4

Reverse Engineering Platform Independent Models from Business Software Applications

Rama Akkiraju[1], Tilak Mitra[2] and Usha Thulasiram[2]
[1]*IBM T. J. Watson Research Center*
[2]*IBM Global Business Services*
USA

1. Introduction

The reasons for reverse engineering software applications could be many. These include: to understand the design of the software system to improve it for future iterations, to communicate the design to others when prior documentation is either lost or does not exist or is out-dated, to understand competitors' product to replicate the design, to understand the details to discover patent infringements, to derive the meta model which can then be used to possibly translate the business application on to other platforms. Whatever the reasons, reverse engineering business applications is a tedious and complex technical activity. Reverse engineering a business application is not about analyzing code alone. It requires analysis of various aspects of a business application: the platform on which software runs, the underlying features of the platform that the software leverages, the interaction of a software system with other applications external to the software system being analyzed, the libraries and the components of the programming language as well as application development platforms that the business application uses etc. We argue that this context in which a business application runs is critical to analyzing it and understanding it for whatever end-use the analysis may be put to use. Much of the prior work on reverse engineering in software engineering field has focused on code analysis. Not much attention has been given in literature to understanding the context in which a business application runs from various perspectives such as the ones mentioned above. In our work we address this specific aspect of reverse engineering business applications.

Modern-day business applications are seldom developed from scratch. For example, they are often developed on higher-level building blocks such as programming language platforms such as J2EE in case of Java programming language and .Net in case of C# programming language. In addition most companies use even higher level application development platforms offered by vendors such as IBM's Websphere and Rational products [18][19], SAP's NetWeaver [20]and Oracles' Enterprise 2.0 software development platforms for Java J2EE application development [21] and Microsoft's .NET platform for C# programming language [22] etc. These platforms offer many in-built capabilities such as web application load balancing, resource pooling, multi-threading, and support for architectural patterns such as service-oriented architecture (SOA). All of these are part of the context in which a business application operates. Understanding this environment is crucial

to reverse engineering any software since the environment significantly influences how code gets written and managed. Reverse engineering models from business applications written on platforms that support higher level programming idioms (such as the ones noted above) is a difficult problem. If the applications developed involve several legacy systems, then reverse engineering is difficult to achieve due to the sheer nature of heterogeneity of systems. The nuances of each system may make reverse engineering difficult even if the code is built using the same programming language (e.g., Java) using the same standards (such as J2EE) on a given platform.

To understand automated reverse engineering, we must first understand model driven development/architecture [2] [3] and the transformation framework. Model driven development and code generation from models (aka *forward engineering*) has been discussed in literature. In a model driven development approach, given two meta-models, i.e., a source meta-model and a target meta-model and the transformation rules that can transform the source meta-model into the target meta-model, any given platform independent model that adheres to the source meta-model can be translated into a platform specific model (PSM) that adheres to the target meta-model. The resulting PSM can then be translated into various implementation artifacts on the target platform. This is called *forward engineering*. By reversing this approach, platform independent models can be extracted from platform specific models and implementation artifacts. Extraction of models from existing artifacts of a business application is termed *reverse engineering*. Figure 1 shows forward engineering transformation approach while Figure 2 shows reverse engineering transformation approach. The gears in the figures represent software transformations that automatically translate artifacts on the left to the artifacts on the right of the arrows they reside.

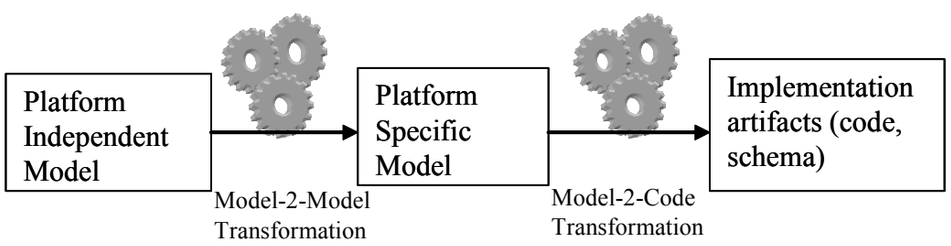

Fig. 1. Model driven transformation approach in forward engineering.

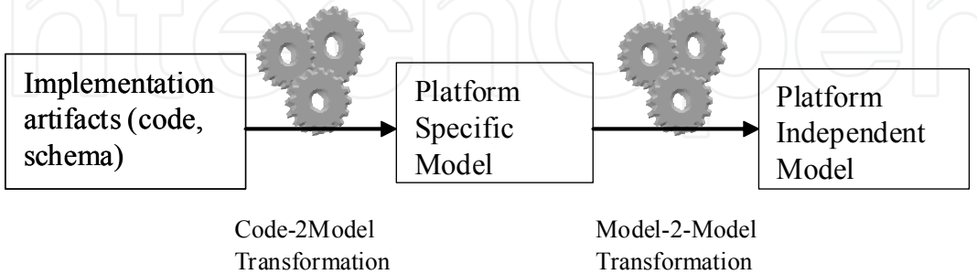

Fig. 2. Model driven transformation approach in reverse engineering.

