The SOAMIG Process Model in Industrial Applications

C. Zillmann, A.Winter
OFFIS
Institute for Information Technology
{zillmann, winter}@offis.de

A. Herget, W. Teppe, M. Theurer
Amadeus Germany
{aherget, wteppe, mtheurer}@de.amadeus.com

A. Fuhr, T. Horn, V. Riediger
University of Koblenz
{afuhr, horn, riediger}@uni-koblenz.de

U. Erdmenger, U. Kaiser, D. Uhlig, Y. Zimmermann
pro et con GmbH
{uwe.erdmenger, uwe.kaiser, denis.uhlig, yvonne.zimmermann}@proetcon.de

***Abstract*—The SOAMIG Project aims at a general migration process model with an emphasis on transformation-based conversion. The SOAMIG Process Model is divided into several phases and disciplines, which describe and organize general migration activities. The process is applied in two industrial software migration projects addressing architecture and code migration.**

***Keywords*-migration, transformation, process model, soa;**

I. Motivation

Most commercially built information systems are based on traditional technologies preventing them from unfolding their full potential. Service-oriented architecture (SOA) promises an increasing flexibility of companies and enables reusing existing software assets for rapidly changing business needs [15]. Migrating legacy systems to SOA enables both, the reuse of already established and proven software components and the integration with newly created services, including their orchestration [13]. Migrating to new software architectures usually requires addressing other migration aspects like data, code and user interfaces [14].

Current process models in software development rarely account for migration activities [13]. Activities in migration projects include legacy analysis (e.g. program comprehension, identifying reusable software assets), and legacy conversion [7]. Next to these reengineering and reverse engineering activities, evolution projects address target and legacy architectures and require migration strategies. Furthermore, if e.g. SOA is aspired, the project has to deal with business process modeling and service and orchestration designs.

On the one hand, existing migration process models like [5] mostly blind out forward engineering activities. On the other hand, existing development process models like the V-Model XT, the Rational Unified Process (RUP) or IBM's SOMA method [1] solely focus on forward engineering. Hence, a well-defined adaptable methodology for migration projects is required.

The SOAMIG-Project[1] aims at providing a general transformation-based migration process model with an emphasis on code and architecture migration. The SOAMIG consortium consist of four project members: Amadeus (Germany), the leading transaction processor for the global travel and tourism industry, is the legacy expert and allocates one the legacy systems. Pro et con, a software reengineering and migration expert, is responsible for Java and COBOL migration and tool development. University of Koblenz is the expert for graph-based technologies and responsible for development of program analysis and transformation tools and provides rules for a technical migration. OFFIS contributes expert knowledge on enterprise and software architecture, is responsible for the target architecture development, SOA realization, and provides in cooperation with University of Koblenz for the generic SOAMIG Migration process.

This paper introduces the SOAMIG Process Model (Sec. II) followed by an industrial architecture migration (legacy to SOA) and a code migration (COBOL to Java) in Sec. III. Finally, Sec. IV summarizes the current results and sketches future activities.

II. The SOAMIG Process

SOAMIG aims at defining an adaptable iterative migration process model. The process model is developed in conjunction with an architecture migration (RAIL) and a code migration (LCOBOL).

A. The SOAMIG Phases

The SOAMIG process distinguishes four organizational phases exposing important milestones in migration projects (cf. Fig. 1). The phases collate several disciplines (cf. Sec. II-B) highlighting activities during migration.

1) Preparation: Starting point of every migration project is legacy code which has to be prepared and standardized in the *Pre-Renovation* discipline by various reengineering activities to alleviate conversion activities.

The project infrastructure including defining project goals and work packages or managing resources is set up in the *Project Setup* discipline. Migration projects require a high level of automatization by using appropriate tools. General development of reengineering and conversion tools is covered by *Tool Initialization*; their adaptation to detailed project-specific requirements is addressed in *Tool Adaptation* in *Conceptualization* phase.

[1]SOAMIG is partly funded from April 2009 to March 2011 in the German Federal Ministry of Education and Research funding program for small and medium sized enterprises (01IS09017A-D). (cf. www.soamig.de).

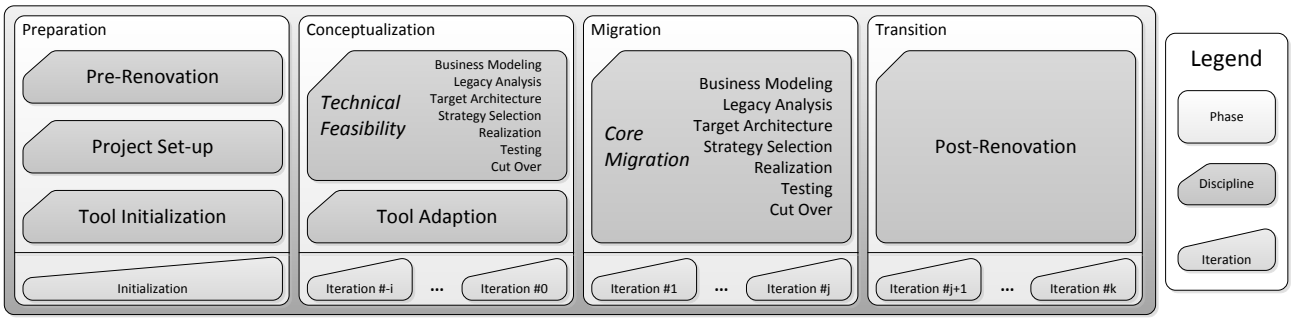

Figure 1. The SOAMIG Phases

2) Conceptualization: 70%-75% of activities in reengineering projects are independent of detailed project needs [4]. A broad automatization seems possible by eligible migration factories. A central activity in migration projects is assessing feasibility of migration and applicability of provided tool sets during *Technical Feasibility*. This discipline includes passing through all SOAMIG core disciplines (cf. Sec. II-B) for a small, but representative part of the legacy system to ensure proper provision and adaptation of migration tools.

3) Migration: Migrating the entire system is applied after setting up a general migration strategy and tool support. In the *Migration* phase, all SOAMIG core disciplines (cf. Sec. II-B) are performed iteratively in different intensities, resulting in a migrated system in production.

4) Transition: Code migration usually leads to hardly maintainable code, which requires additional reengineering. Software quality degrades by adopting mindsets from legacy to target structures directly [16]. The quality of the migrated system has to be improved in the *Post-Renovation* discipline e.g. by refactorings in the target environment.

B. The SOAMIG Core Disciplines

Seven SOAMIG Core disciplines (cf. Fig. 2) are performed during *Conceptualization* phase for a small part of the legacy system and eventually in the *Migration* phase for the entire system. Most of these disciplines use model-driven techniques based on an integrated repository [13], [18].

1) Business Modeling: One objective of SOAMIG is the migration to SOA, which requires analyzing the business processes of legacy systems to allow a reasonable tailoring of services in the *Target Architecture* discipline. The evaluation and documentation of supported business processes is handled by the *Business Modeling* discipline using UML2 activity diagrams and Business Process Modeling Notation (BPMN). These models are integrated with architecture and code models in the SOAMIG repository.

2) Legacy Analysis: Legacy Analysis deals with exploring and comprehending the legacy system. Available information like user or technical documentation, legacy test cases, architecture description and source code have to be analyzed. In SOAMIG, static and dynamic analysis techniques including FGM [2] and JGraLab/GReQL ([8], [6]) are applied. Service candidates are discovered by mapping business processes from *Business Modeling* to the legacy.

3) Target Architecture: Finding a best target architecture deals with both, the legacy system and the required software support [17] in the target system. The target architecture is iteratively approximated, starting from a technically ideal architecture and taking into account special requirements of the legacy to enable economic migration. The SOA target architecture consist of *service design*, the *realization design* and the *orchestration design*. The *service design* describes the interfaces of the target architecture services. The *realization design* describes how to implement the services or the user interfaces, and finally the *orchestration design* specifies how to orchestrate services to support business processes. In code migration projects, the target architecture is predetermined by target language structures.

4) Strategy Selection: Strategy Selection decides on the cut-over strategy, which defines delivery of (parts of) the migrated system and on the realization strategy for converting each package. Cut-over strategies vary from conversion in one go (big bang) to iterative strategies, providing stepwise migration and calling for bridging architectures to enable collaboration of parts of legacy and target system [5]. Choosing the most suitable migration strategy is an important step (cf. [7]). Performing iterative migrations also includes deciding on the parts of the system to be migrated in each iteration. The realization strategy addresses the conversion of each migration package. This includes project, package and service realization strategies. Alternative strategies are reimplementation, transformation-based conversion, and wrapping. The corresponding strategy is selected according quality and business value of each migration package [3]. SOAMIG primarily focuses on transformation-based conversion, which requires the development of appropriate converters in LCOBOL (cf. Sec. III-B).

5) Realization: During *Realization*, functionality of the legacy system is converted to the target system according to the realization strategy selected in *Strategy Selection*. Migration projects deal with migrating functionality, user interfaces and data, etc. SOAMIG especially focuses on transformation-based migration. So, it is aspired to convert as much code as possible by an automated transformation using SOAMIG converters and translators. In SOA migrations, a special focus lies on services and service

orchestration. Whereas service functionality could be extracted and migrated (semi-)automatically, the orchestration of services usually has to be newly implemented since monolithic (non-SOA) legacy systems lack the required orchestration information.

6) Testing: Testing deals with ensuring equivalent behavior of legacy and migrated system by applying regression tests from the legacy system to the migrated system. System tests account for correctness within the target environment. The chosen testing strategies depend on the embedding of the migrated system. For stand-alone-systems, test cases are derived by recording messages from the legacy and replaying them to the target system. Systems embedded in a landscape of systems additionally requires effort to simulate interactions with all surrounding systems [16].

7) Cut Over: Cut Over concludes the core migration in SOAMIG. The migrated system is deployed at the customer's site, while the legacy system is turned off. To keep decisions and results based on the legacy system comprehensible for future analysis, in some cases, the legacy has to be preserved. Cut Over follows the cut-over strategy selected in *Strategy Selection*. A fallback strategy is required to ensure switching back to the old system without loss, if serious errors occur during migration. This also includes a reverse migration procedure to reconvert e.g. data changes already made in the target system before fallback [16]. In addition, the migrated system has to be monitored to ensure that it behaves as expected.

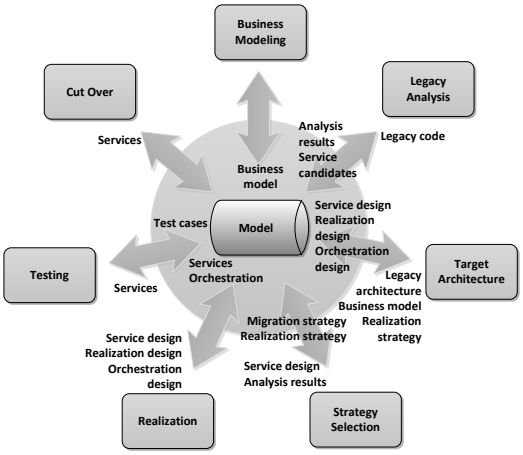

Figure 2. The SOAMIG Disciplines

Since SOAMIG aims at providing a general migration methodology incl. tool support, main activities deal with supplying and evaluating tools for SOA and code migration.

III. Applying the SOAMIG Process

The SOAMIG Process Model is validated by migrating the architecture of Amadeus Germany's RAIL-System to SOA. The ability of transformation-based code migrations is shown by a COBOL-Java migration in LCOBOL.

A. RAIL

The RAIL migration addresses the (semi-)automated transformation on architecture level. RAIL is a 229.228 LOC monolithic Java rich client offering functionality for selling Deutsche Bahn products (e.g. reservations, tickets), developed by Amadeus. For reducing deployment complexity, the system should be migrated to SOA. The migration is currently in *Conceptualization* and addresses *Technical Feasibility* to set up and evaluate a migration factory and prepare the migration of the entire system.

According to the *Preparation* phase, the migration started with setting up existing and newly developed tools for analyzing and migrating RAIL. Finally, a three-step tool chain was developed and adapted in *Tool Initialization* and *Tool Adaptation* including (1) JavaFE [18] for extracting facts from the legacy system, (2) JGralab/GReQL ([8], [6]) and Flow Graph Manipulator [2] for analysis and conversion, and (3) JGen [18], which re-generates Java code from the stored facts. These facts are stored for further analysis and conversion in a graph-based repository, which provides access to all tool components by an XML-based data exchange. The repository is capable to represent code facts in Java-graphs integrated with corresponding graph-based architecture and business process models created by conventional CASE-tools.

Next to this tool chain, further newly developed tools were applied, like the *Business Process Tracer* for mapping the modeled business processes to code artifacts during dynamic analysis. The *SoamigExtractor* identifies, selects, and extracts Java fragments in RAIL to be converted to services. Selection is controlled by the business process model and information gathered during dynamic analysis. By using graph-based GReQL queries [8] SoamigExtractor delivers slices of the original Java program represented by subgraphs, which are then converted to Java by *JGen*.

The *Technical Feasibility* phase considers migrating the functionality of selling a specific product of Deutsche Bahn: "Ticket with Timetable (TwT)". *Business Modeling* provided a process model of TwT with UML activity diagrams [11]. The *Business Process Tracer* is used in *Legacy Analysis* to map business processes to legacy code [12]. Next to identifying business process relevant legacy code, these activities also detected deeply nested legacy components. Further static analysis with JGraLab's GReQL confirmed a close linkage especially between the business logic and the user interfaces, which effected various decisions on target architecture and migration strategy.

A classical three tier target architecture consisting of a view layer, a business process layer and an enterprise service bus layer was defined in the *Target Architecture* activity. The enterprise service bus layer offers services for e. g. ordering tickets over an external system.

Concerning the deeply nested components in the legacy system, a (semi-)automated, iterative approach was chosen in *Strategy Selection*. In *Preparation*, identified service candidates are successively converted, starting with fractional tool support. Subsequent migrations result in progressively expanding tool support to provide higher level of automation for further migrations. In *Realization*, identified services on the enterprise service bus layer

were migrated iteratively. Based on the analysis results of *Legacy Analysis*, accordant classes and methods of the legacy system were extracted using SOAMIG's migration tools. Furthermore, a new data model is currently generated (semi-)automatically. Extracted code and data model were adapted to match the service interfaces. The process orchestration of TwT on the business process layer was newly implemented. Due to the tight coupling in the legacy system and the SOAMIG focus on migrating core functionality, the user interface is implemented manually.

Currently, the most relevant parts of TwT have been migrated to SOA. It is planed to complete TwT until March 2011. Furthermore, developing a data model extractor to increase the degree of automatization is in progress. Experiences and results of the *Technical Feasibility* disciplines will be used in an entire migration of RAIL in future work.

B. LCOBOL (Legacy COBOL)

LCOBOL addresses the automated transformation of legacy software systems on code level. The project aims at migrating an 81.600 LOC MF-COBOL transaction system with embedded SQL to Java/JDBC. It is intended to migrate to a web services-based architecture. LCOBOL provides an industrial access control system and has been maintained by pro et con for years.

SOAMIG migration focuses on transformation-based approaches. Thus, LCOBOL requires preferably automated transformation techniques and accentuates *Tool Initialization* in *Preparation* and *Tool Adaptation* in *Conceptualization*. In particular, mappings from constructs in COBOL to their equivalent Java representation have to be defined [9]. Transformation strategies reflect the need of further maintenance by current maintenance staff, which results in keeping the migrated code next to its original. In contrast, enabling further extensions in the target system, requires to make use of target language specific constructs. A reasonable compromise is realized in the "COBOL→Java"-Translator, building on pro et con's migration generation chain [10].

Prior to COBOL to Java transformation, refactorings, e. g. dead code elimination and code standardizations, are applied in *Pre-Renovation* to alleviate automated migration. *Legacy Analysis*, including control flow and GoTo-analysis, was accomplished with FGM Flow Graph Manipulator [2]. FGM uses an architectural representation and fine grained representations of each COBOL program which was built during *Project Set-up*.

In LCOBOL the *Target Architecture* refers to web services, which map the TOMCAT controlled legacy application server. *Realization* in LCOBOL is based on transformation *Strategy*: Each COBOL program was translated to a set of Java classes using SOAMIG's Translator. Dynamic SQL (JDBC) in the Java code replaces embedded static SQL. Transforming the transaction monitor was based on automatically identifying middleware interfaces and mapping those to web service interfaces. In *Post Renovation* it is planned to reengineer the Java classes by appropriate refactorings. Currently, the SOAMIG Translator is a prototype. At the complete stage it is planned to achieve an automation degree of more than 80%.

IV. Conclusion

SOAMIG introduces an iterative and generic software migration process model with a special focus on conversion by transformation. Case studies showed the applicability for both, architecture migrations and code migrations. Both, RAIL and LCOBOL projects will be finished in March 2011. Experiences and results of migrating the business process TwT in SOAMIG subproject RAIL will be applied in an entire RAIL migration. Results, tools, mapping descriptions and experiences of the LCOBOL subproject will be used in further maintenance and migration projects. Finally SOAMIG shows that funding collaborative work of universities and practitioners enables industries to rely on current research results and allows research institutes researching practical application.

References

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

[2] Beier, A., Uhlig, D.: *Flow Graph Manipulator (FGM): Reverse Engineering Tool für komplexe Softwaresysteme*, Softwaretechnik-Trends, 2(29):39–40, Mai 2009.

[3] Bennett, K., Ramage, M., Munro, M.: *Decision model for legacy systems*, IEE Proc. Software 1999; 146(3):153-159

[4] Borchers, J.: *Erfahrungen mit dem Einsatz einer Reengineering Factory in einem großen Umstellungsprojekt*, HMD - Praxis der Wirtschaftsinformatik, 194:77–94, März 1997.

[5] Brodie, M. L., Stonebraker, M.: *Migrating Legacy Systems, Gateways, Interfaces and The Incremental Approach*, 1995.

[6] Ebert, J., Bildhauer, D.: *Reverse Engineering Using Graph Queries*, Graph Transformations and Model Driven Engineering, LNCS 5765, 2010

[7] De Lucia, A., Francese, R., Scanniello, G., Tortora, G.: *Developing legacy system migration methods and tools for technology transfer*, Software: Practice and Experience, 38(13):1333–1364, 2008,

[8] Ebert, J., Riediger, V., Winter, A.: *Graph Technology in Reverse Engineering, The TGraph Approach*, 10th Workshop Software Reengineering (WSR 2008), Bonn, GI. Bd. 126:67–81, 2008.

[9] Erdmenger, U.: *Vom COBOL-Server zum Java-Webservice*, Softwaretechnik-Trends, (2)30:64–65, Mai 2010.

[10] Erdmenger, U., Kaiser, U., Loos, A., Uhlig, D.: *Methoden und Werkzeuge für die Software Migration*, LNI 126:83–97, 2008.

[11] Fuhr, A., Haas, J.: *Erhebung und Modellierung von Geschäftsprozessen in RAIL*, Internal SOAMIG Report, 2009.

[12] Fuhr, A., Horn, T., Riediger, V.: *Dynamic Analysis for Model Integration*, SWT-Trends, 2(30), Mai 2010.

[13] Fuhr, A., Horn T., Winter, A.: *Model-Driven Software Migration*, In SE 2010, LNI 159:69–80, 2010.

[14] Gimnich, R., Winter, A.: *Workflows der Software Migration*, SWT-Trends 2(25):22–24, 2005.

[15] Gold, N., Knight, C., Mohan, A., et al.: *Understanding Service-Oriented Software*, IEEE Softw., 21(2):71–77, 2004.

[16] Teppe, W.: *The ARNO Project: Challenges and Experiences in a Large-Scale Industrial Software Migration Project*, 13th CSMR, IEEE CSP:149–158, 2009.

[17] Zillmann, C., Gringel, P., et al.: *Iterative Zielarchitekturdefinition in SOAMIG*, SWT-Trends, 2(30):39–40 Mai 2010.

[18] Zimmermann, Y., Uhlig, D., Kaiser, U.: *Tool- und Schnittstellenarchitektur für eine SOA-Migration*, SWT-Trends, 2(30):66–67, Mai 2010.