Extending SOMA for Model-Driven Software Migration into SOA

Andreas Fuhr, Tassilo Horn, Andreas Winter
University of Koblenz-Landau
{afuhr,horn,winter}@uni-koblenz.de

Rainer Gimnich
IBM Software Group, Frankfurt/Main
gimnich@de.ibm.com

Abstract

This paper proposes model-driven techniques to extend IBM's SOMA method towards migrating legacy systems into Service Oriented Architectures (SOA). The proposal explores how graph-based querying and transformation techniques enable the integration of legacy assets into the new architecture.

1 Motivation

Today, "SOA" is a buzzword often used by IT marketing. SOA is viewed as business-driven IT architecture and provides a software development and deployment paradigm promising faster development by heavily enabling reuse possibilities and the agility to adapt to changes. According to a recent poll [7], 84% of the interviewed businesses are actually using SOA or have planned to do so.

But until today, there is no broad consensus about how to design and implement a SOA and many IT businesses have been developing their own methods. One of the well known ones is "SOMA" (Service-Oriented Modeling and Architecture) developed at IBM [1]. SOMA intends to identify, design, implement and deploy services and leans to model-driven development, as there is an IBM internal tool supporting the method based on IBM Rational Software Architect (RSA). SOMA is strongly focused on extensibility and relies on dedicated methods for legacy software analysis and transformations.

This paper, focuses on a MDD based extension for a complementary method of software migration, while retaining the conceptual framework of SOMA for service quality. The reminder presents an extension of the SOMA method in order to facilitate identification of service candidates from legacy systems and transforming them to services in the new SOA infrastructure. Software migration by transformation has been proven to be an effective method to handle legacy systems [9].

2 The SOMA Method

The SOMA method is divided into the seven phases [1] described in the following paragraphs:

1. Business Modeling analyzes the state of a company at the beginning of a project. During this phase, business processes are identified.
2. Solution Management adapts the SOMA method to the actual project needs.
3. Service Identification uses three complementary methods to identify service candidates. *Domain Decomposition* is a top-down technique analyzing business processes. *Goal-Service Modeling* explores business goals to identify service candidates. Finally, *Existing Asset Analysis* is a bottom-up technique examining legacy systems coarse-grained. During this analysis, the functionality of the legacy system is explored on a high level instead of analyzing the implementation of this functionality. The objective of Existing Asset Analysis is to identify functionality that is needed as service. It is explored *what* a system does instead of *how* it is done. Activities on how to analyze legacy code are not provided by SOMA and must be contributed by an additional method.
4. Service Specification designs services and service components, along with their non functional requirements that are implemented in the SOA. The service specification (interface description of the service) is created and messages and message flows are designed.
5. Service Realization defines how services will be realized by service components. In SOMA, service components can be implemented from scratch, existing services can be used, or legacy code can be transformed or wrapped to implement a service. In this phase, the decisions on how a service will be implemented are met. The goal of this phase is to identify how functionality can be implemented or how existing functionality can be extracted and migrated into a service component. SOMA itself does not provide source code analysis techniques and their application for supporting service realization decisions. Such techniques must be complemented to SOMA.
6. During Service Implementation, the service components implementing the required services are realized. New code is written, legacy functionality is transformed or wrapped, and services are orchestrated. Activities to reuse legacy implementations e.g. by transformations or wrapping are not described in SOMA and must be added.
7. Service Deployment deals with bringing the system into use.

Concluding, SOMA does not prescribe all aspects of SOA development. But, it is designed to allow the inclusion of additional methods and techniques. The next section describes how to extend SOMA for software migration.

3 Model-driven Software Migration Techniques Leveraging SOMA

During software migration towards SOA, many different artifacts are analyzed or created. To get an consolidated view on these artifacts and enable integrated analysis, well-defined meta-models to precisely define the required artifacts and their interdependencies have to be defined [11]. The following paragraphs describe our extensions to three central SOMA phases founding on such a meta-model. As example the open-source Java tool *GanttProject* [4] was chosen which consists of about 1200 classes.

GanttProject manages tasks and resources and displays the project schedule as Gantt chart.

During *Service Identification* legacy systems are analyzed in a coarse-grained manner to identify functionality needed for service. Here we propose to analyze legacy business process models and source code by evaluating queries on models. Applying these techniques to the example one service candidate was identified: GanttProject's resource managing capabilities could be the starting point for a ResourceManagement service and provides operations for adding, editing, and deleting resources.

After specifying the service in more detail, legacy functionality which can be implemented as service is explored during *Service Realization*. In our example, the existing class HumanResourceManager (HRM in short) has been identified to be able to implement the ResourceManagement service. This class will later be transformed into a service component implementing the service. To get executable code after migrating this class, all other classes HRM depends on have to be migrated as well. Again, we propose a model-driven approach to explore these dependencies. The legacy source code is transformed into a model conforming to a call viewpoint and needed informations are extracted by evaluating queries on the model.

Java code is transformed into a TGraph [2] conforming to a Java 6 meta-model. GReQL (Graph Repository Query Language [6]) is used as query language. It is a declarative language featuring regular path expressions to describe relations between vertices. The GReQL query in Listing 1 retrieves all interfaces and classes which are used by the HRM either by method calls (methods of HRM call methods of other classes), types of fields, local variables and method parameters, and type hierarchy dependencies (HRM derives from another class or implements interfaces).

Finally, during *Service Implementation* legacy functionality identified during *Service Realization* has to be migrated into a service component. In our approach, we use model transformations to migrate legacy code into service components. The transformation language GReTL (Graph Repository Transformation Language [8]) uses GReQL queries (like in Listing 1) to capture the elements which are transformed. The transformation rule itself is operationally specified which is aligned to our meta-meta-model and slightly comparable to QVT Operational Mappings [10].

```
from hrmC : V{ClassDefinition},
     hrmM : V{MethodDefinition},
     dep  : V{Type}
with hrmC.name = "HumanResourceManager"
 and hrmC <--{IsClassBlockOf}<--{IsMemberOf} hrmM
 and (
  hrmM ( // method invocations
    <--{IsBodyOfMethod} <--{IsStatementOfBody}
    <--{AttributedEdge,^IsBreakTargetOf,
        ^IsContinueTargetOf,^IsTypeDefinitionOf}*
    & {MethodInvocation}
    <--{IsDeclarationOfInvokedMethod}
    & {MethodDefinition}
    -->{IsMemberOf} -->{IsClassBlockOf}
  ) | ( // method parameters
    <--{IsParameterOfMethod} <--{IsTypeOf}+
    <--{IsTypeDefinitionOf}
  ) | ( // local variables
    <--{IsBodyOfMethod} <--{IsStatementOfBody}
    <--{AttributedEdge,^IsBreakTargetOf,
        ^IsContinueTargetOf,^IsTypeDefinitionOf}*
    & {VariableDeclaration}
    <--{IsTypeOfVariable} <--{IsTypeDefinitionOf}
  ) dep
  or hrmC ( // member fields
    <--{IsClassBlockOf} <--{IsMemberOf}
    & {Field} <--{IsFieldCreationOf}
    <--{IsTypeOfVariable} <--{IsTypeDefinitionOf}
  ) | ( // superclass & interfaces
    (<--{IsSuperClassOfClass} |
     <--{IsInterfaceOfClass}) <--{IsTypeDefinitionOf}
  ) dep
 )
reportSet theElement(dep <--&{Identifier}).name end
```

Listing 1: GReQL query retrieving dependencies

4 Conclusion

Summarizing, SOMA is a powerful method to develop SOAs. It can easily be extended by graph-based reverse engineering and transformation techniques to enable model-driven software migration. In the SOAMIG project, these methods and extensions will be used to realize real-world migrations into a SOA [11].

References

[1] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, K. Holley. SOMA: A method for Developing Service-Oriented Solutions. *IBM Systems Journal*, 47(3):377–396, 2008.

[2] J. Ebert, V. Riediger, A. Winter. Graph Technology in Reverse Engineering, The TGraph Approach. In [5], 67–81.

[3] A. Fuhr. Model-driven Software Migration into a Service-oriented Architecture, Bachelor Thesis, Universität Mainz, 2009.

[4] GanttProject. The GanttProject, 2009. `http://ganttproject.biz/`.

[5] R. Gimnich, U. Kaiser, J. Quante, A. Winter (ed.). 10th Workshop Software Reengineering (WSR 2008), GI Lecture Notes in Informatics, Bonn, 2008.

[6] K. Marchewka. Entwurf und Definition der Graphanfragesprache GReQL2. Diploma Thesis, Universität Koblenz-Landau, Institut für Softwaretechnik, 2006.

[7] W. Martin. SOA Check 2008: Status Quo und Trends im Vergleich zum SOA Check 2007. TU Darmstadt, itresearch, 2008.

[8] F. Rheindorf. Herleitung eines operationalen Ansatzes zur Modelltransformation im Kontext modellgetriebener Softwareentwicklung. Diploma Thesis, Universität Koblenz-Landau, Institut für Softwaretechnik, 2006.

[9] W. Teppe, R. Eppig. Das ARNO Projekt, Herausforderungen und Erfahrungen in einem großen industriellen Software-Migrationsprojekt. In [5], 99–113.

[10] The Object Management Group. Meta Object Facility 2.0: Query/View/Transformation Specification, 2008.

[11] A. Winter, J. Ziemann. Model-based Migration to Service-oriented Architectures: A Project Outline. In H. Sneed (ed.) *CSMR 2007 Workshops*, 107–110, 2007.