Prior art [1] [5] [7] [10] [11] [12] and features in vendor tools such as the IBM Rational Software Architect (RSA) offer transformation methods and tools (with several gaps) to extract models. However, most of the reverse engineering work has focused on extracting

the structural models (e.g., class models) from implementation artifacts [15] [16] [17]. For example, if a UML model were to be derived from Java code, reverse engineering techniques have looked at deriving structural models such as classes, their data members and interfaces, etc. This approach, although works to a degree, does not provide a high-enough level of abstraction required to interpret the software application at a semantic level. These low level design artifacts lack the semantic context and are hard to reuse. For example, in a service-oriented architecture, modular reusable abstraction is defined at the level of services rather than classes. This distinction is important because abstraction at the level of services enables one to link the business functions offered by services with business objectives. The reusability of the reverse-engineered models with the current state-of-the-art is limited by the lack of proper linkages to higher level business objectives.

In this chapter, we present a method for extracting a platform independent model at *appropriate* levels of abstraction from a business application. The main motivation for reverse engineering in our work is to port a business application developed on one software development platform to a different one. We do this by reverse engineering the design models (we refer to them as platform independent models) from an application that is developed on one software development platform and then apply forward engineering to translate those platform independent models into platform specific models on the target platform. Reverse engineering plays an important role in this porting. While the focus of this book is more on reverse engineering, we feel that it is important to offer context to reverse engineering. Therefore, our work will present reverse engineering mainly from the point-of-view of the need to port business applications from one platform to the other. In the context of our work, a 'platform' refers to a J2EE application development platform such as the ones offered by vendors such as IBM, SAP and Oracle. In this chapter, we present a service-oriented approach to deriving platform independent models from platform specific implementations. We experimentally verify that by focusing on service level components of software design one can simplify the model extraction problem significantly while still achieving up to 40%-50% of model reusability.

The chapter is organized as follows. First, we present our motivation for reverse engineering. Then, we present our approach to reverse engineering followed by the results of our experiment in which we reverse engineer design models from the implementation artifacts of a business application developed and deployed on a specific software development platform.

2. Our motivation for reverse engineering: Cross-platform porting of software solutions

If a software solution is being designed for the first time, our objective is to be able to formally model that software solution and to generate as much of implementation/code from the model on as many software platforms as possible. This will serve our motivation to enable IT services companies to support software solution development on multiple platforms. In cases where a software solution already exists on a platform, our objective is to reuse as much of that software solution as possible in making that solution available on multiple platforms. To investigate this cross-platform portability, we have selected two development platforms namely IBM's WebSphere platform consisting of WebSphere Business Services Fabric [19] and SAP's NetWeaver Developer Studio [20].

One way to achieve, cross-platform portability of software solutions is by reusing code. Much has been talked about code reuse but the promise of code reuse is often hard to realize. This is so because code that is built on one platform may or may not be easily translated into another platform. If the programming language requirements are different for each platform or if the applications to be developed involve integrating with several custom legacy systems, then code reuse is difficult to achieve due to the sheer nature of heterogeneity. The nuances of each platform may make code reuse difficult even if the code is built using the same programming language (eg: Java) using the same standards (such as J2EE) on the source platform as is expected on the target platform. There is a tacit acknowledgement among practitioners that model reuse is more practical than code reuse. Platform independent models (PIMs) of a given set of business solutions either developed manually or extracted through automated tools from existing solutions can provide a valuable starting point for reuse. A platform independent model of a business application is a key asset for any company for future enhancements to their business processes because it gives the company a formal description of what exists. The PIM is also a key asset for IT consulting companies as well if the consulting company intends to develop pre-built solutions. The following technical question is at the heart of our work. *What aspects of the models are most reusable for cross-platform portability?* While we may not be able generalize the results from our effort on two platforms, we believe that our study still gives valuable insights and lessons that can be used for further exploration.

In the remaining portion of this section, we present our approach to cross-platform porting of software solutions.

3. Our approach to reverse engineering

Models are the main artifacts in software development. As discussed earlier, models can be used to represent various things in the software design and development lifecycle. We have discussed platform independent models (PIMs), and platform specific models (PSMs) in Introduction section. These models are at the heart of forward engineering and reverse engineering. In forward engineering, typically platform independent models are developed by humans as part of software design efforts. In reverse engineering, these models are typically derived automatically using model driven transformations. In either case, the elements that constitute a platform independent model have to be understood. Therefore, we begin with details on what constitutes platform independent models and how to build them.

