Model-based Migration to Service-oriented Architectures

A Project Outline

Andreas Winter

Johannes Gutenberg-Universität Mainz
Institut für Informatik,
D-55128 Mainz
http://www.gupro.de/~winter

Jörg Ziemann

Institut für Wirtschaftsinformatik im
Deutschen Forschungszentrum für künstliche Intelligenz
D-66123 Saarbrücken
http://www.iwi.uni-sb.de/

Abstract

The systematic development of service-oriented architectures in conjunction with reusing already existing software requires integration of model-based software development techniques and software migration strategies. Correspondingly, we propose an approach that combines and extends methods from enterprise modelling and reengineering for developing service-oriented architectures.

1 Motivation

Service-oriented architectures (SOA) promise flexible composition of components implementing well defined business tasks to facilitate adaptability of software systems, enabling enterprises to cope with fast changing business requirements. The development of SOA requires a profound knowledge of enterprises. *Enterprise models* describing static as well as dynamic aspects of an organization are an established means to capture such knowledge. While static elements like business functions and organizational structures of an enterprise shape individual services, dynamic business process models drive the definition of service orchestrations. *Services* specify encapsulated functionality independent from concrete implementation issues. Services may interact to provide compound services. The functionality of services is implemented by *components* fulfilling the service specification, which can easily be composed to comprehending software systems. [Arsanjani, 2004].

On the other hand, most enterprises already run software systems that are established and time-proven. These legacy systems usually are not implemented in a service-oriented way. They provide software functionality which supports enterprises business processes by monolithic and deeply interwoven software modules. Thus, evolving legacy systems towards changing business processes is limited.

Keeping legacy systems evolvable requires migration into more flexible architectures. Software migration is viewed as a transformation of software systems into a new environment, without changing its functionality [Gimnich/ Winter, 2005].

This paper outlines a research agenda towards migrating legacy systems to service oriented architectures by relating the intended enterprise model to the legacy software system.

2 Migration to SOA

Migrating legacy software systems to service-oriented architectures is influenced by the intended enterprise model and the legacy software system. In SOA migrations, the intended enterprise model defines the business needs of the target design (cf. [Ackermann et al., 2005]), which is determined by services and their interactions. The legacy system already contains required functionality of the service oriented target system.

In order to build flexible and reusable systems which obtain the full business potential of SOA, a sound methodology is required, taking into account existing systems and current business requirements of enterprises as well. This includes the definition of services and service interactions regarding business needs, reflected in the new enterprise model, combined with already implemented functionality based on earlier (probably no longer available) enterprise models. In a more technical vein, components implementing services have to be discovered and extracted from the legacy system and have to be transferred into the intended SOA implementation [Ziemann et al., 2006].

The presented approach on migration to service oriented architectures comprises technologies from software reengineering and business process modelling. Fig. 1 sketches a horseshoe model (cf. [Kazman et al., 1998]) on SOA migration.

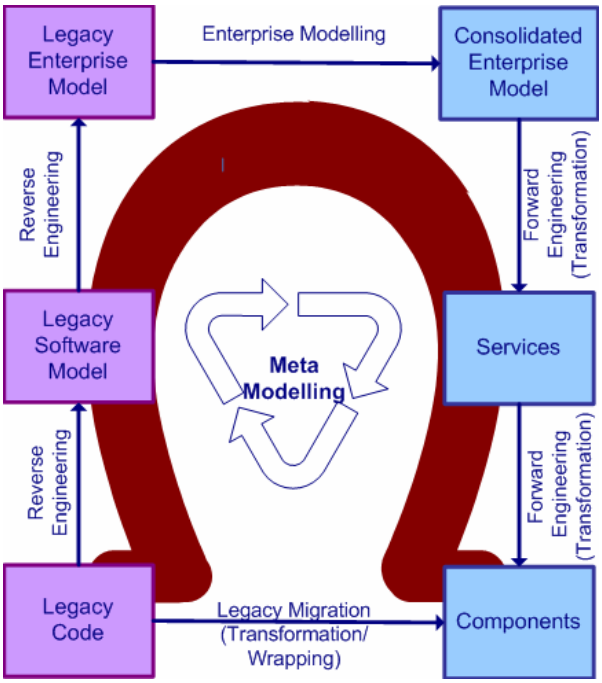

Figure 1 SOA-Migration Horseshoe

Legacy and target system are viewed on three conceptual levels showing the *code level* (Legacy Code, Components), the (platform specific) *model level* (Legacy Software Model, Services), and the *conceptual level* (Legacy Enterprise Model, Consolidated Enterprise Model). The left part presents artifacts from the legacy system. The right part presents the service oriented target system following the SOA layering [Arsanjani, 2004].

The approach follows a **model-based manner**. Software artefacts on all levels of abstraction are represented by appropriate models. These models provide the foundation for further *reverse engineering, enterprise modelling, forward engineering*, and *legacy migration* activities. A conceptual framework enabling model-based software migration requires an integrated representation of all required software artefacts. This conceptual framework can be built upon *meta modelling* technologies including facilities to model querying and model transformation.

2.1 Reverse Engineering

Reverse Engineering lifts representation of a software system on a higher level of abstraction. The main objective of the reverse engineering step in a SOA migration is to support the understanding of already existing software systems and deriving information for higher level software- and enterprise models.

Starting from its *source code*, re-documentation activities are used to re-document the existing software system on code and architectural level. A re-documented *legacy software model*, provides an important foundation to identify services provided by the legacy system. Re-documenting software systems requires the application of reverse-engineering techniques like static and dynamic code analysis, code querying [Kullbach/Winter, 1999], and re-architecting [Koschke, 2005]. Static analysis provides identifying major components and dynamic analysis will be used to determine information exchange. These reverse techniques have to be accompanied by the analysis of existing program documentation and interrogation of software developers and users.

Using additional techniques from business process modeling the software model (and other available artefacts) found the basis to derive an enterprise model for the legacy system.

From an MDA-perspective, deriving architectural representations from code can be viewed as a (backward) transformation form code to platform specific models. In the same vein, deriving the legacy enterprise model is a transformation from platform specific models to platform independent models. Due to lacking information on code and architecture, further knowledge from document analysis and interrogations of users and developers have to be included in these models. To keep the models in sync, model synchronization (e.g. [Königs/Schürr, 2005]) is needed.

2.2 Enterprise Modelling

Enterprise Modelling techniques are used to capture current enterprise requirements and to optimize the resulting models for a model-driven transformation to SOA. The requirements from the legacy system are summarized in the *legacy enterprise model.*

Deriving the legacy enterprise models requires, next to business modelling techniques, additional engineering techniques to (partially) extract those information form the *legacy software model*. Current enterprise models will be consolidated with legacy enterprise models and adapted for opportunities offered by SOA, e.g. by a stronger focus on enterprise components and process concepts realized by SOA standards like service choreography. The outcome is a *consolidated enterprise model*. This model also contains non-functional requirements to migrate to service oriented architecture and all requirements valid for those services taken over from the legacy system.

The adoption of legacy enterprise models into consolidated enterprise models can be viewed as model evolution. Due to the need, of reusing artefacts from the legacy system in the upcoming SOA system, this evolution has to keep track of all dependencies to the legacy system

2.3 Forward engineering

Forward engineering comprises all activities to realize the resulting software system. These activities include the

shaping of services aiming at optimal support of business requirements defined in the consolidated enterprise model.

A major objective in SOA migration projects is the mapping of these services to the already existing functionality provided by the legacy system. This mapping also influences the granularity of service definitions. The structure of the legacy system might provoke coarser grained services, than those, which were orchestrated by smaller services in SOA development projects from scratch.

To avoid unnecessary or uneconomical re-engineering activities, profitability analysis is required for each service. This analysis includes both, estimating the utility of the service within the SOA, and evaluating the effort and benefit of detecting and transferring legacy code fragments into the new environment. Profitability analysis also includes the decision on the migration strategy: new development of services, wrapping existing functionality (and keeping the legacy system unaffected), or transforming legacy code in new environments [Sneed et al., 2004]. Integrated models of the legacy system and the intended SOA will provide the foundation for this economical analysis.

In a model driven manner, the *consolidated enterprise model* represents the platform independent model, which is transformed into a platform specific model. In service oriented architectures, this platform specific model (*services*) specifies all required services and their interactions. Further transformation (and implementation) steps result in implementations by *components*. Since service implementations already exist in the legacy systems, this transformation step includes identification of those components from the legacy code.

2.4 Legacy Migration

Legacy Migration deals with transferring source code to a new environment. In SOA migration, this new environment is a set of SOA *components*. Here, the migration only addresses those services, which can be adopted into the target SOA environment by wrapping or transformation

Transferring legacy code in components requires the discovery of service implementation in the *legacy code*. Traces from the *Service model* back to the legacy code will provide entry points for service discovery. Using additional reengineering technologies, like querying, data-, and control flow analysis, and slicing, will be used for identifying service implementations in the legacy system.

From a model driven perspective these activities are code transformations. Program transformation systems (cf. [Visser, 2005]) like TXL [Cordy, 2001] or Stategо [Visser, 2001] and specialized transformators will be applied to migrate these programs.

2.5 Meta Modeling

These migration activities strongly rely on integrated models, representing all required software artefacts including source code, software architecture and enterprise models in the original legacy system and intended SOA system. Analyzing existing software systems, deriving higher level representations and storing additional information requires an integrated representation of all relevant software artefacts. Metamodels defining representations of software systems already exist for various modelling levels. On source-level e.g. [Dean et al, 2001], [Ferenc et al., 2001] present metamodels for C++. [Kullbach et al 1998] introduce an integrated and comprehensive view on multi-language systems using COBOL, PL1, CSP, IMS- and SQL-Databases, and JCL. The Dagstuhl Middle Model (DMM) [Lethbridge et al., 2004] provides a language independent view on software systems. In a broader sense the UML metamodel [OMG, 2005] defines a metamodel on architectural level and partially on enterprise-model level. Specialized metamodels for software architecture are defined e.g. by [Koschke, 2000] and in the context of SOA the PIM4SOA Metamodel was defined [ATHENA, 2004]. A prominent representative for metamodels on enterprise-model level is given by [Scheer, 1992].

Integrating these models according a shared metamodel provides traceability between the participating models. Traceability visualizes the interconnection between participating models. During migration to a service oriented architecture traceability facilitates the identification of appropriate chunks of legacy code to be extracted for certain services. Following traceability links from services in the service model, via the consolidated enterprise model, the legacy enterprise model, the legacy software model, and to the legacy code, establishes references between the implementation of services by components and the legacy implementation. From a business side, traceability is important to ensure the compliancy of IT systems with enterprise demands and for controlling purposes, e.g. for correlating the outputs of a certain component to a business function. If the resulting SOA implementation only wraps some legacy components, these traceability links also support further maintenance activities.

3 Conclusion

This proposal originates in the observation, that successful software migration to service oriented architectures has to bridge (business oriented) enterprise models and (technical oriented) software models. The here presented project outline tries to combine enterprise modelling and software reengineering techniques to a holistic software migration method based upon model-driven software development strategies.

4 References

[Ackermann2005ERD] Ackermann E., Gimnich, R., Winter, A.: Ein Referenz-Prozess der Software-Migration (erweiterte Kurzfassung), Softwaretechnik-Trends. 25(4). 2005, 20-22.

[ATHENA, 2004] ATHENA IP (Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications Integrated Project IST-507849): Platform-independent model for service-oriented architecture (PIM4SOA). http://pim4soa.sourceforge.net/index.html, 2004.

[Arsanjani, 2004] Arsanjani, A.: Service-oriented modeling and architecture, How to identify, specify, and realize services for your SOA, IBM, 2004, http://www-128.ibm.com/developerworks/webservices/ library/ws-soa-design1/.

[Cordy 2004] J. R. Cordy, TXL - A Language for Programming Language Tools and Applications, Proc. LDTA 2004, ACM 4th International Workshop on Language Descriptions, Tools and Applications, Barcelona, 2004.

