Improving Application Integration with Model-Driven Engineering

Clemens Dorda Uwe Heinkel Bernhard Mitschang
Institute of Parallel and Distributed Systems (IPVS), University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
{Firstname.Lastname}@ipvs.uni-stuttgart.de

ABSTRACT
Modern software for Enterprise Application Integration (EAI) provide tools for modeling integration scenarios. A drawback of these tools is the missing functionality to exchange or integrate models of different EAI products. Consequently, developers can describe real heterogeneous IT environments only partially. Our goal is to avoid the creation of these so-called 'integration islands'. For that purpose we present an approach which introduces an abstract view by technology-independent and multivendor-capable modeling for both development and maintenance. With this approach, we propose a toolset- and repository-based refinement of the abstract view to automate the implementation with real products and the deployment on real platforms.

Keywords
Enterprise Application Integration, Model-Driven Engineering, Software Lifecycle.

1. INTRODUCTION
The history of integration projects in enterprises is almost as long as the usage of information systems itself. People often refer to the advantages and positive effects of application and system integration, e.g. for faster and more automated business process execution, but they very often disregard the risks for the IT infrastructure. For example, a drawback of integration projects is the endangering for the agility of the IT infrastructure, because they link together previously autonomous application systems. Consequently, the overall system agility gets considerably reduced and subsequent changes are difficult to adapt.
This ability decreases more and more with every new integration project. During an integration project, developers build large distributed systems. In the following sections, we call these large distributed systems integration landscape. In such an integration landscape, we have dependencies between existing stand-alone applications. For example, if an online shop accesses the current warehouse stock, then it is impossible to change the schema of the warehouse stock without considering the online shop programming logic. The poorer the documentation, the greater the problem, since knowledge of the integrated applications is usually distributed over many people or organizations inside of an enterprise.
These facts result in a situation where central applications cannot be enhanced, because the risk becomes too big for an unintentional negative impact on other critical systems. Therefore, with an increasing grade of integration, it becomes more and more difficult, complex, and expensive to adapt the system landscape to changing needs. The agility gets lost.
In this paper, we present methods to maintain agility over the whole lifecycle of an integration landscape. We want to achieve our goals by introducing both an approach, which describes a process model for integration projects, and an appropriate generic tool support for development and documentation. The goal of this approach is to improve especially documentation aspects compared to the state-of-the-art of available integration solutions. However, our approach does not address semantic integration problems, for example semantic data heterogeneity. This topic is very complex on its own, and there are other researchers who address this problem. A good introduction is given in [1]. Further experiences and challenges with Enterprise Information Integration (EII) are subsumed in [2].
This paper is structured as follows: in Section 2, we discuss main problems of integration projects and introduce typical integration scenarios. After that, we give an overview about related work in Section 3. Section 4 introduces our RADES (Reference Architecture for the Development and Support of EAI Solutions) approach and describes its concepts as well as its tool support. In Section 5, we give a short conclusion and outlook on future work.

2. PROBLEM STATEMENT
To motivate our RADES approach, we have a closer look at typical integration problems and scenarios. In the first subsection, we identify three main problems of integration projects. These problems are difficult to handle with current tools. Based on these problems, we derive and discuss the most important scenarios of application and system integration in the second subsection. From our point of view, tool support for application and system integration should focus on these scenarios.

2.1. The Main Problems of Integration Projects
In general, integration projects suffer from three inherent problems: (1) divergence between system documentation and the current state of system runtime, (2) heterogeneity, and (3) distribution of systems and organizations.
The first problem we discuss is the *divergence between system documentation and system runtime*. During an information system's life cycle, there are for example changes to interfaces or message formats from time to time. Unfortunately, many people forget to update the related documentation. The consequence is an inconsistent system documentation, and as a result of that an often unknown system state of the participating application systems.
A problem that comes along with inconsistent documentation is informal documentation, which often leads to varying interpretations by different readers. The consequence is an extensive reengineering of the system's runtime environment.

The second problem, *heterogeneity*, intensifies the first problem. Due to the fact that system landscapes become more and more heterogeneous, people have to deal with different integration products from different vendors. Consequently, they also have to deal with different philosophies of tool support and meta data management. This makes it more and more difficult to get a consistent view over the whole integration landscape. However, heterogeneity is not only limited to systems and applications – heterogeneity also applies to people. Integration projects are often done with different partners, who use different processes, documentation methods, or notations. Consequentially, heterogeneity is a problem for all integration project artifacts.

The *distribution of IT systems and organizations* is the third problem. Looking at distributed systems and applications, it is obvious that people have to put big effort to do consistent changes, because they have to be done on many places. These changes are error-prone, because developers mostly have to perform them manually. Looking at organizations, distribution of responsibilities over different people and departments leads to a troublesome retrieval of required information.

People usually try to solve these three problems by keeping them technologically and organizationally simple. The technical solution consists of a loose coupling of both application and resource systems through the integration product. Assuming a suitable architecture, developers can do changes in application systems transparent behind the affected interfaces by changing their implementation. The organizational solution consists of a strict project management and documentation guideline.

These solutions are good and very important, and they are the precondition for our approach. Nevertheless, it is necessary to introduce additional concepts, because these solutions are not sufficient to solve all of the problems mentioned above.

First of all, organizations need a central point for application and system documentation. Usually, this documentation point will be realized with a document management system. We call this a central documentation system in the succeeding sections. The documentation stored in this central system should allow a consistent and homogeneous up-to-date view onto the runtime environment of integration solutions. Second, a defined process model has to guarantee the consistency between documentation and runtime, supported by appropriate tools. Finally, this process model must fit seamlessly into existing engineering processes.

2.2. Scenarios for Application Integration

To define an approach which addresses the requirements mentioned in the last section, it is necessary to look at some typical scenarios in integration projects. We are convinced that an appropriate process model must support these scenarios.

The first scenario is the *management of documentation*. This scenario is outstanding, because it affects the other ones. We already mentioned the need for a central documentation system, but more difficult is the problem how to create good and sufficient documents. This leads to the question how we can measure the quality of documentation in general. However, we do not discuss this topic in our paper. For our needs, a good solution is to find a way to get as much documentation as possible from our meta data. Together with human-made documentation, we need an automated way to collect this documentation and store it into the central documentation system.

A *new development of an integration solution* is the second scenario. Usually, developers start working from a preliminary design and refine it step by step towards a runtime environment. From our point of view it is important that developers store their in-between work into the central documentation system mentioned above. