics](image)

### 4.5 Validation of SLOs

Figure 7 depicts the metamodel for defining the validation of Service Level Objectives (SLOs). Every SLAParameter is associated with at least one Obligation which is an abstract base class and is further specialised by SLOs and Action Guarantees. An SLO represents a Service Level Objective which was agreed upon by customer and provider. The provider guarantees to keep the SLO within
certain limits, e.g., the response time of a service function will be kept below 5 sec. An SLO is associated with at least one Period which defines the time interval in which the obligation has to be evaluated. An ActionGuarantee expresses a commitment to perform a certain action if a given precondition is met. In case the SLO is violated, the associated QualifiedActions are performed (for details see next section), e.g., a notification of the customer, the restart of a server process, the adaptation of network priorities, etc. An obligation is associated with an Expression which is refined by the Predicate and LogicExpression classes. A predicate compares two metric classes (see previous section), while a logic expression logically compares two expressions. An obligation is also associated with an EvaluationTrigger (see next section for details) which specifies when the obligation has to be evaluated.

Figure 7: Metamodel for describing the validation of SLOs

4.6 Triggers and Actions

Figure 8 depicts the triggers initiating the evaluation of an obligation and the action performed as a result of a SLO violation. The EvaluationTrigger class is refined by the ScheduleEvaluation and the EventEvaluation class. The ScheduleEvaluation class represents the evaluation of an obligation based on a regular basis which is defined by the associated Schedule class. The EventEvaluation class represents an obligation triggered by the associated Event class. Currently, the only event defined so far is NewValue which means an evaluation is triggered whenever a new value of a SLA parameter is available.

The QualifiedAction is the abstract base class for actions carried out by a party participating in the SLA. The party that is obliged to carry out the action is specified by the associated Actor class. Currently, only a Notification is
defined as a valid action. A notification is associated with the SLA parameter that was violated and the action guarantee which caused the notification. The execution of an action can be restricted by the `ExecutionModality` which defines whether the action is always carried out when the condition yields true or only the first time, etc.

![Figure 8: Metamodel for describing Triggers and Actions](image)

### 4.7 Sample Application of the Profile

Figure 9 depicts a sample SLA which is defined using the UML profile described in the previous sections. This example intends to provide a better understanding.

![Figure 9: Sample SLA defined using the SLA profile](image)
of the SLA profile; it does not represent a real world example (cp. section 5). For the sake of brevity, the figure only shows the calculation of a SLA parameter and its related metrics; the definition of the obligations is not shown. The SLA defines the SampleService service that offers the GetPrice operation for which an SLA parameter is defined. The SLA parameter defines the average response time of the GetPrice operation. The value of the SLA parameter is provided by ArithmeticMetric named AvgRespTime which is computed as the sum of the average response times for two hosts (AvgRTHost1 and AvgRTHost2) divided by two. The average response times AvgRTHost1 and AvgRTHost2 are StatisticalMetric that compute the average based on the values stored in two TimeSeries (RTsHost1 and RTsHost2). These TimeSeries are filled regularly with the actual values of the RawMetric RTHost1 and RTHost2. The retrieval of the raw metrics is scheduled by the SampleSchedule class.

5 Prototype Implementation

The prototype implementation is ongoing work. Some important components demonstrating the feasibility of the proposed approach already exist, other components especially for the runtime environment are still underway.

The SLA UML profile has been defined using a UML tool called Kase [Wei03] which offers powerful support for model transformations according to the MDA. The SLA patterns that are defined with Kase are transformed using Kase’s Python library (www.python.org). Currently, we are working on transforming SLA patterns (the PIM) into a document conforming to the WSLA specification [LKD+03] (the PSM). The generated SLA instance can then be deployed into the WSLA environment which includes setting up the necessary WSLA services that monitor the SLA (the PSC). This will result in a first fully functionally prototype. At present, the necessary configuration information is stored in an XML file. This will be replaced by an LDAP repository in the future. Currently, the SLA patterns are simply read from the file system. In the future, the SLA repository will be implemented as a Web Service. The same holds true for the binding service which is currently invoked at the shell command level. For the future, a graphical engineering environment for the deployment of SLAs is envisaged.

The prototype is tested and evaluated in our e-business infrastructure. It consist of a Web proxy on the client side and a Web server, a Web container, EJB application servers, a CORBA environment and a relational database on the server side. Our environment is fully ARM-instrumented [TOG98] and thus may serve for collecting performance measurement data needed for validating SLOs (see [DSSK03a, DSSK03b] for details). We are working on developing a collection of reusable SLA patterns for such an infrastructure.

In order to validate the genericity of our approach, additional transformations for other management systems such as our management core presented in [DK01] will be implemented.
6 Conclusions

Service Level Management is a must in service-oriented architectures. Today, developers of management solutions in large scale application environments face the challenge of heterogeneous management architectures and platforms. The enormous complexity of such environments demands a systematic approach to unify the management of SLAs. This paper has presented a comprehensive development approach that is based on the principles of OMG’s Model Driven Architecture. The focus of this paper was to underline the importance of a platform-independent approach and to show how such an approach can be engineered using appropriate UML extensions and MDA tools. The model-based SLM approach takes much of the burden of the platform complexity away from the service user and the service provider and lets them concentrate on their business objectives. The proposed SLA models and transformations facilitate substantially the process of SLA negotiation and configuration. Furthermore, the re-use and maintenance of SLAs is supported.

The new approach is currently being implemented. For the specification of the SLA patterns the UML tool called KASE is used, which eases the MDA-approach significantly. The necessary transformations are implemented using Python scripts, and WSLA was chosen as the first SLA management platform. Our implementation experiences so far, using these tools and platform have increased significantly our confidence that the MDA-based approach to SLM can help to master the inherent complexity of today’s SLM environments.

The future work will focus on gaining experience on the efficiency of the MDA-based approach by applying it to our e-business infrastructure. In addition, the UML profile will incrementally be made more generic in order to also support other SLM approaches such as WSMN. Furthermore, our work will concentrate on making the specification process more modular, i.e., SLA patterns will be composable by using existing patterns as building blocks in order to define large-scale SLAs.

References