3.1 Creating platform independent models

Object Management Group (OMG) provides some guidance on how to build platform independent models. Many tool vendors support the development of platform independent models. UML is the popular language of choice in the industry for representing platform independent models. In our work, we build on top of OMG's guidance on building platform independent models. We enhance the OMG modeling notions in two ways:

1. *We use a 'service' as first-class modeling construct instead of a 'class' in building the structural models*. A service is a higher level abstraction than a class. In a service-oriented architecture, the modular reusable abstraction is defined at the level of services rather

than classes. This distinction is important because abstraction at the level of services enables one to link the business functions offered by services with business objectives/performance indicators. Establishing and retaining linkages between model elements and their respective business objectives can play a significant role in model reuse. This linkage can serve as the starting point in one's search for reusable models. A service exposes its interface signature, message exchanges and any associated metadata and is often more coarse-granular than a typical class in an object-oriented paradigm. This notion of working with services rather than classes enables us to think of a business application as a composition of services. We believe that this higher level abstraction is useful when deciding which model elements need to be transformed onto the target platforms and how to leverage existing assets in a client environment. This eliminates lower level classes that are part of the detailed design from our consideration set. For code generation purposes we leverage transformations that can transform a high level design to low-level design and code. For reverse engineering purposes, we focus only on deriving higher level service element designs in addition to the class models. This provides the semantic context required to interpret the derived models.

2. *We define the vocabulary to express the user experience modeling elements using the 'service' level abstractions*. Several best practice models have been suggested about user experience modeling but no specific profile is readily available for use in expressing platform independent models. In this work, we have created a profile that defines the language for expressing user experience modeling elements. These include stereotypes for information elements and layout elements. Information elements include screen, input form, and action elements that invoke services on the server side (called service actions) and those that invoke services locally on the client (non-service actions). Layout elements include text, table and chart elements.

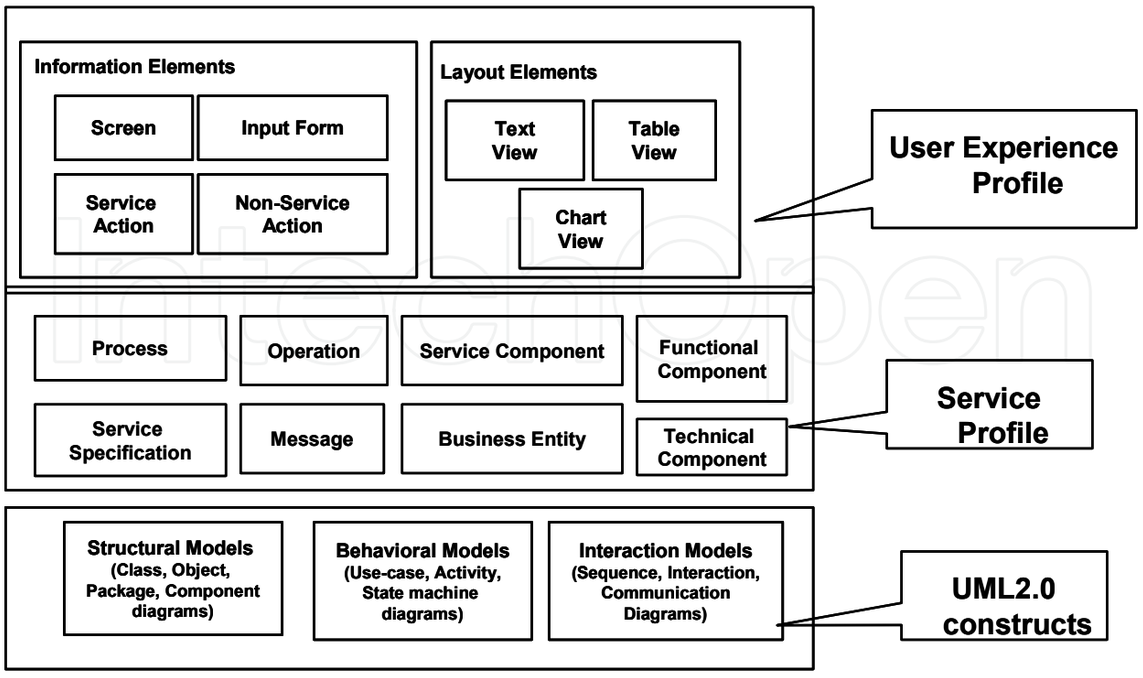

Fig. 3. Platform independent modeling elements: Our point-of-view

Figure 3 above shows the set of modeling elements that we have used to build platform independent models of a given functional specification. The bottom layer in figure 3 contains the traditional UML 2.0 modeling constructs namely the structural, behavioral and interaction models. These models are then elevated to a higher level of abstractions as *services* in a service profile. Finally, the user experience profile that we have developed based on the best practice recommendations gives us the vocabulary required to capture the user interface modules.

So far we have discussed the elements that constitute a platform independent model (PIM). To derive PIM models from implementation artifacts one typically develops model driven transformations. These transformations codify the rules that can be applied on implementation artifacts to derive models in the case of reverse engineering. In the case of forward engineering, the transformation rules codify how to translate the PIM models into implementation artifacts. In the next section, we present transformation authoring framework.

3.2 Transformation authoring

'Transformations create elements in a target model (domain) based on elements from a source model' [6]. A model driven transformation is a set of mapping rules that define how elements in a given source model map to their corresponding elements in a target domain model. These rules are specified between the source and target platform metamodels. Depending on what need to be generated there could be multiple levels of transformations such as model-to-model, model-to-text, model-to-code and code-to-model. Also, depending on the domain and the desired target platform multiple levels of transformations might be required to transform a PIM into implementation artifacts on a target platform in the case of forward engineering and vice versa for reverse engineering. For example, transformations may be required across models of the same type such as a transformation from one PSM to another PSM to add additional levels of refinement or across different levels of abstraction such as from PIM to PSM or from one type of model to another such as from PSM to code or even PIM to code. In our case, we use the traditional PIM-to-PSM and PSM-to-code transformations for forward transformations and code-to-PSM and PSM-to-PIM transformations for model extraction or reverse engineering. Operationally, multiple levels of transformations can be chained so that the intermediate results are invisible to the consumer of the transformations.

Source: Platform Independent Model (PIM) artifacts	**Target: SAP NetWeaver artifacts**
Operation	Operation
Message	InputOperationMessage, FaultOperationMessage, OutputOperationMessage
ServiceComponent	Service
Entity	BusinessObject
FunctionalComponent	BusinessObject

Table 1. Transformation mappings between the metamodels of our platform independent model and SAP NetWeaver composite application framework module.

Table 1 shows the transformation rules between the metamodels of our PIM and SAP NetWeaver composite application framework (CAF) (PSM) module. Extracting the metamodel of the target platform may not be trivial if that platform is proprietary. One may have to reverse engineer it from exemplars. We reverse engineered models from exemplars in our work. Figure 5 shows how these transformation mapping rules are developed using IBM Rational Software Architect transformation authoring tool. In this work, we developed the transformation rules manually through observation and domain analysis. Automated ways of deriving transformation rules is an active area of research [1].

Transformation Authoring for Forward Engineering: After authoring the model-to-model transformations, the target models need to be converted to implementation artifacts on the target platform. In our work, our objective was to generate Java code and database schema elements for both IBM WebSphere and SAP NetWeaver platforms. For this we have used the Eclipse Modeling Framework (EMF)'s Java Emitter Templates (JET) [6]. Templates can be constructed from fully formed exemplars. Model-to-code transformations can then use these templates to generate the implementation artifacts in the appropriate format.

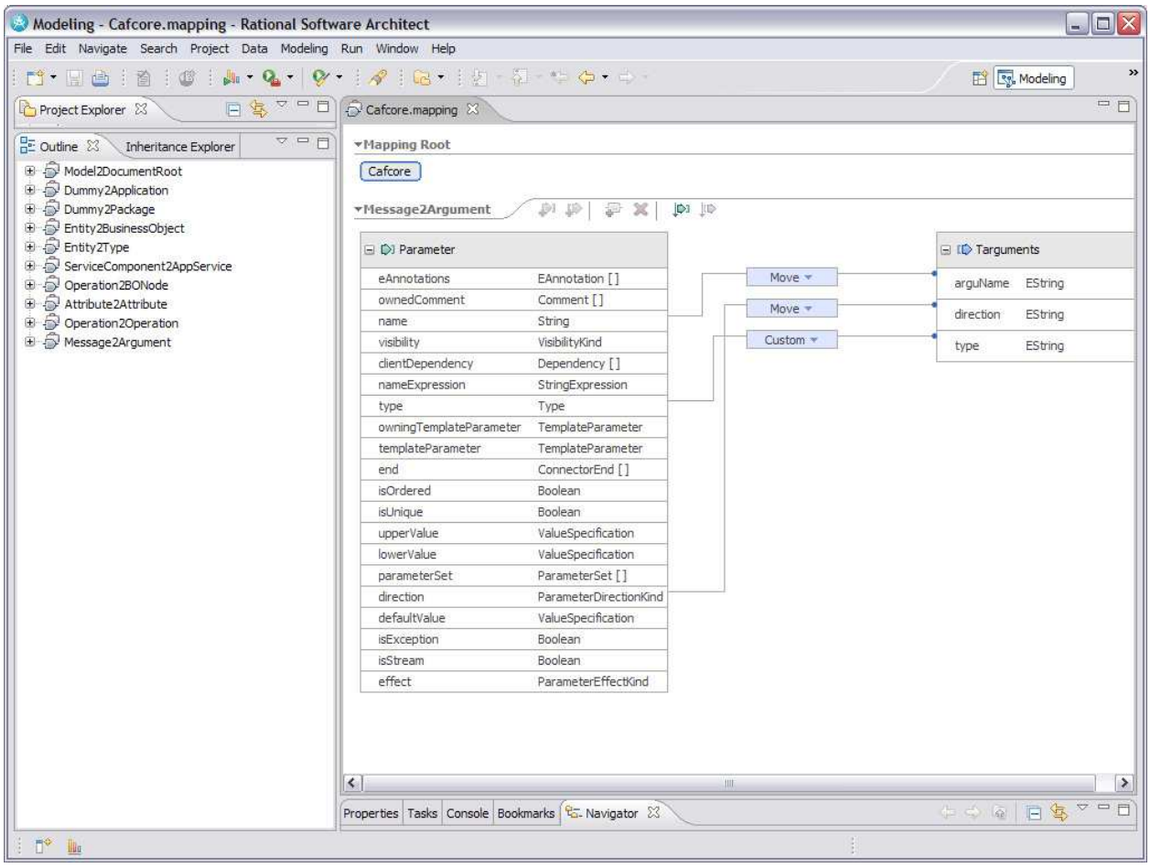

Fig. 5. A visual representation of transformation mapping rules in IBM Rational Software Architect transformation authoring tool.

As mentioned earlier, the model-2-model and model-2-code generation transformations are typically chained so that the two step process is transparent to the user.

The transformations created using mapping rules such as the ones in Table 1 which are codified using a tool such as the one shown in figure 5 can then be run by creating a specific instance of the transformation and by supplying it a specific instance of the source model (eg: A specific industry PIM). The output of this transformation is implementation artifacts on the target platform. The obtained transformations can then be imported into the target platforms and fleshed out further for deployment.

Transformation Authoring for Reverse Engineering: Figure 6 shows our approach for converting platform specific artifacts into a platform independent model. Platform specific code, artifacts, UI elements and schema are processed in a Model Generator Module to generate a platform specific model. The platform specific code, artifacts, UI elements and schema could be present in many forms and formats including code written in programming languages such as Java, or C, or C++ and schema and other artifacts represented as xml files or other files. A Model Generator Module processes the platform specific artifacts in their various formats and extracts a platform specific model from them. In order to do this, it has to know the metamodel of the underlying platform. If one exists, then the implementation artifacts can be mapped to such a platform specific model. But in cases where one does not exist, we use a semi-automated approach to derive metamodels from specific platforms.

In general, extracting the meta-models for non-standards based and proprietary platforms is an engineering challenge. Depending on the platform, varying amounts of manual effort

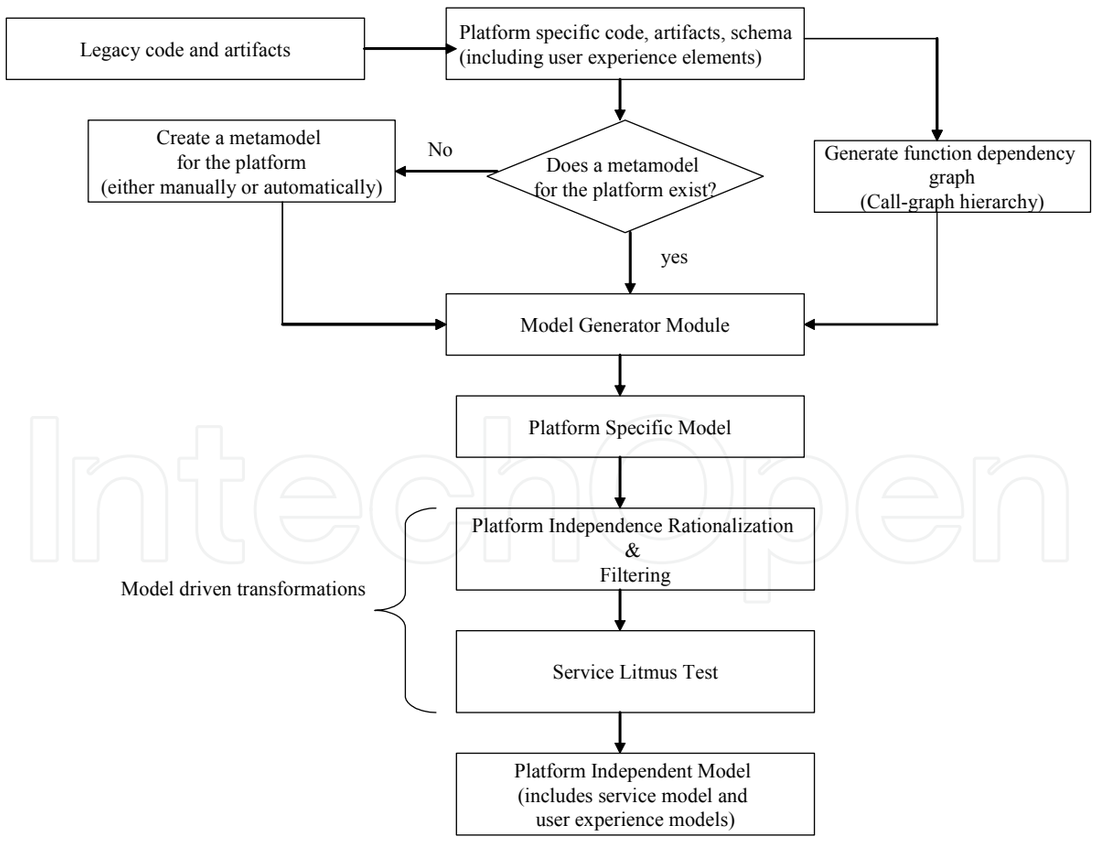

Fig. 6. **Model derivation**: Our approach to deriving platform independent models from implementation artifacts

may be required to extract the meta-modal of the platform. If the meta-models are not published or not accessible, then one may have to resort to manual observation of exemplars to derive the meta-model from the exemplar. This means an exemplar with all possible types of elements needs to be constructed. An exemplar contains the implementation artifacts which include code, schemas, xml files etc. The meta-model extraction may be automated using exemplar analysis tools available in vendor tools such as IBM's Rational Software Architect (RSA). However, an exemplar must be created first to conduct the exemplar analysis. In our work, for the two vendor platforms chosen, we were able to obtain the metamodels for one of the vendor platforms while we had to manually create the other using exemplar creation and exemplar analysis.

This metamodel is then used by the Model Generator Module to generate a platform specific model for specific model instances. Then, filtering is performed to extract only those elements that would be of 'value' at platform independent level in an SOA environment. The rationalization and filtering mechanism can employ predefined rules to perform this. For example, models of artifacts such as factory classes for business objects, and auxiliary data structures and code that setup environment variables and connectivity with legacy systems etc need not be translated onto platform independent models. These types of business objects, data structures, application services, their operations are cleansed and filtered at this stage. Then from the platform specific model, we extract service models and apply a service litmus test as given in IBM's SOMA method [4] to categorize services as process services, information services, security services, infrastructure services etc. SOMA method defines these categories of services. Each service along with its ecosystems of services can be examined in detail to derive this information either automatically or manually. Once done, additional tagging is done on services to note which ones are exposed externally and which ones are internal implementations. The litmus test can be administered manually or can be automated if there is enough semantic information about the code/artifacts to know about the behavior and characteristics. In our work, we used a user-directed mechanism for doing this filtering. A tool has been developed to enable a developer to conduct the filtering. This along with the user experience elements and models are all extracted into a platform independent model via model-driven transformations. In addition one can use code analysis tools to understand the call-graph hierarchy to retrieve an ecosystem of mutually dependent services. Several vendor tools are available for doing this for various programming languages. We use IBM's Rational Software Architect (RSA) [18] tool to do code analysis [6]. This information is captured and reflected in a platform specific model which then gets translated into a platform independent model via model driven transformations. This helps generate a service dependency model at the platform independent model. The service model and the service dependency information together provide static and the dynamic models at the platform independent level.

4. Experimental results

We hypothesize that by focusing on service level components of software design one can simplify the model extraction problem significantly while still achieving up to 40%-50% of model reusability. We have validated our hypotheses experimentally by transforming the derived platform independent model on to a different target software platform in 5 instances of business processes. This in essence is forward engineering the reverse

engineered models. We believe that this is a good measure of quality of reverse engineered models. If the reverse engineered models are 'good enough' to be used as inputs to code generation (onto another platform) that means we have made progress toward model reusability. Therefore, for our experiments we chose to put the reverse engineered models to test. The results are consistent with our hypothesis and show 40-50% of savings in development effort. The two platforms investigated are IBM WebSphere and SAP NetWeaver platforms. We tried our approach on 5 different platform independent models – either modeled or derived. On an average, we have noted that by using our transformations we can reduce the develop effort by 40%-50% in a 6 month development project (Table 2).

Phase	Activity	Effort in hours				Model-driven Transformations
		Low	Medium	High	Very High	
Develop & Deploy						
	Develop Back-end Data Objects					No
	Develop Data Dictionary Objects	4	5	7	8	No
	Develop Business Objects					No
	Develop Services					No
	Develop Back-end Services (custom RFCs/BAPIs)					No
	Develop Custom RFC/BAPI(s)	8	16	32	40	No
	Expose RFC/BAPI(s) as Web Services using Web Service Creation Wizard	0.1	0.1	0.1	0.1	No
	Publish Web Services into Service Registry (UDDI Registry)	0.1	0.1	0.1	0.1	No
	Unit Testing of back-end Services	0.25	0.25	0.25	0.25	No
	Develop Entity Services (Local BOs)					
	Development Local (CAF Layer) Business Objects with Attributes, Operations	0.5	0.75	1	1.5	Yes
	Development of Relationship amongst Business Objects	0.1	0.1	0.1	0.1	Yes
	Unit Testing of Local BOs	1	1	2	3	No
	Import External Services					
	Import RFC/BAPI into CAF Core	0.1	0.1	0.1	0.1	Yes
	Import Enterprise Services into CAF Core	0.1	0.1	0.1	0.1	Yes
	Map External Services to Operations of Business Objects	1	1	1	1	No
	Develop Application Services					
	Develop Application Services with Operations	1	2	4	6	Yes
	Map External Services to Operations of Application Services	1	1	2	2	Yes
	Implement Operations with Business Logic	8	16	24	36	No
	Expose Application Services as Web Services	0.1	0.1	0.1	0.1	Yes
	Publish Web Services into Service Registry (UDDI Registry)	0.1	0.1	0.1	0.1	No
	Unit Testing of Application Service Operations	0.5	1	2	4	No
	Deploy Entity Services into Web Application Server (WAS)					No
	Deploy Application Services into Web Application Server (WAS)		0.1			No
	Configure External Services after Deploying into Web Application Server (WAS)	1	1	2	2	No
	Develop User Interfaces					
	Develop Visual Composer(VC) based User Interfaces	4	8	16	32	No
	Implement GP Callable Object Interface	8	12	20	24	No
	Develop WebDynpro (WD) Java based User Interfaces	16	32	48	64	Yes
	Develop Adobe Interactive Form (AIF) based User Interfaces	16	24	32	48	No
	SAP NetWeaver (Composite Core + Web DynPro)					
	With Model-driven Transformations	18.9	36.15	55.4	73.9	
	Total	38.6	68.25	106.5	144	
	Percentage Generated by Model-driven Transformations	**48.96**	**52.97**	**52.02**	**51.32**	

Table 2. Catalogs the development phase activities that our transformations help automate and the development effort reductions associated with them on SAP NetWeaver platform.

Our rationale for focusing on service abstractions in models is to keep the reverse transformations simple and practical. This allows developers to develop the forward and reverse transformations relatively quickly for new platforms and programming languages. In addition, one has to consider the capabilities of various vendor software middleware platforms as well in trying to decide how much of the modeling is to be done or to be extracted. For instance, software middleware platforms these days offer the capability to generate low level design using best-practice patterns and the corresponding code given a

high-level design. So, trying to extract every aspect of a design from implementation artifacts might not be necessary depending on the target software middleware platform of choice. We believe that this insight backed by the experimental results we have shown is a key contribution of our work.

5. Conclusions

In this paper, we presented our approach to porting software solutions on multiple software middleware platforms. We propose the use of model-driven transformations to achieve cross-platform portability. We propose approaches for two scenarios. First, in cases where no software solution exists on any of the desired target middleware platforms, we advocate developing a platform independent model of the software solution in a formal modeling language such as UML and then applying model-driven transformations to generate implementation artifacts such as code and schemas from the models on the desired target platforms. Second, if a software solution already exists on one specific middleware platform, we propose applying reverse transformations to derive a platform independent model from the implementation artifacts and then applying forward transformations on the derived model to port that software solution on to a different target platform. We advance the traditional model-driven technique by presenting a service-oriented approach to deriving platform independent models from platform specific implementations.

The experiments we have conducted in deriving platform independent models from implementation artifacts have provided useful insights in a number of aspects and pointed us to future research topics in this area. The ability to leverage existing assets in a software environment depends significantly on the granularity of services modeled and exposed. Providing guidance on how granular the services should be for optimal reuse could be a topic for research. Rationalizing services that operate at different levels of granularity is another topic for further research.

6. Acknowledgements

We would like thank many of our colleagues at IBM who have contributed to related work streams which have helped inform some of the ideas presented in this paper. These colleagues include: Pankaj Dhoolia, Nilay Ghosh, Dipankar Saha, Manisha Bhandar, Shankar Kalyana, Ray Harishankar, Soham Chakroborthy, Santhosh Kumaran, Rakesh Mohan, Richard Goodwin, Shiwa Fu and Anil Nigam.

7. References

[1] Andreas Billig, Susanne Busse, Andreas Leicher, and Jörn Guy Süß;. 2004. Platform independent model transformation based on triple. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware* (Middleware '04). Springer-Verlag New York, Inc., New York, NY, USA, 493-511.

[2] Mellor, S. J., Clark, A. N., and Futagami, T. Model-driven development. IEEE Software 20, 5 (2003), 14–18.

[3] Frankel, David S.: Model Driven Architecture: Applying MDA to Enterprise Computing. OMG Press: 2003. OMG Unified Modeling Language Specification, Object Management Group, 2003,

[4] Arsanjani A.: Service Oriented Modeling and Architecture (SOMA). http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/
[5] Albert, M et al.: Model to Text Transformation in Generating Code from Rich Associations Specifications, In: Advances in Conceptual Modeling – Theory and Practice, LNCS 4231, Springer Berlin:2006.
[6] Java Emitter Templates (JET) http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html
[7] S. Sendall, W. Kozaczynski; "Model Transformation - the Heart and Soul of Model-Driven Software Development". IEEE Software, vol. 20, no. 5, September/October 2003, pp. 42-45.
[8] Sendall S. Kuster J. "*Taming Model Round-Trip Engineering*". In Proceedings of Workshop 'Best Practices for Model-Driven Software Development.
[9] Uwe Aßmann, Automatic Roundtrip Engineering, Electronic Notes in Theoretical Computer Science, Volume 82, Issue 5, April 2003, Pages 33-41, ISSN 1571-0661, 10.1016/S1571-0661(04)80732-1.
[10] Nija Shi; Olsson, R.A.; , "Reverse Engineering of Design Patterns from Java Source Code," *Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on* , vol., no., pp.123-134, 18-22 Sept. 2006.
[11] L. C. Briand, Y. Labiche, and J. Leduc. Toward the reverse engineering of uml sequence diagrams for distributed java software. *IEEE Trans. Software Eng.*, 32(9):642–663, 2006.
[12] G. Canfora, A. Cimitile, and M. Munro. Reverse engineering and reuse re-engineering. *Journal of Software Maintenance and Evolution - Research and Practice*, 6(2):53–72, 1994.
[13] R. Fiutem, P. Tonella, G. Antoniol, and E.Merlo. A clich´e based environment to support architectural reverse engineering. In *Proceedings of the International Conference on Software Maintenance*, pages 319–328. IEEE Computer Society, 1996.
[14] Gerardo CanforaHarman and Massimiliano Di Penta. 2007. New Frontiers of Reverse Engineering. In *2007 Future of Software Engineering* (FOSE '07). IEEE Computer Society, Washington, DC, USA, 326-341. DOI=10.1109/FOSE.2007.15 http://dx.doi.org/10.1109/FOSE.2007.15
[15] Atanas Rountev, Olga Volgin, and Miriam Reddoch. 2005. Static control-flow analysis for reverse engineering of UML sequence diagrams. In *Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering* (PASTE '05), Michael Ernst and Thomas Jensen (Eds.). ACM, New York, NY, USA, 96-102. DOI=10.1145/1108792.1108816 http://doi.acm.org/10.1145/1108792.1108816
[16] Rountev A., Kagan S., and Gibas, M. Static and dynamic analysis of call chains in Java. In International Symposium on Software Testing and Analysis, pages 1.11, July 2004.
[17] L. Briand, Y. Labiche, and Y. Miao. Towards the reverse engineering of UML sequence diagrams. In Working Conference on Reverse Engineering, pages 57.66, 2003.
[18] IBM Rational Software Architect. http://www.ibm.com/developerworks/rational/products/rsa/
[19] IBM WebSphere Services Fabric: http://www-01.ibm.com/software/integration/wbsf/
[20] SAP NetWeaver: Adoptive technology for the networked Fabric. http://www.sap.com/platform/netweaver/index.epx
[21] Oracle Enterprise 2.0 http://www.oracle.com/technetwork/topics/ent20/whatsnew/index.html
[22] Microsoft .Net http://www.microsoft.com/net

Reverse Engineering - Recent Advances and Applications
Edited by Dr. A.C. Telea

ISBN 978-953-51-0158-1
Hard cover, 276 pages
Publisher InTech
Published online 07, March, 2012
Published in print edition March, 2012

Reverse engineering encompasses a wide spectrum of activities aimed at extracting information on the function, structure, and behavior of man-made or natural artifacts. Increases in data sources, processing power, and improved data mining and processing algorithms have opened new fields of application for reverse engineering. In this book, we present twelve applications of reverse engineering in the software engineering, shape engineering, and medical and life sciences application domains. The book can serve as a guideline to practitioners in the above fields to the state-of-the-art in reverse engineering techniques, tools, and use-cases, as well as an overview of open challenges for reverse engineering researchers.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen