# Model-Driven Software Migration

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

**Andreas Winter**
Carl von Ossietzky University Oldenburg
winter@informatik.uni-oldenburg.de

**Abstract:** In this paper we propose model-driven techniques to migrate legacy systems into Service-Oriented Architectures (SOA). The proposal explores how querying and transformation techniques on TGraphs enable the integration of legacy assets into a new SOA. The presented graph-based approach is applied to the identification and migration of services in an open source Java software system.

## 1 Introduction

Today, almost every company runs systems that have been implemented a long time ago and which are still adapted and maintained to meet current needs. Very often, adapting legacy software systems to new requirements also requires their migration to new technologies. Migrating legacy systems, i. e. transferring software systems to new environments *without* changing its functionality [SO08], enables already proven applications to stay on stream instead of passing away after some suspensive servicing [RB00].

A current technological advance promising better reusability of software assets is provided by *Service-Oriented Architectures (SOA)*. SOA is viewed as an abstract, business-driven approach decomposing software into loosely-coupled *services* enabling the reuse of existing software assets for rapidly changing business needs [GKMM04]. A service is viewed as an encapsulated, reusable and business-aligned asset coming with a well-defined *service specification* that provides an interface description of the functionality. The service specification is implemented by a *service component* which is realized by a *service provider*. Its functionality is used by *service consumers* [AGA+08].

Migrating legacy systems to services enables both, the reuse of already established and proven software components and the integration with newly created services, including their orchestration to support changing business needs. The application of model-driven techniques to software migration, presented here, is part of the SOAMIG project[1], which addresses the migration to Service-Oriented Architectures.

Software development and maintenance projects require a well-defined methodology. Major activities in software maintenance projects deal with legacy code. These include *legacy analysis*, i.e. understanding legacy systems and identifying reusable software assets and *legacy conversion*, i.e. migrating legacy assets. Current process models do not account for these activities, although approaches to software migration (e.g. [BS95]) are known.

This paper focuses on applying model-driven techniques to migrating legacy assets to SOAs. IBM's *SOMA method* [AGA+08] provides a process model for SOA development

[1] SOAMIG is funded by the Ministry of Education and Research (01IS09017C) cf. www.soamig.de .

and evolution, which serves here as a methodological framework to identify and extend migration activities and their technological support. Service-Oriented Modeling and Architecture (SOMA) includes seven incremental and iterative phases identifying, specifying and implementing services. In the first place, SOMA is designed to develop SOAs from scratch and does not provide support for integrating legacy assets.

The extension of SOMA towards migration, presented here, is based on model-driven strategies. Models represent different views on software systems including business process models, software architecture and programming code [WZ07]. Legacy analysis and conversion is based on queries and transformations on these models. In this paper, the *TGraph Approach* [ERW08] is applied as one possible model-driven technology. By migrating an exemplary service in the open source software GanttProject [Gan09], it will be shown how SOMA can be extended by model-driven technologies to provide a comprehensive methodology to SOA development, including a broad reuse of legacy code assets.

The paper is organized as follows: Section 2 describes the SOMA method and motivates where SOMA can be extended by model-driven techniques. Section 3 introduces the *TGraph Approach* as one possible technological space. In Section 4, the integrated method is applied to identify, specify, realize and implement one service by reusing GanttProject's legacy code. Section 5 briefly contrasts the integrated SOA migration approach presented here with current work in model-driven software analysis and migration. Finally, Section 6 summarizes and reflects the obtained results.

## 2 Service-Oriented Modeling and Architecture (SOMA)

IBM's SOMA method [AGA+08] is an iterative and incremental approach to design and implement service-oriented systems. It describes how to plan, design, implement and deploy SOA systems. SOMA is designed to be extensible in order to make use of additional, specialized techniques supporting project-specific needs. The following subsections shortly describe the seven SOMA phases and outline where they can be extended towards software migration.

**Business Modeling:** At the beginning of a project, the business is analyzed during this phase. Business goals and the business vision are identified, as well as business actors and business use cases.

*SOA migration* does not require to extend Business Modeling.

**Solution Management:** This phase adapts SOMA to the project needs. This includes choosing additional techniques to solve project-specific problems.