[Dean et al., 2001] Dean, T., Malton, A., Holt, R.: Union Schemas as a Basis for a C++ Extractor, 8th Working Conference on Reverse Engineering, IEEE Computer Society, 59-67, 2001.

[Ferenc et al. 2001] Ferenc, R., Elliot. Sim, S, Holt, R. C., Koschke, R, Gyimothy, T.: Towards a Standard Schema for C/C++, 8th Working Conference on Reverse Engineering, IEEE Computer Society, 49-58, 2001.

[Gimnich/Winter, 2005] Gimnich, R., Winter, A.: Workflows der Software-Migration, Softwaretechnik-Trends. 25(2). 2005, 22-24.

[Kazman et al.,1998] Kazman R., Woods S., Carriere, J.: Requirements for Integrating Software Architecture and Reengineering Models: CORUM II, Working Confernce on Reverse Engineering, IEEE Computer, 1998, 154-163.

[Königs/Schürr, 2005] A. Königs, A. Schürr, Multi-Domain Integration with MOF and extended Tri-ple Graph Grammars, In: Bezivin, J., Heckel, R. (ed.), Language Engineering for Model-Driven Software Development, Dagstuhl Seminar Proceedings, IBFI, 2005.

[Koschke, 2000] Koschke, R,: Atomic Architectural Component Recovery for Program Understanding and Evolution, Ph.D. Thesis, Institute for Computer Science, University of Stuttgart, 2000.

[Koschke, 2005] Koschke, R.: Rekonstruktion von Software-Architekturen, Ein Literatur- und Methoden-Überblick zum Stand der Wissenschaft, Informatik, Forschung und Entwicklung, 3(19), 2005, 127-140.

[Kullbach et al.,1998] Kullbach, B., Winter, A., Dahm P., Ebert, J.: Program Comprehension in Multi-Language Systems, 5th Working Conference on Reverse Engineering, IEEE Computer Society, 135-143, 1998.,

[Kullbach/Winter, 1999] Kullbach, B., Winter, A.: Querying as an Enabling Technology in Software Reengineering, Proceedings of the 3rd Euromicro Conference on Software Maintenance & Reengineering IEEE Computer Society. Los Alamitos. 1999, 42-50.

[Lethbridge et al., 2004] Lethbridge, T. C., Tichelaar S., Ploedereder, E.: The Dagstuhl Middle Metamodel, A Schema for Reverse Engineering, in Favre J.-M., Godfrey, M, Winter, A.: Proceedings of the International Workshop on Meta-Models and Schemas for Reverse Engineering (ateM 2003), Electronic Notes in Theoretical Computer Science, Vol.~94, http://www.science-direct.com/science/journal/15710661, 7-18, 2004.

[OMG 2005] Object Management Group, Unified Modeling Language, Version 2.0, Superstructure Specification, http://www.omg.org/docs/formal/05-07-04.pdf, 2005.

[Scheer, 1992] Scheer, A.-W.: Architecture of Integrated Information Systems: Foundations of Enterprise Modeling, Springer:Berlin, 1992.

[Sneed, et al., 2004] Sneed, H. M., Hasitschka M., Teichmann, M.-T.: Software-Produktmanagement –Wartung und Weiterentwicklung bestehender Anwendungssysteme, 2004.

[Visser, 2001] Visser, E.: Stratego: a Language for Program Transformation based on Rewriting Strategies, In: A. Middeldorp (Ed.), Rewriting Techniques and Applications (RTA'01), LNCS 2051, 357-361, 2001.

[Visser, 2005] Visser, E.: A Survey of Strategies in Rule-Based Program Transformation Systems, Journal of Symbolic Computation, Special issue on Reduction Strategies in Rewriting and Programming 40(1), 831-873, 2005.

[Ziemann et al., 2006] J. Ziemann, K. Leyking, T. Kahl, D. Werth: Enterprise Model driven Migration from Legacy to SOA. In: Gimnich, R.; Winter, A.: Workshop Software-Reengineering und Services, MKWI, Passau, 2006.