Supporting the Evolution of Model-driven Service-oriented Systems: A Case Study on QoS-aware Process-driven SOAs

Ernst Oberortner, Uwe Zdun, Schahram Dustdar
Distributed Systems Group, Information Systems Institute
Vienna University of Technology
Vienna Austria
{*e.oberortner,zdun,dustdar*}*@infosys.tuwien.ac.at*

Agnieszka Betkowska Cavalcante, Marek Tluczek
Telcordia Poland Sp. z o.o.
Poznan, Poland
{*abetkows,mtluczek*}*@telcordia.com*

***Abstract*—Process-driven service-oriented architectures (SOA) need to cope with constant changing requirements of various compliance requirements, such as quality of service (QoS) constraints within service level agreements (SLA). To the best of our knowledge, only little evidence is available if and in how far process-driven SOAs deal with the evolution of the requirements. In this work, we evaluate an incremental and model-driven development approach on the evolution of the requirements and the domain model in the context of an industrial case study. The case study focuses on advanced telecom services that need to be compliant to QoS constraints. This paper answers questions about the applicability of the incremental development approach, the impact of requirement changes, possible drawbacks of using a non-incremental development approach, and general recommendations based on the findings. Our results provide guidelines for dealing with the evolution of model-driven service-oriented systems.**

***Keywords*-service-oriented architecture; case study; domain-specific language; model-driven development; model evolution;**

I. Introduction

Process-driven service-oriented architectures (SOA) are often facilitated to automate and enhance the core functionality of an enterprise's business processes. Developing and maintaining process-driven SOAs is done by differently skilled stakeholders, such as technical or non-technical experts. Furthermore, enterprise-wide requirements must be supported, such as compliance governance, internal policies, and risk management. Mostly, the compliance requirements are fuzzy and incomplete at an early development stage. Reasons can be the different domain interpretations by the stakeholders or the evolving business and technological requirements. Permanent communications between the stakeholders and the developers result in changing requirements. To avoid complex and time-consuming updates, an incremental development approach is desired to keep the changes small and lightweight in later development stages.

In this paper we focus on an incremental model-driven development approach within an industrial case study. The case study focuses on advanced telecom services that have to comply to quality of service (QoS) constraints within service level agreements (SLA). We use the model-driven development (MDD) paradigm [8] for capturing the various SOA requirements. To support the different stakeholders in specifying the QoS compliance concerns, we utilize a domain-specific language (DSL). The development of a model-driven DSL – from now on called just DSL – starts in our approach with the definition of its language model, that is the domain model [3]. During the case study's implementation, the requirements evolved and the knowledge about the QoS domain matured, leading to unavoidable changes of the domain model and the DSL.

We state questions about (1) the applicability of an incremental approach, (2) the impact of changing requirement in later development stages, (3) the handicaps of using an non-incremental development approach, and (4) general recommendations in the area of model-driven service-oriented systems. We try to answer these questions based on the findings during the evolution of our case study's implementation. The paper should be treated as a guideline for developers of similar projects.

II. An Incremental Development Approach for Model-Driven DSLs

Our incremental development approach is based on existing approaches, such as described in [9]. The approach is divided into two sub-processes. First, we start the modeling and the design of the domain model [8]. Second, we design and implement an the external DSL [3] based on the domain model. Each sub-process is further divided into the following four phases:

- In the *Collaboration* phase, the stakeholders have to collaborate with each other. During this activity, the domain concepts and the requirements are defined.
- In the *Design* phase, the developers *design* the domain model based on the requirements and the gathered understandings about the domain.
- In the *Development* phase, the language developers can *develop* the domain model and its dependent components using their favourite modeling tool or language workbench.

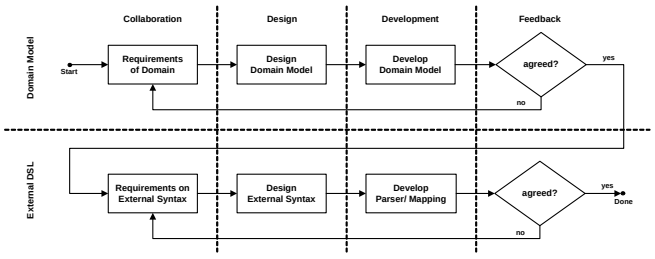

Figure 1. An incremental development approach for developing a domain model and an external DSL

- In the *Feedback* phase, the domain and technical experts give feedback to the language developers on the domain model's contained concepts.

At every stage of the development process, the stakeholders can change or extend the requirements. Hence, the DSL developers have to discuss with the domain experts the new requirements and perform changes in the domain model and its dependent components. Afterwards, the changes can be taken into the updating process of the DSL. The same incremental development approach can be used for each feature of the DSL.

III. The Case Study

The case study focuses on advanced telecom services offered by mobile virtual network operators (MVNO). An MVNO serves as a proxy between customers and the audio and video streaming providers. It offers services that process media search requests and stream the customers' favoured media content. This functionality makes it possible for the customers to watch, for example, live soccer matches with a selected audio commentary language. In Table I, we explain the offered services by the MVNO enterprise.

Service	Description
Login	This service authenticates the customers to access the system
Search	This service offers the functionality of searching movies in a favoured language
Stream	This service streams the selected movie in the selected language to the customer

Table I
The MVNO's offered services

The terms and conditions of the offered services are regulated by appropriate SLAs that contain various QoS requirements for the services. The key features of the case study are that the services have to meet particular QoS requirements. It should be possible to associate the listed MVNO services with QoS compliance concerns. For example, any service needs to be available at least 99%, i.e., running and answering the customer's requests.

It is crucial for the MVNO to monitor and avoid any potential violations with regard to the services offered to the clients. Also, the MVNO services' quality depends on the third party services' quality, making it unavoidable to monitor any performance drops of the quality of the third parties' services. Hence, monitoring QoS in this case study is a non-trivial task.

A. The Initial Version of the Case Study's Implementation

The initial QoS requirement in the industrial case study was the annotation of services with particular QoS compliance concerns. Furthermore, it was required to specify assertions for runtime QoS compliance violations. An example of a QoS requirement of the MVNO's `Search` service is that an assertion should be thrown if the Availability of the `Search` service is lower then 99%.

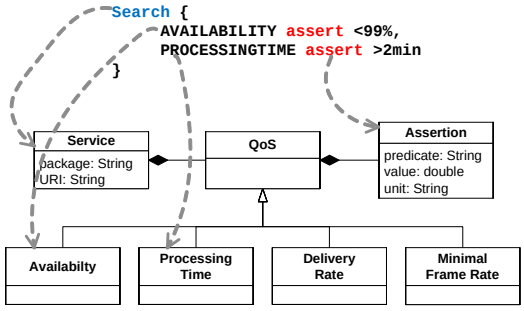

Figure 2. The initial version of the case study's implementation

We show In Figure 2 the initial version of the external DSL, the domain model model and the corresponding mappings. The external DSL has a textual and block-oriented concrete syntax. The DSL users specify services by listing the service names, annotate them with the domain model's QoS compliance concerns within the curly braces (`{...}`), and define assertions by using the `assert` keyword. The domain model's class `Service` can be instantiated to define services in the DSL, such as the `Search` service. Services can consist of QoS compliance concerns, which is assured by the composition between the `Serivce` and `QoS` classes. As required, the class `Assertion` provides the facility for defining assertions which should be thrown if QoS violations occur during the runtime of the system.

During the project's lifetime, new requirements were discovered which lead to extensions and changes of the domain model. The next section describes the extensions and changes of the domain model's initial version.

B. The Current Version of the Case Study's Implementation

The current version requires to associate QoS compliance concerns with processes that orchestrate existing services. A further requirement is to define rules of QoS conditions that are checked during the runtime of the system. If a condition is violated, the appropriate action should be performed, similar to the previous version. The reason of the introduction of rules is to be able to define gradations

of to QoS requirements that are connected with an AND logical operator. An example of QoS compliance concerns corresponding to the offered `Search` service is: If the Availability is less then 99%, then send a e-mail to the system administrator, AND if the Availability is less then 95%, then send an SMS.

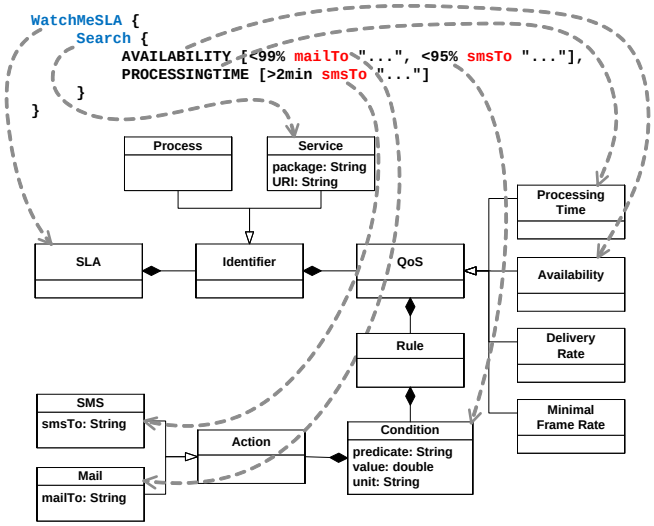

Figure 3. The current version of the case study's implementation

In Figure 3 we illustrate the current version of the external QoS DSL, the domain model, and the corresponding mapping. In the upper portion we show the current version of the external QoS DSL that has a textual and block-oriented concrete syntax. The QoS rules are specified within square brackets (`[...]`) and separated by commas (`,`). The lower portion of Figure 3 illustrates the domain model.

C. Case Study: Implementation Details

```
FMF::Class create SLA

FMF::Class create Identifier
FMF::Class create Process -superclasses Identifier
FMF::Class create Service -superclasses Identifier

FMF::Class create QoS
FMF::Class create Availability -superclasses QoS
...
FMF::Class create Rule
FMF::Class create Condition -attributes {
     predicate String
     value double
     unit String
}

FMF::Composition create IdentifierQoS -ends {
     {Identifier -aggregatingEnd true}
     {QoS -roleName qosConcerns -multiplicity * -navigable true}
}
...
```

Figure 4. The implementation of the current domain model in Frag

Figure 4 shows an excerpt of the current domain model's implementation within the language workbench Frag [10]. The classes of the domain model, such as `SLA` or `QoS`, are defined by using the `FMF::Class create` statement. The attributes of the domain model's classes, such as `predicate` of `value`, are defined by using the `-attributes` extension of the `FMF::Class create` statement. Relationships between the classes can be specified by using `FMF::Composition create` statements, such as the `IdentifierQoS` relation between the `Identifier` and `QoS` classes.

After the implementation of the domain model, the external DSL was designed and developed. The upper portion of Figure 5 illustrates an example of specifying QoS compliance concerns by using the external DSL. The lower part Figure 5 shows an excerpt of the implemented mapping that maps the parsed QoS compliance concerns onto the domain model.

```
WatchMeSLA {
     Search {
          AVAILABILITY [<99% mailTo "...", <95% smsTo "..."],
          PROCESSINGTIME [>2min smsTo "..."]
     }
}
DSL::DSLMapping create ServiceMapping -mapping {
     @rep * {
          @seq {
               @elt serviceName {$token == SERVICE_NAME} {
                    ## create an instance of the Service class
                    Service create $serviceName
               }
               @elt qosConcerns {$token == QOS_CONCERNS} {
                    ## parse the specified QoS concerns and
                    ## assign them to the Service object
                    $service qosConcerns \
                         QoSMapping map [QoSParser parse $qosConcerns] "..."
               }
     }
}
DSL::DSLMapping create QoSMapping -mapping {
     @rep * {
          @seq {
               @elt qosMeasurement {$token == QOS} {
                    if {[string compare $qosMeasurement "Availability"]==0} {
                         ## create an instance of the Availability class
                         Availability create $qosMeasurement
                    }
               }
               @elt actions {$token == ACTIONS} {
                    ## parse the actions
               }
     }
}
```

Figure 5. The implementation of the current external DSL using Frag

D. Answering the Stated Questions under Examination

Question 1: *Is the incremental development approach applicable in the case study?*

In the context of the case study, the requirements were not well defined at the beginning, making later changes unavoidable. To answer the question, the choice of following an incremental development process was successful. It was important to two-fold the incremental development process, starting with the domain model's design and followed by the development of the external DSL. After a positive feedback of the stakeholders, the language developers started to implement the external DSL on a stable version of the domain model.

