Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 211 (2008) 5–15
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

The Role of Visual Modeling and Model Transformations in Business-driven Development

Jana Koehler Rainer Hauser Jochen Küster
Ksenia Ryndina Jussi Vanhatalo Michael Wahler

IBM Zurich Research Laboratory, Business Integration Technologies
Säumerstr. 4, CH-8803 Rüschlikon, Switzerland

Abstract

This paper explores the emerging paradigm of business-driven development, which presupposes a methodology for developing IT solutions that directly satisfy business requirements and needs. At the core of business-driven development are business processes, which are usually modeled by combining graphical and textual notations. During the business-driven development process, business-process models are taken down to the IT level, where they describe the so-called choreography of services in a Service-Oriented Architecture. The derivation of a service choreography based on a business-process model is simple and straightforward for toy examples only—for realistic applications, many challenges at the methodological and technical level have to be solved. This paper explores these challenges and describes selected solutions that have been developed by the research team of the IBM Zurich Research Laboratory.

Keywords: Business-process modeling, business-driven development, Service-Oriented Architecture

1 Introduction

An improved alignment of the IT infrastructure of an enterprise with its business needs and requirements is a trend that has dominated and driven innovations in information technology over the past couple of years. Terms such as Web services [25], Service-Oriented Architecture [7], model-driven [24] and agile [18] development, and industry standards such as the Web Service Description Language, WSDL [5], and the Business Process Execution Language, BPEL [1], or recent specifications such as the Service-Component Architecture [2] all relate to this trend.

The new technologies focus on improving the agility of the enterprise software development process to match the pace at which the business needs to change in

[1] Email: koe@zurich.ibm.com, http://www.zurich.ibm.com/csc/bit

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2008.04.025

order to keep up with market trends and competition. A key to improved software development agility is the capability of IT departments to create solutions that directly realize business goals through well-designed business processes.

Business-driven development (BDD) [19] is a methodology for developing IT solutions that directly satisfy business requirements and needs. BDD requires that "a mechanism needs to be devised by which IT efforts are interlocked with business strategy and requirements through an execution framework that is standardized, well understood, and can be executed repeatedly and successfully" [19]. The main phases of such a "mechanism" are illustrated in Figure 1.

The *Model* phase comprises the identification of business goals and requirements and the modeling of the underlying business processes. The business-process models are an essential means to create a link between the business needs and the IT implementations. In the *Develop* phase, the business-process models are refined through a number of transformations until an implementation is obtained that can then be integrated with the existing IT infrastructure in the *Deploy* phase. The resulting deployed solution is *monitored* to measure how it achieves the originally stated business goals. Finally, needs for changes and *adaptation* of the running business processes can be derived and fed back into the original business-process models. In the BDD process, business requirements flow downwards from the business level to the IT level, while IT requirements flow upwards from the IT level to the business level. What sounds so straightforward and easy in theory turns out to be a very challenging endeavor in practice. In this paper, we will focus on the two predominant of the challenges encountered:

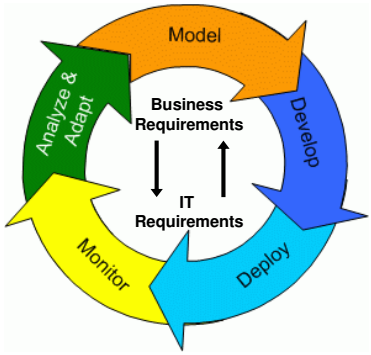

Fig. 1Main BDD phases.

(i) A business-process model that has been designed with business requirements and goals in mind is not necessarily a model that describes a scalable, reliable, and performant choreography of reusable IT services.

(ii) A service choreography derived in a top-down manner from a business-process model may not so easily or even not at all integrate with an existing IT infrastructure.

The first challenge results from a large gap between the world and perspective of a business analyst and the realities of today's programming models and software-engineering approaches. The second challenge results from the gap between an ideally designed new solution and the realities of the existing IT infrastructure involving software, hardware, and network topology. The next section will work through a small example to discuss in which form these challenges occur and how they can be addressed.

2 From Analysis Models to Design Models of Business Processes

There seems to be a widely accepted belief that *the* model of a business process exists, i.e., that there is a single model that describes the business process and that this single model is suitable for the business expert as well as the IT expert who is supposed to implement the business process. In our own work, we came to the conclusion that this assumption is very unrealistic. Instead, it is necessary (as it is common today in object-oriented programming) to distinguish between the *analysis model* of a business process and its *design model*, and to develop a methodology that enables the seamless transition from the analysis model to the design model. To illustrate the problem, let us consider the example in Figure 2, which describes a very simplified process for handling insurance claims as it may be initially depicted by a claims specialist. [2] The process consists of three subprocesses *Search Information*, *Record Claim*, and *Settle Claim*.

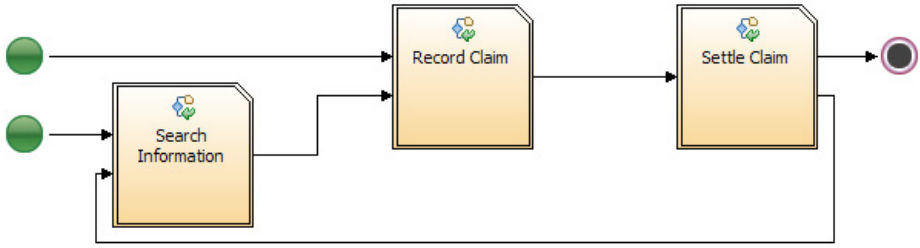

Fig. 2. Analysis model of a simplified claim handling process in insurance.

The intended meaning is described by the claims specialist as follows: "The process starts when a new claim is recorded or by revisiting an existing one to search for additional information on the claim. New information about the claim is recorded, and then the settling of the claim is attempted. The process finishes if the claim was settled, i.e., if either a benefit is payed or the claim is rejected because it is not covered by the insurance policy of the claimant, for example. If a settlement cannot be achieved, e.g., the benefit offered by the insurance is not accepted by the claimant, the process has to be resumed by searching for additional information."

The graphical model shown in Figure 2 depicts each of the subprocesses by a rectangle, explicitly denotes start and end points of the process, and shows the control-flow inside the process by connecting the modeling elements with directed edges. [3] Multiple incoming or outgoing edges denote alternative, sequential paths of execution in the process, i.e., there is no parallelism involved in this example. The decisions that select among these alternative execution paths are not explicitly represented in this model. The model is a typical *analysis model* of a business process. An analysis model describes what the process is doing. It shows the initial

[2] A realistic claim-handling process such as the reference process contained in the IBM Insurance Application Architecture [12] contains about a dozen subprocesses and more than 100 individual process activities.

[3] The figures show screenshots of the example model represented in IBM WebSphere Business Modeler, version 6.

partitioning of the process into subprocesses and activities with the main flow of control and, optionally, of data. It completely abstracts from IT-related aspects, but can be used for simulation and discussion with business analysts.

However, this model is not yet ready for implementation. First, we do not know which data will drive the process and represent the claim information. Secondly, we have no information on the underlying decision logic or business rules that guide the selection of the alternative execution paths. This information is added to the model in Figure 3, which shows a data-flow model with explicit decision and merge points. The figure shows the initial *design model* for the claim-handling process. In general, a design model contains a refined partitioning of the process that reflects existing application systems and shows an IT-based flow of data and control. It must be ready to be mapped to the desired target programming model. A fully refined design model in addition describes how the process is realized using hardware, software, and people.

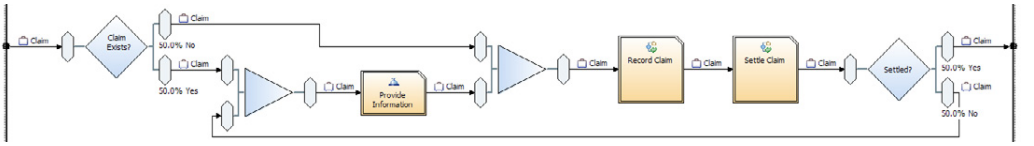

Fig. 3. Initial design model with data-flow, decision and merge points made explicit.

The transition from Figure 2 to Figure 3 is a process that consists of manual and automatic steps, involving the domain expert and an IT or Business architect who is able to work between the business and IT worlds, taking input from both sides:

- In a fully automatic step, the sequential and parallel branching and merging in a process model can be made explicit by adding explicit control actions to the process model such as *Join*, *Fork*, *Decision*, *Merge*, e.g., known from UML activity diagrams [21]. In our example, the original control-flow model is transformed into a control-action normal form, which explicitly adds control actions to the graphical model and restricts activity nodes to having only a single incoming and outgoing edge. Two decision and merge nodes are automatically added to the model, whereas the names of the decisions, *Claim Exists?* and *Settled?*, are added by the user.
- The flow of business information is made explicit in a manual top-down analysis step, e.g., it means that one annotates the input and outputs of the subprocesses with data abstractions such as *claim*, *policy*, *customer information*, and these inputs and outputs are possibly connected to model the data-flow inside the process.
- In a manual bottom-up step, data structures existing in the available IT infrastructure are revisited, which can reveal a reusable data type that can capture the required business information and drive the process. Similarly, reusable services that exist in this infrastructure can be identified to implement parts of the process. In our example, an existing Web service *Provide Information* can be reused to

implement the *Search Information* subprocess. This Web service uses a message of type *Claim* that is found suitable for representing all necessary information to drive the claim-handling process.

- In a fully automatic bottom-up step, reusable data types and services can be imported into the business-process modeling tool, where they become available as modeling elements. Service models will usually be added manually to a process model, where they are used to replace or refine existing process activities, whereas a control-flow can be refined fully automatically into a data-flow once the data type has been selected by the user. In our example, the *Search Information* subprocess is replaced by the *Provide Information* service and the *Claim* message type is assigned to the control-flow of Figure 2 to yield the corresponding data-flow model in Figure 3.

The two automatic steps in our example can be implemented as model transformations that transform the business-process model in Figure 2 to the model shown in Figure 3. First, the original control-flow model is transformed into a control-action normal form. Second, a message type *Claim* was selected by the user and then used in an automatic model transformation that changes the control-flow into a data-flow driven by *Claim*. Once the data or message type entering each decision point in the process is known, the decision conditions can be expressed by referring to the attributes and values of the data or message type. The specification of the decision condition based on the process data is a further manual refinement step of the design model.

However, we are still not done. Specific target platforms may further restrict the control- and data-flows of design models that can be mapped to code. Our simple example contains a forward edge from the *Claim Exists?* decision to the *Record Claim* activity and a backward edge from the *Settled?* decision to a merge preceding the *Provide Information* service. The backward edge leads to a cyclic process model. Unfortunately, a language such as BPEL does not support unstructured cycles, but only offers well-structured loops in the form of *while*-activities [1]. This means, another transformation must be applied that transforms models with unstructured cycles into models with loops. Figures 4 and 5 show the result of such a fully automatic cycle-removal transformation [11,15].

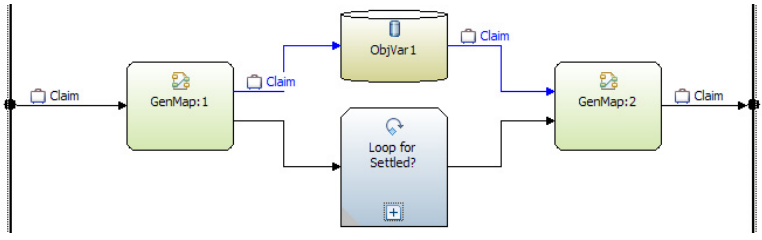

Fig. 4. Design model with loop.

Figure 4 shows a newly created loop activity that was added by the model transformation and encapsulates all those activities of the process that were reachable by the cycle. A *map* activity was added that receives the arriving *Claim* message,

puts it into a data store visualized by a repository symbol (which can also represent a variable in the target programming model, e.g., BPEL), and then triggers the loop via a control-flow edge. Once the loop has terminated, another map activity receives the modified *Claim* message and the control-flow from the loop to pass the message on to the process interface.

Figure 5 shows the loop body contents. In our example, all activities have been placed inside a loop and a *Bypass Region* decision has been added to encapsulate the *Provide Information* service, which is only invoked in some of the process executions. The access of the loop body to the data in the repository is not directly visible in the graphical visualization, it is visible only via additional textual attributes, which we do not show here because of space restrictions.

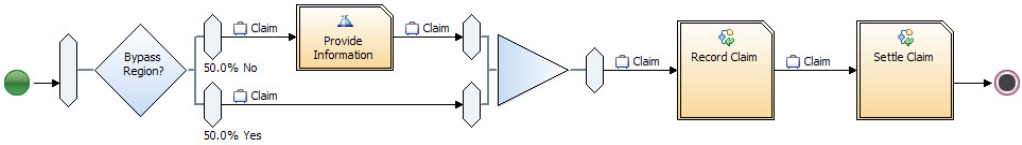

Fig. 5. The loop body containing the repeatable process activities.

Our final example design model is no longer in a representation format that would be ideal for business people. However, only from this model of the process, can now runtime code be generated automatically that meets the requirements of our intended target programming model using BPEL/WSDL. With our transformations, we have made a systematic step-by-step transition from the business view to the IT view of the process, in which each model results from its predecessor through a well-defined, gradual change. The changes were necessary to reuse existing services and data structures and address restrictions of the target programming model. One can easily imagine that scalability and performance requirements will also significantly influence how the analysis model is refined into the corresponding design model, for example when we have a choice of more than one reusable service or when quality-of-service considerations become important. The example also showed that even a simple scenario requires a sophisticated mixture between manual and automatic, bottom-up and top-down steps.

In the next section, we will review business-driven development from a larger perspective and discuss what methodological and tooling underpinnings are required to make this new style of software development successful.

3 Methodologies and Tools for Business-Driven Development

Figure 6 positions business-driven development as a software development process focusing on business processes and facing a tradeoff between the need to preserve customer investments, while moving towards a modern Service-Oriented Architecture.

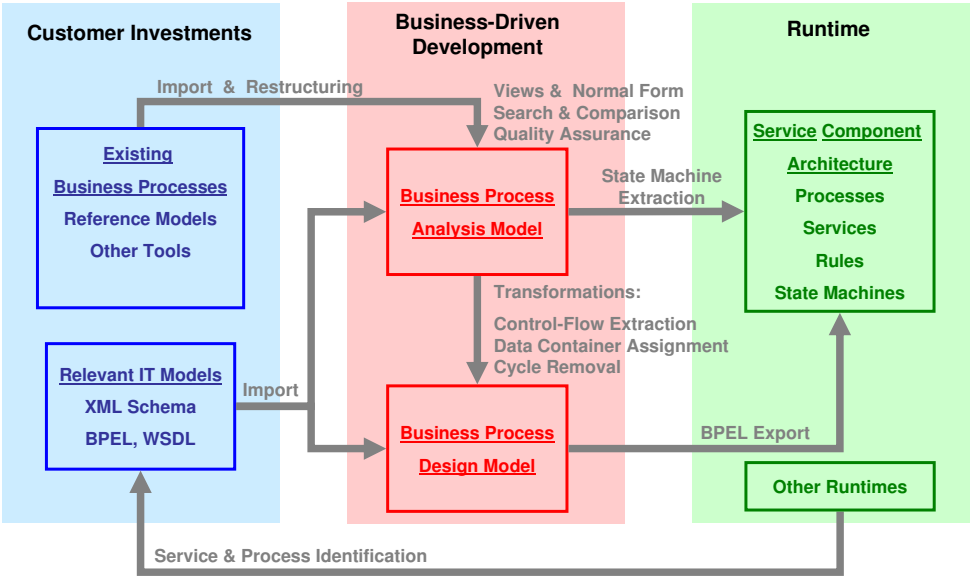

Fig. 6. Business-driven development and contributions by our research team.

In the following, we will use this figure to discuss interesting research problems and briefly review selected contributions made by our research team. The work in Zurich started in 2001 driven by the general question of how software based on Web services should be developed in the future. The work was influenced by trends such as Model Driven Architecture [8] and requirements of business integration. Very quickly, we focused on the problem of generating BPEL from business-process models [14] — an improved and extended BPEL code generation is part of IBM WebSphere Business Modeler today [16]. We realized quickly that cyclic process models are very natural for business analysts, but their transformation into BPEL is not at all straightforward. Extending our process-model-to-BPEL transformation to a larger class of process models resulted in another model transformation, called *cycle removal* [11,15]. Our team continues to focus on model transformations in the context of business-driven development, and we pursue our work today in two strands.

In one strand, we develop methodologies that underpin business-driven development and make it usable in concrete customer scenarios. At the core of these methodologies is the distinction between four types of process models: the *analysis* and *design models*, which we discussed in the previous section, the usage of *reference models* that describe best practices for an industry, and *legacy process models*, i.e., process models that many customers have built, but that have only been used indirectly as input into a software development process. The investments into these legacy models must be preserved; however, the models must also be further enhanced in their quality to serve as starting points for the generation of high-quality code and architectural solutions. This often requires their *import* into our own tools and a *restructuring* that quite often reveals semantic errors that need to be corrected.

Reference models can help in producing better To-Be analysis models and can also serve to guide the refinement of the analysis model into the design model if a reference model contains not only process models but also ready-to-use service components and data models as it is for example the case for the IBM Insurance Application Architecture [12]. The systematic usage of reference models to improve an existing business process is a challenging task when we think beyond toy examples. It requires a comparison of different types of models that can be facilitated for the human expert by using a *normal-form* representation and condensed *process views*. Specific transformations of process models such as *control-flow extraction*, which changes a data-flow model into a control-flow model, *data container assignment*, which changes a control-flow model into a data-flow model, and *cycle removal* can be fully automated. Other transformations that often require refinement and refactoring steps of the current model must be done by the user, but can nevertheless be supported by tools.

The need to interleave bottom-up and top-down steps in the development process leads us directly to the problem of roundtripping for business-process models. The theoretical boundaries for roundtripping between BPEL and an expressive business-process modeling language, for example, are easy to determine. Given the Turing equivalence of both languages, it is in general undecidable whether an arbitrary, e.g., modified, BPEL program is equivalent to a given business-process model. However, incomplete forms of reasoning about model equivalence make a lot of sense in many scenarios. For example, it makes sense to reimport a modified BPEL into a business-process modeling tool and resort to a human expert to compare it with the original business-process model from which it was generated. This can make it easier to communicate changes from the IT level back to the business and to support bottom-up steps in business-driven development that require the extraction of business-process models from the IT level.

In our methodological work, we develop detailed methodology guidelines that show how manual and automatic steps interleave, when they are done and why. A specific interest of us is the interleaving of bottom-up and top-down development steps, i.e., the problem of how business requirements flow down and how IT requirements flow up. One of the most challenging and important tasks that needs to take place at the business and the IT level is *service and process identification*—finding the right granularity of services and thinking about reuse very early in the modeling and software development process. Today, this still is a quite poorly understood step, rather than a well-understood and teachable skill.

Our methodologies also address specific questions that result from the use of different forms of models. Historically, business processes have mostly been represented by flow diagrams. Today, there is a trend to link them with business objects [20] and business state machines [3]. The semantic relationship between the various forms of models and their role in business-level and programming models are a very interesting and wide field for research.

In our second strand of research, we investigate fundamental questions that occur in our methodologies and tooling, which are often not specific to business-

process models. Many academic solutions have been developed to describe model transformations declaratively, e.g., [17,22], but the industrial reality is usually plain code. To speed up and improve the quality of our own transformation development, we developed a model transformation framework that provides a convenient API to business-process models represented in IBM WebSphere Business Modeler and a set of elementary transformations for reuse.

The testing of our transformations still is a tedious task, and creating in particular the test examples is a huge effort. The efficient generation of model instances that can be influenced by model transformations is another problem of interest to us [6]. Previous approaches either rely on exponential methods [13] or require scripts to be written [10].

Our transformations often contain many lines of code that validate the source and target models of the transformations, i.e., they check whether the model is eligible for the transformation and whether the target model that was produced satisfies a set of given design constraints. The management of these validation and design constraints at the code level creates a huge software maintenance problem, which is even aggravated by the fact that models evolve further with each new version of a software product and that transformations must consider numerous dependencies between models at the business and IT level [4].

Quality assurance for models is another fundamental question that is also related to the management of design constraints for models. There seems to be an intuitive feeling of what constitutes a good model in contrast to a bad model, but so far no really practical and user-friendly solutions to improve the quality of process models by enforcing design constraints exist. Similarly, scalable (perhaps incomplete) algorithms to detect typical design errors in process models that would lead to poor runtime code are not yet part of any business-process modeling tool.

We are also interested in algorithmic techniques that help to improve the visualization of large process models and the search capabilities of tools to find models in large collections [23]. Finally, the semantics, the definition of normal forms, and the development of algorithms and tools that allow us to compare process models with each other (be they given in the same or in different modeling languages) are also on our research agenda.

IBM's policy of basing its products on the open-source Eclipse platform makes it much easier for us to prototype our transformations and tools as plugins that extend these products, to cooperate with other research teams, and to test out our solutions in customer engagements. Nevertheless, much work remains to be done. Working with models in a tool is still a heavy-weight undertaking today. For example, many of our transformations, which physically produce a new model, can be thought of as *views*, with a view being a transformation that is not persisted [9]. However, generating such views on models, maintaining them and letting the user freely decide whether the view should be persisted as a new model or update an existing one, leads to many technical and theoretical challenges such as maintaining the consistency among several related models.

A possible vision for the future of business-driven development could be a

complete fusion of process model and code, instead of keeping physically distinct platform-independent and platform-specific models. The model is the code, in which graphical and textual elements are combined and an initial (perhaps mostly) graphical model is refined until it becomes executable. Refinement and abstraction steps will allow a user to move between different editions of a model and will be supported by quality-ensuring methods.

4 Summary

The paper presents an overview on the paradigm of business-driven development that centers around business processes and aims at a tighter alignment of the IT infrastructure with business needs and requirements. We review the main phases in a business-driven development cycle and then focus on challenges that arise from the need to transform business-process models into executable services within a Service-Oriented Architecture. We discuss a methodology that distinguishes between the analysis model of a business process and its corresponding design model and describe bottom-up and top-down model transformations that we developed to fill gaps in the business-driven development cycle. Several open or only partially solved research problems are identified and positioned within the business-driven development paradigm.

References

[1] T. Andrews et al. Business process execution language for web services. http://www.ibm.com/developerworks/webservices/library/ws-bpel, 2002.

[2] BEA Systems, IBM, IONA, Oracle, SAP AG, Siebel Systems, and Sybase. Service component architecture. http://www.ibm.com/developerworks/library/specification/ws-sca, 2005.

[3] D. Chappell. Introducing microsoft windows workflow foundation. MSDN Library, 2005.

[4] S.-K. Chen, H. Lei, M. Wahler, H. Chang, K. Bhaskaran, and J. Frank. A model driven XML transformation framework for business performance management. In *Proceedings of the IEEE Conference on e-Business Engineering*, pages 71–78. IEEE Press, 2005.

[5] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. The web services description language WSDL. http://www.ibm.com/software/solutions/webservices/resources.html, 2001.

[6] K. Ehrig, J. M. Küster, G. Taentzer, and J. Winkelmann. Generating instance models from meta models. In *8th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006)*, volume 4037 of *LNCS*, pages 156–170, 2006.

[7] T. Erl. *Service-Oriented Architecture: Concept, Technology, and Design*. Prentice Hall, 2005.

[8] D. Frankel. *Model-Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003.

[9] T. Gardner, C. Griffin, J. Koehler, and R. Hauser. A review of OMG MOF 2.0 Query / Views / Transformations submissions and recommendations towards the final standard. MetaModelling for MDA Workshop, York, England. Also available as an OMG position paper at http://www.omg.org/docs/ad/03-08-02.pdf, 2003.

[10] M. Gogolla, J. Bohling, and M. Richters. Validating UML and OCL models in USE by automatic snapshot generation. *Software and Systems Modeling*, 4(4):386–398, 2005.

[11] R. Hauser and J. Koehler. Compiling process graphs into executable code. In *Proceedings of the 3rd International Conference on Generative Programming and Component Engineering (GPCE-04)*, volume 3286 of *Lecture Notes in Services and Software*, pages 317–336. Springer, 2004.

[12] IBM Insurance Application Architecture. http://www.ibm.com/industries/financialservices/iaa.

[13] D. Jackson et al. The Alloy modelling language and analyzer. http://alloy.mit.edu.

[14] J. Koehler, R. Hauser, S. Kapoor, F. Wu, and S. Kumaran. A model-driven transformation method. In *Proceedings of the 7th Conference on Enterprise Distributed Object Computing (EDOC-03)*, pages 186–197. IEEE Press, 2003.

[15] J. Koehler, R. Hauser, S. Sendall, and M. Wahler. Declarative techniques for model-driven business process integration. *IBM Systems Journal*, 44(1):47–65, 2005.

[16] R. Kong. Transform WebSphere Business Integration Modeler process models to BPEL. IBM developerWorks article, http://www.ibm.com/developerworks/websphere/library/techarticles/0504_kong/0504_kong.html, IBM, 2005.

[17] J. M. Küster. Definition and validation of model transformations. *Software and Systems Modeling*, 5(3), 2006.

[18] R. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2002.

[19] T. Mitra. Business-driven development. IBM developerWorks article, http://www.ibm.com/developerworks/webservices/library/ws-bdd, IBM, 2005.

[20] P. Nandi and S. Kumaran. Adaptive business objects: A new component model for business integration. In *Proceedings of the 7th International Conference on Enterprise Information Systems*, pages 179–188, 2005.

[21] Object Management Group (OMG). *The Unified Modeling Language 2.0*, 2005.

[22] K. Ryndina and J. M. Küster. A Layered Approach to Defining a Transformation Language - Informal Description and Validation by Case Study. Technical report, IBM Research, Research Report RZ 3547, February 2005.

[23] B. Srivastava, J. Vanhatalo, and J. Koehler. Managing the life cycle of plans. In *Proceedings of the 17th Innovative Applications of Artificial Intelligence Conference*, pages 1569–1575. AAAI Press, 2005.

[24] M. Völter and T. Stahl. *Model-Driven Software Development : Technology, Engineering, Management*. Wiley, 2006.

[25] O. Zimmermann, M. Tomlinson, and S. Peuser. *Perspectives on Web Services - Applying SOAP, WSDL and UDDI to real-world projects*. Springer, 2003.