*SOA migration* requires to extend Solution Management: Customizing SOMA for migration requires the application of reengineering and migration techniques as depicted in the reminder.

**Service Identification:** In this phase, SOMA uses three complementary techniques to identify *service candidates*, i. e. functionality that forms a service.

*Domain Decomposition* is a top-down method decomposing the business domain into functional areas and analyzing the business processes to identify service candidates. *Goal-Service Modeling* identifies service candidates by exploring the business goals and sub-

goals. *Legacy Asset Analysis* finally explores the functionality of legacy systems. Documentation, APIs or interfaces are analyzed to identify service candidates. The source code is only analyzed coarse-grained. It is merely evaluated which functionality exists and not which components manifest the function. All three techniques are performed incrementally and iteratively. For each identified candidate, an initial service specification is created and a trace to the source of identification is established.

*SOA migration* requires to extend Service Identification: SOMA does not describe how to analyze legacy systems. In Section 4.3, we extend SOMA by reverse-engineering legacy code, which enables model-driven queries and transformations to identify service candidates including their code base.

**Service Specification:** This phase deals with describing the service design in detail. The initial service specification is refined, messages and message flows are designed and services are composed. This phase results in a comprehensive description of the service design. SOMA uses an UML profile for SOAs to describe the service design. Later, the service specification will be transformed into WSDL code in order to implement the service as a Web Service as is proposed by SOMA.

*SOA migration* requires to extend Service Specification: To gather the information needed for the service design, messages and message parameters can be derived from legacy code. We extend SOMA by queries to support design decisions in Section 4.4.

**Service Realization:** In this phase, it is decided which services will be implemented in the current iteration and it is constituted how to implement them.

After having chosen a set of services, the implementation strategy must be defined. Encapsulation of services allows to choose different ways to implement each service. Common strategies to form new service components include (1) implementation from scratch, (2) wrapping of legacy components or (3) transforming the required legacy components.

In SOMA, legacy functions are usually wrapped and then exposed as services. This has several drawbacks. The legacy system still requires maintenance, the wrapper needs to be created and requires maintenance during further evolution. Transforming the legacy code avoids this wrapping trap but requires appropriate transformation means. After deciding on transformation as implementation technique, legacy systems must be analyzed fine-grained. Functionality that is able to implement services has to be identified in the legacy code. In addition, it is important to clearly understand how this functionality is embedded in the legacy system, since it has to be separated to build a self-contained service.

*SOA migration* requires to extend Service Realization: SOMA does not consider how to implement services by reusing legacy code. In Section 4.5, a model-driven technique is presented to analyze legacy systems fine-grained in order to understand the implementation of legacy functionality.

**Service Implementation:** During Service Implementation, services are actually realized. According to the decisions derived in the Service Realization phase, services are developed, wrappers are written or legacy code is transformed. Finally, all services are orchestrated and message flows are established.

*SOA migration* requires to extend Service Implementation: SOMA does not include tech-

niques to transform legacy code into services. In Section 4.6 it is shown how transformations are used to transform legacy code into service implementations.

**Service Deployment:** The final phase of SOMA is Service Deployment. It deals with exposing services to the customer's environment. Final user-acceptance tests are executed and the SOA is monitored to verify that it performs as expected.

*SOA migration* does not require extensions of Service Deployment.

Concluding, five phases (solution management, identification, specification, realization and implementation) have been identified where SOMA must be extended to support migration of legacy systems into SOAs. The next section will describe which role model-driven development does play in extending SOMA. In Section 4, the extended SOMA method is exemplarily applied to the migration of an example Java application.

## 3 Model-Driven Development in Software Migration

Migrating legacy systems demands an integrated view on legacy code, software architecture and business processes [WZ07] to be able to identify and extract services. Therefore, an integrated representation of all these artifacts is essential. Model-driven approaches provide these technologies: (a) representation of models (metamodels), (b) querying models (query languages) and (c) transforming models (transformation languages).

Today, many model-driven approaches are known. Metamodels can be described by using the OMG's *Meta Object Facility* (MOF [OMG06]) or INRIA's KM3 [Ecl07]. Well-known transformation languages include QVT (Query/View/Transformation [OMG07]) or ATL (Atlas Transformation Language [ATL09]). All these approaches are suited for extending SOMA. However, in this paper, a graph-based approach is used which has already been applied in various reverse- and reengineering projects [ERW08].

The *TGraph Approach* is a seamless graph-based approach. Models are represented by graphs conforming to a *graph schema* (a metamodel). They can be queried with the graph query language *GReQL* (Graph Repository Querying Language [BE08]) and can be transformed using *GReTL* (Graph Repository Transformation Language [EH09]).

*TGraphs* are typed, attributed, directed and ordered graphs. Thus, they are based on a general graph model which allows appropriate tailoring for certain modeling purposes. In contrast to object-oriented modeling, edges are viewed as first-class objects. Hence, they can have attributes and traversal is always bidirectional. An API for accessing and manipulating TGraphs is given by the graph library *JGraLab*[2].

*GReQL* is a declarative graph query language and an enabling technology in reengineering, since various analyses of legacy systems can be mapped to graph queries [KW98]. One of GReQL's especially powerful features are *regular path expressions* which can be used to formulate queries that utilize the structure of interconnections between nodes and which support transitive closure.

*GReTL* is a Java framework for programming transformations on TGraphs making heavy use of GReQL to specify the mappings from source to target elements.

[2] http://jgralab.uni-koblenz.de

All these technologies are applied in Section 4 to identify, to extract and to migrate a service candidate of a Java example application within the SOMA methodology.

## 4 Extending SOMA by Model-Driven Techniques

The previous sections motivated the need of extending SOMA to enable the reuse of legacy software assets in software migration and shortly presented the TGraph approach. Our approach extends SOMA by applying model-driven techniques when appropriate.

Picking up the structure of introducing the SOMA phases in Section 2, the integrated approach is applied to the migration of one functionality of GanttProject into a Service-Oriented Architecture [Fuh09, HFW09]. GanttProject [Gan09] is a project planning tool. It manages project resources and displays project schedules as Gantt charts. GanttProject is implemented by about 1200 classes. The required migration is exemplified by identifying and migrating a service to *manage project resources* by transforming the appropriate legacy code.

### 4.1 Business Modeling

SOMA's first phase analyzes the current business situation. In this paper we focus on analysis and reuse of legacy software. Business modeling is not considered in detail, although it is important to analyze legacy business processes and to define processes to be supported by the SOA. Here, it is assumed that the business process of managing project resources shall be realized by the new SOA and its implementation will rely on GanttProject.

### 4.2 Solution Management

Solution Management adapts the SOMA method to the current project needs. Since GanttProject is a Java system, a TGraph representation for Java systems is required. Our Java 6 metamodel contains about 90 vertex and 160 edge types and covers the complete Java syntax. The GanttProject sources are parsed according to that metamodel, resulting in a graph of 274.959 nodes and 552.634 edges. This graph and the implicit knowledge on resource management, provide the foundation for service identification, service specification, service realization and service implementation.

### 4.3 Service Identification

The identification of services from legacy systems requires coarse-grained analysis. Firstly, the graphical user interface of GanttProject is explored and functionality to manage *project resources* is identified as one main feature of the software. Analyzing the GUI is only one entry point for identifying functionality and could be extended by additional explorations (e.g. test cases or documentation). Quickly scanning the legacy code then detects functionality providing the management of project resources.

Identifying pieces of functionality in legacy code is a challenging task and still an open research issue [KLS+07]. GReQL queries are used to identify functionality in the graph representation and corresponding GReTL transformations visualize the query result. String

search is used to detect possible code areas referring to “resources”. Further interconnections of code objects are specified by declarative path expressions. The resulting subgraph is transformed by GReTL into a TGraph conforming to a simple UML schema. Further XMI-based filters (cf. [EW06]) might be used to render these structures in UML tools.

```
1  VertexClass umlClass = createVertexClass("uml.Class",
2    "from t: V{Type} with t.name =~ \".*[Rr]esource.*\" reportSet t end");
3  createAttribute("name", umlClass, createStringDomain(),
4    "from t: keySet(img_uml$Class) reportMap t, t.name end");
5  createEdgeClass("uml.Association", umlClass, umlClass,
6    "from c: keySet(img_uml$Class), c2: keySet(img_uml$Class) "
7  + "with c <--{IsBlockOf} <--{IsMemberOf} <--{^IsBreakTargetOf,"
8  + "        ^IsContinueTargetOf,^IsTypeDefinitionOf,^IsClassBlockOf,"
9  + "        ^IsInterfaceBlockOf}* [<--{IsTypeDefinitionOf}] c2 "
10 + "reportSet c, c2 end",
11   "from t: $ reportMap t, first(t) end", "from t: $$ reportMap t, second(t) end");
12 createEdgeClass("uml.IsA", umlClass, umlClass,
13   "from c: keySet(img_uml$Class), c2: keySet(img_uml$Class) "
14 + "with c (<--{IsSuperClassOf} | <--{IsInterfaceOfClass}) "
15 + "        <--{IsTypeDefinitionOf} c2 "
16 + "reportSet c, c2 end",
17   "from t: $ reportMap t, first(t) end", "from t: $$ reportMap t, second(t) end");
```

Listing 1: GReTL transformation from Java to UML

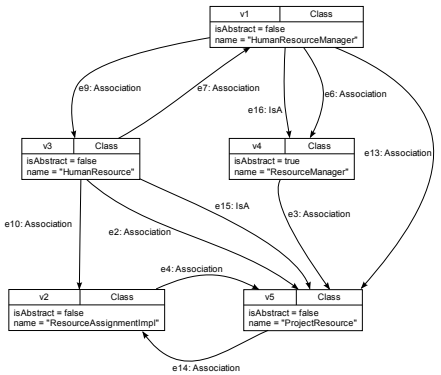
Listing 1 shows a GReTL transformation supporting coarse-grained legacy code analysis. For each legacy class or interface containing “resource” in the name, the transformation creates an UML-class node in the target TGraph (lines 1–4). In addition, associations are drawn between those class nodes whenever one node uses (e. g. by method calls or variable types) another node (5–11). Inheritance is visualized by “IsA” edges (12–17). These edges can be indirect relations since GReQL path expressions consider transitive closure. The transitive path expression in lines 7–9 (the part with $*$) matches indirect relations between classes. The visualized result of this GReTL transformation is shown in Figure 1a.

Looking at the result, the class HumanResourceManager implementing the interface ResourceManager is identified as functionality to manage project resources. Based on this information, an initial service specification for the service candidate IResourceManager is created and traces to the legacy code are noted (Figure 1b). Initial service operations for the new service are derived from the legacy interface. Only the *needed* functionality is transfered into the design. In this phase, no further information about the method signatures of the initial service specification is gathered. The following SOMA phases will specify the service in more detail.

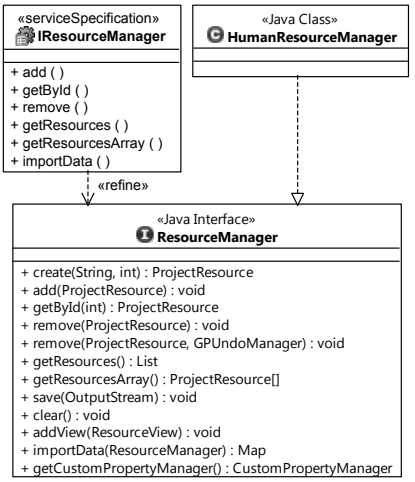
### 4.4 Service Specification

*Service Specification* refines the IResourceManager service specification. A *service provider* component is created which will later implement the service specification.

In addition, message flows are created to enable communication with the service. For *method parameters* in the legacy interface, *request messages* are created that are passed




(a) Visualization of classes and interfaces possibly providing functionality to manage resources




(b) IResourceManager service identified from legacy source code

Figure 1: Service Identification results

to the service. For *return types* in the legacy system, *response messages* are defined that will be returned by the new service. Request and response messages can be derived from legacy code.

Listing 2 shows a GReQL query taking an interface or class name as input and returning method parameters (lines 2–5) and return types (lines 6–9) as output. This information is used to derive message parameter types from legacy code.

```
1 let classname :="HumanResourceManager" in tup(
2   from hrmClass : V{ClassDefinition}, usedType : V{Type, BuiltInType}
3   with hrmClass.name = classname and hrmClass <--{IsClassBlockOf}<--{IsMemberOf}
     <--{IsParameterOfMethod} <--{IsTypeOfParameter} [<--{IsTypeDefOf}] usedType
4   reportSet (hasType(usedType, "BuiltInType")) ?
5     usedType.type : theElement(usedType<--&{Identifier}).name end,
6   from hrmClass : V{ClassDefinition}, usedType : V{Type, BuiltInType}
7   with hrmClass.name = classname and hrmClass <--{IsClassBlockOf} <--{IsMemberOf}
     <--{IsReturnTypeOf} [<--{IsTypeDefOf}] usedType
8   reportSet (hasType(usedType, "BuiltInType")) ?
9     usedType.type : theElement(usedType<--&{Identifier}).name end)
```

Listing 2: GReQL query retrieving method parameters and return types

The result of the specification phase is shown in the class diagram in Figure 2. The service specification now contains information about parameters (They are hidden in the ResourceManagerProvider component since they are already shown in the service specification). In addition, request and response messages are defined and one parameter type (HumanResourceType) for these messages is derived from legacy code.

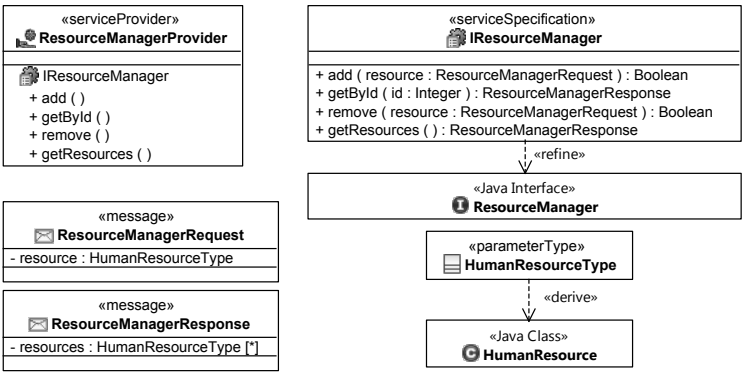



Figure 2: Detailed design of IResourceManager service

At the end of this phase, the design of the service itself is mostly completed. The next step will be to decide how the service should be implemented.

### 4.5 Service Realization

The first decision to make during Service Realization is how to implement the IResourceManager service. Model transformation approaches are also suited for code transformation. Thus, in this paper, the legacy code is transformed into a service implementation to provide the business functionality. If service realization by wrapping would have been decided on, wrappers could be generated analogously.

Service Identification has already identified one class in the legacy code that may provide functionality to the IResourceManager service: the class HumanResourceManager (short: HRM). The complete but minimal code realizing this functionality has to be determined and extracted, including all of the HRM dependencies. These include

- HRM calls methods of other classes (HRM $\rightarrow_{calls}$ method $\rightarrow_{isMemberOf}$ class),
- variables, parameters or return types (e.g. HRM $\rightarrow_{defines}$ variable $\rightarrow_{hasAsType}$ class),
- inheritance hierarchy (HRM $\rightarrow_{specializes}$ class or HRM $\rightarrow_{implements}$ interface).

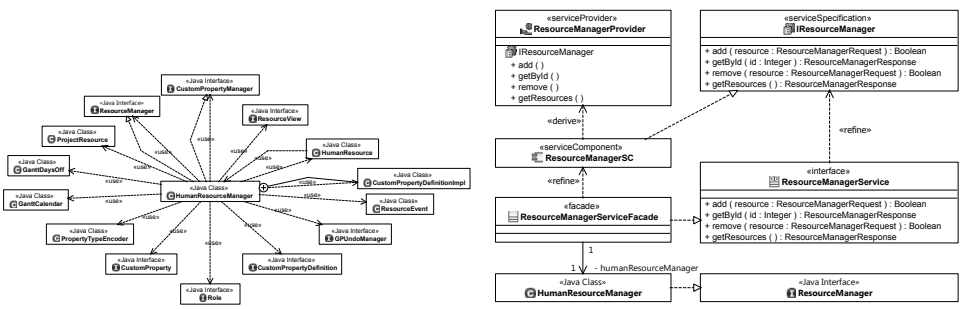
Listing 3 describes the GReQL query retrieving these dependencies. The core part of the query is the path expression in lines 4–9, defining how a dependency relation between caller and callee looks like in the Java metamodel. The query returns a list of all classes and interfaces that HRM depends on. In this example, only the directly related classes are retrieved. However, GReQL could retrieve the transitive closure of dependencies analogously. Figure 3a shows a (manually created) visualization of the query result.

Next, the business functionality must be integrated into the overall service design. This is done according to patterns proposed by Wahli [Wah07].

Figure 3b shows the application of these patterns to create a framework to integrate the legacy code which will be transformed in the next phase. The *service component* ResourceManagerSC implements the service specification. A facade pattern is used to im-

```
from hrmClass:V{ClassDefinition}, hrmMethod:V{MethodDefinition}, usedType:V{Type}
with hrmClass.name = "HumanResourceManager"
  and hrmClass <--{IsClassBlockOf} <--{IsMemberOf} hrmMethod
  and (hrmMethod ((<--{IsBodyOfMethod} <--{IsStatementOfBody} <--{^ReverseEdge}*
       <--{IsDeclarationOfInvokedMethod} -->{IsMemberOf} -->{IsClassBlockOf}
     ) | (<--{IsParameterOfMethod} <--{IsTypeOf}+ <--{IsTypeDefOf}
     ) | (<--{IsBodyOfMethod} <--{IsStatementOfBody} <--{^ReverseEdge}*
       <--{IsTypeOfVariable} <--{IsTypeDefOf}
     ) | (<--{IsReturnTypeOf} <--{IsTypeDefOf}) ) usedType
  or hrmClass ((<--{IsClassBlockOf}<--{IsMemberOf}<--{IsFieldCreationOf}
       <--{IsTypeOfVariable} <--{IsTypeDefOf}
     ) | ((<--{IsSuperClassOfClass} | <--{IsInterfaceOfClass}) <--{IsTypeDefOf}
     ) | ((<--{IsClassBlockOf} <--{IsMemberOf})+) ) usedType )
reportSet usedType end
```

Listing 3: GReQL query retrieving dependencies




(a) Dependencies of legacy HRM class

(b) Implementation design of IResourceManager

Figure 3: Service Realization: Designing the service implementation

plement the service component. The facade class will delegate service requests to the appropriate service implementation — in this example the HRM class and all its dependencies revealed by the GReQL query.

This phase finishes the design of the service. In the next step, this design will be implemented.

### 4.6 Service Implementation

Service Implementation realizes the IResourceManager service, e. g. as Web Service (as suggested by SOMA). Migrating identified source code (cf. Section 4.5) to realize the resource management service combines functionality provided by the IBM Rational Software Architect for WebSphere Software V7.5[3] (RSA) and TGraph technology.

Firstly, the code generation capabilities of the RSA are used to create WSDL code from the service specification which is later used to specify the service interface. Then, the design of the service framework (Figure 3b) is transformed into Java.

[3]IBM, Rational and WebSphere are trademarks of International Business Machines Corporation.

So far, the service lacks of business functionality, which is added by transforming legacy code into a service implementation. The GReQL query described in Listing 3 (Section 4.5) is used to mark the HRM class and all legacy software components it depends on.

The Java code generator of JGraLab is used to extract Java code for all marked classes of the TGraph. This results in Java classes implementing only the business functionality of the IResourceManager service. These classes are not connected to the service framework currently and so the facade class must be edited manually to delegate service requests to the HRM class. In addition, the facade class translates *message parameters* into *objects* known by the HRM class.

Finally, the *Web Service Wizard* of RSA is used to generate a fully functional Web Service. This wizard takes the WSDL interface description, the Java classes of the service framework and the service implementation and creates a Java EE Web Service.

### 4.7 Service Deployment

The Web Service created in the last subsection is deployed to the customer. This phase does not have to be extended by our approach. It concludes the migration.

## 5 Related Work

Whereas a plethora on publication on the development of Service-Oriented Architectures exists, migrating legacy systems to SOA is only addressed in a few papers. The SMART approach [Smi07] deals with the planning of SOA migration projects, but does not provide concrete migration or migration tool support. Correia et al. [CMHER07] and Fleurey et al. [FBB+07] describe general approaches of model-driven migration into a new technology not especially focused on SOA. Correia et al. describe a graph-based approach which mentions SOA as possible target architecture [Mat08]. In contrast to SOAMIG, this approach focuses on annotating functionality in legacy code instead of directly identifying services from source code. Marchetto and Ricca [MR08] propose an approach to migrate legacy systems into a SOA, step by step. However, this approach does not focus on model-driven techniques and uses wrapping as general migration strategy. Another approach focusing on wrapping is described in [Gim07].

In contrast to these approaches, the work presented here provides a coherent model-driven approach to software migration by integrating an established SOA forward-engineering approach with graph-based reengineering technologies. In addition, in SOAMIG software systems are viewed at all levels of abstraction including business processes and code.

## 6 Conclusion and Future Work

In this paper, we described a model-driven approach to migrate legacy systems, extending IBM's SOMA method. The approach was applied to the migration of a functionality of GanttProject towards a Service-Oriented Architecture. This example demonstrated the identification and specification of services by analyzing legacy code, the identification of responsible functionality in legacy code and the transformation of legacy code into a service implementation. As result, a fully functional Web Service was generated whose

business functionality was implemented by transforming legacy code.

The example presented in this paper should be understood as first technical proof-of-concept and not as fully developed method. The approach must be extended and the techniques must be improved. E.g. one issue is the replacement of the string-based service identification technique (which fails when source code does not follow some naming conventions) by a dynamic approach. As part of the SOAMIG project, we are currently exploring the possibility of simulating the execution of business processes while tracing source code execution. Based on these traces, source code might be mapped to processes, supporting service identification.

Another issue is the application of the approach on systems that are not written in Java. In addition to plain architectural migrations as presented in this example, languages (e.g. COBOL $\rightarrow$ Java) must be migrated in language migration projects, too. All TGraph techniques explained in this paper are generic and will work for other languages if metamodels are provided. Currently, a metamodel for COBOL is being developed and will enable our approach to cope with legacy COBOL systems as soon as it is finished. Research will also address code transformation based on these metamodels.

In contrast to "transformation capabilities" of modern tools like IBM's Rational Software Architect or Borland's Together Architect, the TGraph approach offers an integrated view on all models and allows to process all needed queries on one repository. This will allow us to create one homogeneous workflow instead of handling different types of results from different sources leading to compatibility issues.

The results of this ongoing research will have to be confirmed on larger examples in future. In collaboration with our industrial project partners, their Java and COBOL systems will be migrated into Service-Oriented Architectures.

Summarizing, the presented approach already showed first interesting results and promises to improve model-driven migration in future.

## Acknowledgements

We want to express our thanks to Rainer Gimnich, IBM Software Group, Frankfurt/Main, Germany for his support in understanding SOMA and various fruitful discussions on applying SOMA in software migration.

## References

[AGA+08] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley. SOMA: A Method for Developing Service-Oriented Solutions. *IBM Systems Journal*, 47(3):377–396, 2008.

[ATL09] ATLAS Group. ATL User Guide, 2009.

[BE08] D. Bildhauer and J. Ebert. Querying Software Abstraction Graphs. In *Working Session on Query Technologies and Applications for Program Comprehension*, 2008.

[BS95] M. L. Brodie and M. Stonebraker. *Migrating Legacy Systems, Gateways, Interfaces & The Incremental Approach.* Morgan Kaufmann, 1995.

[CMHER07] R. Correia, C. Matos, R. Heckel, and M. El-Ramly. Architecture Migration Driven by Code Categorization. In *Software Architecture, First European Conference, Spain:*

*Proceedings*, volume 4758 of *Lecture Notes in Computer Science*. Springer, 2007.

[Ecl07] Eclipse. KM3. `http://wiki.eclipse.org/KM3`, 2007.

[EH09] J. Ebert and T. Horn. The GReTL Transformation Language. Projectreport, Koblenz, 2009.

[ERW08] J. Ebert, V. Riediger, and A. Winter. Graph Technology in Reverse Engineering: The TGraph Approach. In R. Gimnich, U. Kaiser, J. Quante, and A. Winter, editors, *10th Workshop Software Reengineering*, volume 126 of *GI-Edition Proceedings*, pages 67–81, Bonn, 2008. Ges. f. Informatik.

[EW06] J. Ebert and A. Winter. Using Metamodels in Service Interoperability. In *Postproceedings of 13th Annual International Workshop on Software Technology and Engineering Practice (STEP'05)*, pages 147–156, 2006.

[FBB+07] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, and J. M. Jezequel. Model-driven Engineering for Software Migration in a Large Industrial Context. In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *International Conference on Model Driven Engineering Languages and Systems*, pages 482–497, Berlin, 2007. Springer.

[Fuh09] A. Fuhr. Model-driven Software Migration into a Service-oriented Architecture. Bachelorthesis, Mainz, 2009.

[Gan09] GanttProject. The GanttProject. `http://ganttproject.biz/`, 2009.

[Gim07] R. Gimnich. SOA Migration: Approaches and Experience. *Softwaretechnik-Trends*, 27(1), 2007.

[GKMM04] N. Gold, C. Knight, A. Mohan, and M. Munro. Understanding Service-Oriented Software. *IEEE Software*, 21(2):71–77, 2004.

[HFW09] T. Horn, A. Fuhr, and A. Winter. Towards Applying Model-Transformations and -Queries for SOA-Migration. In *Workshop MDD, SOA und IT-Management*, 2009.

[KLS+07] K. Kontogiannis, G. A. Lewis, D. B. Smith, M. Litoiu, H. Müller, S. Schuster, and E. Stroulia. The Landscape of Service-Oriented Systems: A Research Perspective. In *Proceedings of the International Workshop on Systems Development in SOA Environments*. IEEE Computer Society, 2007.

[KW98] B. Kullbach and A. Winter. Querying as an Enabling Technology in Software Reengineering. In *Proceedings of the 3nd European Conference on Software Maintenance and Reengineering*, pages 42–50. IEEE Computer Society, Los Alamitos, 1998.

[Mat08] C. Matos. Service Extraction from Legacy Systems. In D. Hutchison, H. Ehrig, R. Heckel, T. Kanade, and J. Kittler, editors, *Graph Transformations*, volume 5214, pages 505–507, Berlin, Heidelberg, 2008. Springer-Verlag.

[MR08] A. Marchetto and F. Ricca. Transforming a Java Application in a Equivalent Web-Services Based Application: Toward a Tool Supported Stepwise Approach. In *Proceedings Tenth IEEE International Symposium on Web Site Evolution, Beijing, China (WSE)*. IEEE Computer Society, 2008.

[OMG06] OMG. Meta Object Facility (MOF) 2.0: Core Specification – formal/06-01-01, 2006.

[OMG07] OMG. Meta Object Facility (MOF) 2.0: Query/View/Transformation Specification – Final Adopted Specification ptc/07-07-07, 2007.

[RB00] V. T. Rajlich and K. H. Bennett. A Staged Model for the Software Life Cycle. *Computer*, 33(7):66–71, 2000.

[Smi07] D. B. Smith. Migration of Legacy Assets to Service-Oriented Architecture Environments. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 174–175. IEEE Computer Society, 2007.

[SO08] H. M. Sneed and S. Opferkuch. Training and Certifying Software Maintainers. In *12th European Conference on Software Maintenance and Reengineering (CSMR), Athens, Greece*, pages 113–122. IEEE Computer Society, 2008.

[Wah07] U. Wahli. *Building SOA Solutions Using the Rational SDP*. IBM Redbooks. 2007.

[WZ07] A. Winter and J. Ziemann. Model-based Migration to Service-oriented Architectures: A Project Outline. In H. M. Sneed, editor, *CSMR 2007, 11th European Conference on Software Maintenance and Reengineering, Workshops*, pages 107–110, 2007.