Question 2: *How do changes of the requirements impact the domain model and its dependent components, such as the external DSL?*

To investigate the impact of the requirements' changes, we conduct a simple quantitative evaluation [2]. First, we calculated and compared the absolute model size of each

version of the domain model [6]. Comparing the initial and current versions, the model size increased about 80%.

We also used the lines of code (LOC) metrics of the implemented parsers and mappings for comparing them with the absolute model size of the corresponding domain models. The current DSL increased about 85% in comparison to the initial one. Comparing the enhancement of the external DSLs' parsers and mappings with the enhancement of the domain model sizes, a linear relation can be observed.

Question 3: *What are the drawbacks of a non-incremental development approach?*

In an early stage of the development process the stakeholders' requirements are not well-defined and are subject to permanent changes. Changes of the requirements change the domain model and affect its dependent components, such as code generators, parsers, or mappings. During the case study we discovered that it's not advisable to follow a non-incremental development process and to implement the whole solution at once. To answer the question, a drawback of non-incremental development approaches is that the later the changes the more complex is the development effort of new and changing requirements. Resultant, a non-incremental development approach has to deal with time-consuming maintenance phases.

Question 4: *What are general recommendations for similar projects?*

The current domain model contains four QoS compliance concerns that originate from the case study's requirements. Many other QoS measurements exists [7] that can be considered as new features or sub-domains in every iteration of an incremental development approach. Hence, changes and updates can be kept small and lightweight because the domain model can evolve in a more independent way from its dependent components.

A further recommendation is that the development of an external DSL should start when the domain experts are able to work with the domain model and all requirements are fulfilled. But, the need for code generators or an external DSL can arise at any time during the development process.

IV. Related Work

Kelly and Tolvanen [4] advice to maintain the DSL by using a pilot DSL to see the influence of the required changes. The presented incremental DSL development is in contrast to our approach tailored for their MetaEdit+ CASE tool.

Bierhoff et al. [1] describe an incremental DSL development approach, where the DSL is based on an existing system. The DSL evolves until it is expressive enough to specify the applications functionality. Our approach is designed for developing model-driven systems from scratch which evolve on changing requirements.

Kosar et al. [5] compares various DSL implementation approaches based on one DSL. The authors provide empirical results from implementing one language following ten different implementation approaches. In contrast to our approach, the implementation is considered as a sub-process of the whole development process.

V. Conclusion

In this paper we presented an industrial case study which deals with advanced telecom services and its QoS compliance concerns. In the scope of the case study we utilized DSLs and the MDD paradigm for developing a process-driven SOA, following an incremental development approach. During the case study, the QoS requirements changed and enhanced, making the maintenance of the case study's implementation complex. We stated questions that were under examination during the industrial case study's implementation. The findings during the case study's implementation gave answers to the stated questions and provide guidelines for developers of similar projects.

Acknowledgment

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

References

[1] K. Bierhoff, E. Liongosari, and K. Swaminathan. Incremental Development of a Domain-Specific Language That Supports Multiple Application Styles. In *OOPSLA – 6th Workshop on Domain Specific Modeling*, pages 67–78, October 2006.

[2] N. Fenton and S. L. Pfleeger. *Software metrics (2nd ed.): a rigorous and practical approach*. PWS Publishing Co., Boston, MA, USA, 1997.

[3] M. Fowler. Domain Specific Languages, 2009. http://martinfowler.com/dslwip/ *(last accessed: June 2010)*.

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

[5] T. Kosar, P. E. Martínez López, P. A. Barrientos, and M. Mernik. A preliminary study on various implementation approaches of domain-specific language. *Inf. Softw. Technol.*, 50(5):390–405, 2008.

[6] C. F. J. Lange. Model Size Matters. In *Workshop on Model Size Metrics at MoDELS06*, 2006.

[7] S. Ran. A model for web services discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.

[8] T. Stahl and M. Voelter. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[9] M. Strembeck and U. Zdun. An approach for the systematic development of domain-specific languages. *Softw. Pract. Exper.*, 39(15):1253–1292, 2009.

[10] U. Zdun. A DSL toolkit for deferring architectural decisions in DSL-based software design. *Information and Software Technology*, 52(7):733 – 748, 2010.