Tailoring a Model-Driven Quality-of-Service DSL for Various Stakeholders

Ernst Oberortner, Uwe Zdun, and Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstr. 8/184-1, 1040 Vienna, Austria
{e.oberortner,zdun,dustdar}@infosys.tuwien.ac.at

Abstract

Many business systems have to comply to various contracts and agreements. Multiple technical and non-technical stakeholders with different background and knowledge are involved in modeling such business concerns. In many cases, these concerns are only encoded in the technical models and implementations of the systems, making it hard for non-technical stakeholders to get involved in the modeling process. In this paper we propose to tackle this problem by providing model-driven Domain-specific Languages (DSL) for specifying the contracts and agreements, as well as an approach to separate these DSLs into sub-languages at different abstraction levels, where each sub-language is tailored for the appropriate stakeholders. We exemplify our approach by describing a Quality-of-Service (QoS) DSL which can be used to describe Service Level Agreements (SLA). This work provides insights into how DSLs can be utilized to model and enrich business systems with concerns defined in contracts and agreements.

1. Introduction

A major requirement for many contemporary business systems is to comply to contracts and agreements, such as Service Level Agreements (SLA). SLAs are contracts between service providers and service consumers. SLAs assure that customers get the service they paid for and that the service fulfils the SLA's requirements, such as availability, accessibility, or performance. For a provider it could result in serious financial consequences if the SLAs are not fulfilled. Hence, service providers need to know what they can promise within SLAs and what their IT infrastructure can deliver. To validate SLAs, mainly Quality-of-Service (QoS) is utilized. In this respect, QoS measurements collect information about services [5].

Modeling business systems with different frameworks or notations gains momentum. In this context, multiple stakeholders – technical and non-technical – with different background and knowledge are involved in the modeling process [8]. But, to the best of our knowledge, no framework or notation exists which provides the facilities for modeling business systems with contracts and agreements they have to comply to and for involving multiple technical and non-technical stakeholders.

One possible way of modeling business systems is to use Domain-specific Languages (DSL). DSLs are small languages that are tailored to be particularly expressive in a certain problem domain. A DSL describes the domain knowledge via a graphical or textual syntax which is tied to domain-specific modeling elements. Nowadays, DSLs are often developed by following the Model-driven Software Development (MDSD) paradigm to describe the graphical or textual DSL syntax through a precisely specified language model. Using MDSD-based DSLs for modeling business systems enables technical and non-technical experts to work at higher levels of abstraction [10].

In this paper we introduce our approach for specifying the contracts and agreements, as well as an approach to separate MDSD-based DSLs into sub-languages at different abstraction levels, where each sub-language is tailored for the appropriate stakeholders. To demonstrate our approach, an example is provided, where an MDSD-based DSL for specifying QoS measurements – which are used for validating SLAs – of services is illustrated. The DSL is separated into two different languages. One language is tailored for non-technical experts, and the other one for technical experts. The DSL for non-technical experts, from now on called domain experts, should provide constructs to specify which QoS values have to be measured on which services, such as response time or wrapping time [3]. Furthermore, domain experts should be able to specify the SLAs and actions which should be performed if an SLA becomes violated. On the other hand, the DSL for technical experts should provide constructs for specifying how the different QoS values have to be measured and how the actions are performed on a particular platform or technology.

This paper is organized as follows: The following Section 2 illustrates the structure of MDSD-based DSLs. Section 3 describes our approach. An example of following our approach, using MDSD-based DSLs to specify QoS concerns for service-oriented business systems, is shown in Section 4. Next, Section 5 lists some benefits and drawbacks of our approach which we have collected during this work. Some related works are compared in Section 6. Finally, Section 7 summarizes the paper and characterizes future work.

2. Domain-specific Languages (DSL) based on Model-driven Software Development (MDSD)

A very common development approach for DSLs is Model-driven Software Development (MDSD) because MDSD provides high levels of abstraction and platform-independence. Figure 1 depicts the infrastructure of MDSD-based DSLs, which is also proposed in [15, 2].

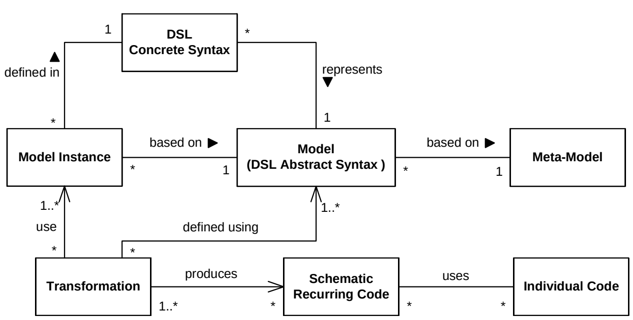

Figure 1. DSLs based on MDSD – relevant concepts

A DSL consists of an abstract and concrete syntax. The abstract syntax, which represents the language model, defines the elements of the domain and their relationships without considering their notations. The meta-model defines how the domain elements and their relations can be described [15]. The concrete syntax describes the representation of the domain elements and their relationships in a form suitable for the stakeholders using the DSL. Abstract and concrete syntax enable DSL users to define model instances with a familiar notation to represent particular problems of the domain. The ultimate goal of the transformations, which are defined on the language model, is to transform the model instances into executable languages, such as programming languages or process execution languages. The MDSD tools are used to generate all those parts of the (executable) code which are schematic and recurring, and hence can be automated.

DSLs based on MDSD, from now on just called DSLs, can provide multiple levels of abstractions to help multiple stakeholders, with different backgrounds and knowledge, to express relations and behaviors of a domain with familiar notations. The goal is that each stakeholder – maybe with the help of other stakeholders – can easily understand, validate, and even develop parts of the needed solution. For instance, domain experts do not have to deal with technological aspects, such as programming APIs or service interface descriptions. Technical experts do not have to deal with unfamiliar domain terminologies [14].

The goal of DSLs is to be more expressive, to tackle complexity better, and to make modeling easier and more convenient [7]. This leads to an intense collaboration between the different stakeholders and lowers the possibility of misunderstandings [13]. However, successful development of a DSL requires the involvement of domain and technical experts, including the design of the notation and evaluation of the clarity of the language.

3. Our DSL Approach

To offer expressive and convenient languages for the different stakeholders, our approach provides a separation of DSLs into multiple sub-languages, where each sub-language is tailored for the appropriate stakeholders. Our approach of separating DSLs into two sub-languages at different abstraction levels, where each sub-language is tailored for the appropriate stakeholders, is illustrated in Figure 2.

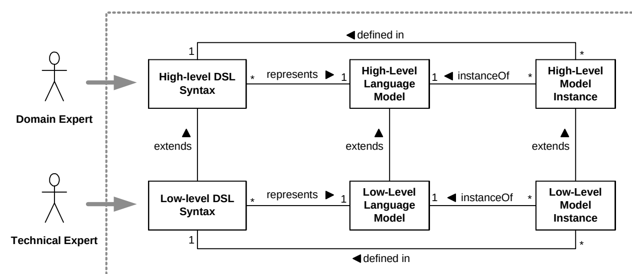

Figure 2. DSLs based on MDSD - separation into high- and low-level languages

On the one hand, domain experts can work in a language, from now on called high-level language, where the terminologies and notations are close or equal to the domain terminology. On the other hand, technical experts can express the additional technical aspects in a language, from now on called low-level language, where the terminologies and notations are close or equal to the terminology of the used technology. The syntax of the high- and low-level languages is based on language models. Low-level language models are extensions of the high-level language models or vice versa. In this way, technical experts are able to add the additionally needed technical aspects. DSLs are then used to define model instances of the high- and low-level language models. Each model instance represents concrete solutions of a particular problems of the domain.

Following our approach does not mean that only a separation into two levels, such as high- and low-level, is possible. It is possible to provide multiple different levels of abstractions where each level of abstraction is tailored for the designated stakeholders. The number of different levels of abstractions depends on the problem domain, as well as on the number of the different stakeholders.

The following section provides an example of following our approach for separating and tailoring model-driven DSLs. As illustrated in Figure 2, a separation into two levels – high- and low-level – is provided, and the levels of abstraction are tailored for domain and technical experts, respectively.

4. An Example: The QoS DSL

The purpose of the following DSL is to enable the DSL users to model service-oriented business systems for measuring Quality-of-Service (QoS) of Web services, Service-Level-Agreements (SLA) based upon QoS measurements, and actions which should be performed if SLAs are violated. We provide two DSLs: The first one, the high-level language, is tailored for domain experts, whereas the second one, the low-level language, is tailored for technical experts and extends the high-level language. Only the merging of the two DSLs results in a complete language model from which a running system can be generated.

Domain experts should be able to specify which QoS values have to be measured for a specific Web service to fulfil the contractually agreed SLAs, as well as actions which should be performed if a certain SLA gets violated. The high-level DSL should provide constructs and expressions that are named similar to the terminology of the QoS and SLA domain the DSL was designed for. An example of specifying the given requirements is: If the response time is longer than 2 minutes, then send an e-mail to the administrator of the service provider.

Technical experts need a language for specifying how the different QoS values are measured in a particular technology, as well as how the defined actions are executed or performed. In this example, the low-level language is an extension of the high-level one, because it enriches the high-level language model with technical concerns, e.g., how to measure the response time in a particular Web service engine. Similar to high-level DSLs, the constructs and expressions of the low-level DSL are named similar or equivalent to the appropriate technology.

In the following we will describe the language models of the high- and low-level DSLs, how the high-level models get extended by the low-level ones, and how both DSLs can be used by domain and technical experts.

4.1. The QoS DSL Models

4.1.1 The High-Level QoS Model

The requirements for the high-level QoS DSL can be formulated as follows: SLAs are associated with QoS measurements of Web services, as well as with actions. In this example, the main attention lies on performance QoS measurements, such as response time and wrapping time [4].

Figure 3 depicts the language model of the high-level QoS DSL. Services are associated with QoSMeasurements. For the time being, we provide classes for measuring `Performance` and `Dependability` QoS measurements, as described in [4]. Each `QoS Measurement` can have `Service Level Agreements` which are in relation with different `Actions` that should be performed if an SLA gets violated. For instance, if the response time of a service is longer than 2 minutes, a mail should be sent to the service provider.

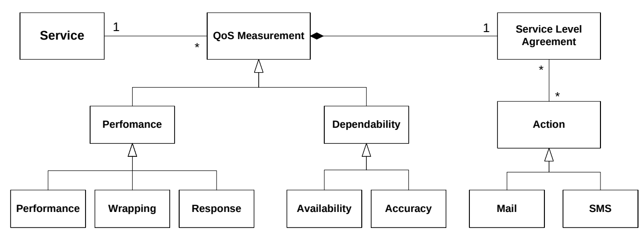

Figure 3. The model of the high-level QoS language

The high-level language model is extended by the following low-level model which contains all necessary constructs for specifying the technological aspects to generate a running system from the model instances described with the DSLs.

4.1.2 The Low-Level QoS Model

The expressions of the low-level QoS DSL depend on the technology on which the DSL is based. We decided to use the open-source Apache CXF Web service framework [16] in our prototype. The requirements can be modelled as follows: The communication between service client and service provider is based on message-flows. Each message-flow consists of a number of phases, where each phase can contain handlers for measuring QoS values. For instance, the handler for measuring the response time is associated to two certain phases of the message flow on the client side.

Figure 4 depicts the low-level language model of the QoS DSL and how the low-level model extends the high-level model. The `Service` class of the low-level model extends the `Service` class of the high-level language model through inheritance. Services are enriched with `Operations` which have a particular number of `Parameters`. For measuring QoS values of services, such as the response time, `QoSHandlers` are associated to services. Again, QoS handlers extend QoS measurements of the high-level language model. In our case, the QoS handler class extends the `Response` class of the high-level language model. In the Apache CXF framework, QoS handlers are associated to `Phases` where each phase corresponds to a certain `MessageFlow`. Using these classes, the technical experts can specify in which phases of which message-flows the QoS values of a service have to be measured.

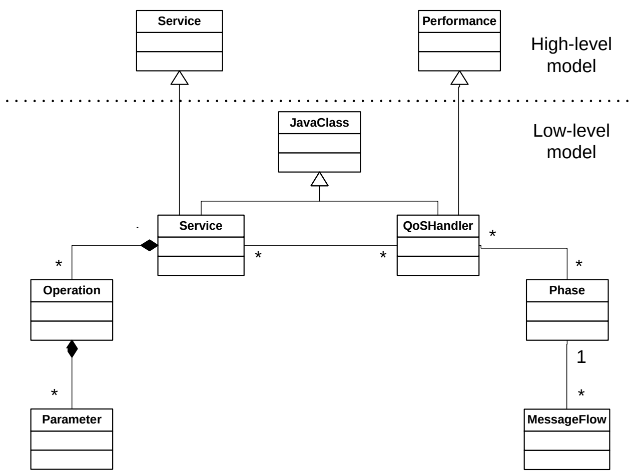

Figure 4. The model of the low-level QoS language

The relationship between the names of the constructs of the DSL syntaxes and the name of the classes defined in the language models can be equivalent or different. If they are different, complex mappings between the DSLs and the language model constructs are required [10]. To avoid this extra effort, the classes of the high- and low-level QoS language models and the provided constructs within the high- and low-level QoS languages in this example, are assumed to be equal.

4.2. Using the QoS DSL: An Example

In this section we want to demonstrate how the language models and the DSLs are connected, so that domain and technical experts can use the appropriately tailored high- and low-level DSLs, respectively. The following DSLs were developed and used within Frag [17].

4.2.1 Using the High-Level QoS DSL

In the high-level language, the domain experts can assign QoS measurements to Web services, define SLAs, and define actions which should be executed if a violation against an SLA occurs. Figure 5 gives a quite technical view of the high-level language, illustrated as Frag code [17].

```
## define a service and
## add some measurements to it
Service create QoSService
  -measure [ResponseTime create QoSResponseTime
    -assert [SLA create ResponseAssertion
      -set predicate "LONGER THAN"
      -set value "10"
      -set unit "SECONDS"
      -set actions [Mail create SendMailToProvider
        -set mailto "admin@provider"]]]
```

Figure 5. Assign QoS measurements to a service by using the high-level QoS DSL

In our example, a Web service, `QoSService`, is created and it is specified that the `ResponseTime` of the service should be measured. An SLA assertion, `ResponseAssertion`, is assigned to the measured response time and which should be performed if the response time is `LONGER THAN 10 SECONDS`. The idea of specifying a predicate (e.g., `LONGER THAN`), a value (e.g., `10`), and a unit (e.g., `SECONDS`) for SLA assertions is taken from [3]. The action to be performed is to sent a `Mail` to the service provider which has the e-mail address `admin@provider`.

Based on the technical view of the high-level QoS DSL in Figure 5, better understandable textual or graphical user interfaces can be generated automatically. A possible visualization of the technical view is illustraded in Figure 6, which was generated by GraphViz [1].

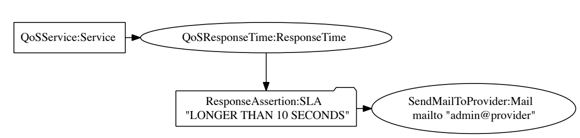

Figure 6. A possible graphical view of the high-level DSL

4.2.2 Using the Low-Level QoS DSL

Using the low-level QoS DSL helps the technical experts to specify how messages flow – between service client and service provider – within the Apache CXF Web service framework [16]. The service client and service provider sides have in- and out-flows, where in-flows are responsible for handling incoming messages, and out-flows are responsible for handling outgoing messages. In- and out-flows consist of phases. After specifying the phases, the technical expert defines where each QoS value has to be measured. Listing 7 provides an excerpt of the usage of the low-level language that concentrates on the description of how the response time can be measured in the Apache CXF Web service framework [16].

```
## define message flows of client
ClientFlow create ClientInFlow -superclasses ClientFlow
ClientFlow create ClientOutFlow -superclasses ClientFlow
...
## define phases of the message flows
OutPhase create OutSetup
OutPhase create OutSetupEnding
...
## assign phases to message flows
ClientOutFlow phases {OutSetup OutSetupEnding}
...
## define in which phases
## the response time is measured
ResponseTime measuredInFlows {ClientOutFlow}
ResponseTime measuredBetweenPhases {
  OutSetup OutSetupEnding
}
```

Figure 7. Specifying technological requirements by using the low-level QoS DSL

First, the in- and out-flows of the service client, `ClientInFlow` and `ClientOutFlow`, are specified. Then the phases of the out-flow, `OutSetup` and `OutSetupEnding`, are defined and assigned to the out-flow of the client, `ClientOutFlow`. Finally, the flows and phases, between which the `ResponseTime` is measured, are specified.

5. Lessons Learned

This section explains the, during the development of the above described QoS DSL, discovered benefits and drawbacks which are important within our approach and which have to be considered in our future work.

One of the big advantages of the separation into high- and low-level languages, as proposed in the example of this paper, is that the technical experts have to specify the technological aspects just once. For instance, the response time is measured within the defined phases every time, independent of the SLAs specified in the high-level language. Hence, the SLAs can be specified multiple times without changing any technological aspects. Furthermore, a common advantage of model-driven DSL approaches is that the language models are easily extensible. Hence, when following our approach, each language model can be extended separately in an easy way.

A drawback is that technological requirements have to be redefined, or at worst remodeled, when the technologies get changed or maybe by an upgrade of the current version of the used technology. But, the requirements within a domain change much more often than the technological requirements. Another discovered disadvantage lies in the overlapping concerns between the different language layers when a separation into multiple sub-languages is provided. To find a remedy, model-driven DSL approaches provide facilities for extending high-level concerns with low-level concerns or vice versa, such as inheritance, associations, or compositions.

As shown, model-driven DSL approaches can suppress the arising drawbacks of providing multiple languages which are tailored for the appropriate stakeholders. The following section mentions some related work and their differences to our approach.

6. Related Work

This section is divided into three parts, where each part refers to work that has been done or that is still in progress with respect to our main contributions of this work.

MDSD-based DSLs:
Kelly and Tolvanen [8] illustrate their collected experiences of designing and developing Domain-specific Modeling Languages (DSML) by five examples. In contrast to our work, their code generators aim to provide full code generation from the defined models of the domain experts. Hence, technical experts are not involved in the modeling process. Dependent on the problem, modeling (service-oriented) business systems without technical stakeholders can be a drawback.

Tailoring DSLs for Various Stakeholders:
Voelkl et al. [9] write about the different roles in the software development process with domain-specific modeling languages (DSML). An introduction to the MontiCore framework is given, which is a code generator and a language processing environment. Language developers can define the syntax of the modeling language in form of a context-free grammar. Within our approach, the syntax of the DSLs is expressed by language models which facilitates the definition of the DSL syntax.

Even though the realization of this approach is different to ours, the idea of this work is similar, as Freudenstein et al. [13] also support multiple stakeholders within their DSL approaches for modeling Web applications.

Defining or Modeling QoS and SLAs:
Rosenberg et al. [3] propose a top-down modeling approach

for capturing functional and non-functional QoS concerns of Web service based business processes. Their approach is based on WS-CDL which is transformed to BPEL code. This approach does not provide multiple separated and tailored languages for technical and non-technical stakeholders.

The following two approaches are extension to UML. The Object Management Group (OMG) [11] introduces a UML profile for modeling QoS. Their QoS framework is separated into three packages: *QoSCharacteristics*, *QoSConstraints*, and *QoSLevels*. A Service-oriented architecture Modeling Language (SoaML) is presented in [12]. This language is also a UML profile and provides the facility for modeling `ServiceContracts` between service providers and consumers. In contrast to our approach, the use of one QoS UML profile requires background knowledge of the UML which is difficult to understand for non-technical stakeholders.

7. Conclusion and Further Work

In this paper, we presented an approach for tailoring model-driven DSLs for various stakeholders with different background and knowledge. The approach is demonstrated by using a DSL for specifying QoS concerns of service-oriented business systems. The DSL was separated and tailored for two different kinds of stakeholders, i.e., domain and technical experts. One language – the high-level language – was tailored for domain experts and provides constructs for specifying the SLAs and actions which should be performed if an SLA becomes violated. The second language – the low-level language – was tailored for technical experts and provides constructs for specifying how the different QoS values have to be measured and how the actions are performed on a particular platform or technology.

The example shows that it is possible to develop a framework or notation for modeling business systems with contracts and agreements they have to comply to. By following our approach, multiple stakeholders – technical and non-technical – with different background and knowledge can be involved in the modeling process.

As future work we envision the adoptation of the presented QoS DSL to its foreseen users to get feedback of the expressiveness of our DSLs from various stakeholders. Also, we want to provide an automatic generation of easily understandable user interfaces based on the language models as shown in Figure 6. Finally, we want to support defining QoS policies, facing the challenges introduced in [6].

Acknowledgement:
This work was supported by the European Union FP7 project COMPAS, grant no. 215175.

References

[1] AT&T Research. Graphviz - Graph Visualization Software. http://www.graphviz.org/.

[2] Benoît Langlois, Consuela-Elena Jitia, Eric Jouenne. DSL Classification. In *OOPSLA 7th Workshop on Domain Specific Modeling*, 2007.

[3] Florian Rosenberg and Christian Enzi and Anton Michlmayr and Christian Platzer and Schahram Dustdar. Integrating Quality of Service Aspects in Top-Down Business Process Development Using WS-CDL and WS-BPEL. In *EDOC '07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, page 15, Washington, DC, USA, 2007. IEEE Computer Society.

[4] Florian Rosenberg and Christian Platzer and Schahram Dustdar. Bootstrapping Performance and Dependability Attributes of Web Services. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 205–212, Washington, DC, USA, 2006. IEEE Computer Society.

[5] L. jie Jin, V. Machiraju, and A. Sahai. Analysis on Service Level Agreement of Web Services. Technical report, HP Laboratories, 2002.

[6] Joe Hoffert and Douglas Schmidt and Aniruddha Gokhale. A QoS policy configuration modeling language for publish/-subscribe middleware platforms. In *DEBS '07: Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 140–145, New York, NY, USA, 2007. ACM.

[7] Juha-Pekka Tolvanen. Domain-Specific Modeling: How to Start Defining Your Own Language, 2008. http://www.devx.com/enterprise/Article/30550.

[8] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, March 2008.

[9] H. Krahn, B. Rumpe, and S. Völkel. Roles in Software Development using Domain Specific Modelling Languages. In *In Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling 2006*, pages 150–158, 2006.

[10] E. Oberortner, U. Zdun, and S. Dustdar. Domain-Specific Languages for Service-Oriented Architectures: An Explorative Study. In *ServiceWave*, pages 159–170, 2008.

[11] OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms.

[12] OMG. Service oriented architecture Modeling Language (SoaML) – Specification for the UML Profile and Metamodel for Services (UPMS), 2008.

[13] Patrick Freudenstein and Jan Buck and Martin Nussbaumer and Martin Gaedke. Model-driven Construction of Workflow-based Web Applications with Domain-specific Languages. In *MDWE*, 2007.

[14] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[15] T. Stahl and M. Voelter. *Model-Driven Software Development*. J. Wiley and Sons Ltd., 2006.

[16] The Apache Software Foundation. Apache CXF. http://cxf.apache.org/.

[17] Uwe Zdun. The Frag Language. http://frag.sourceforge.net/.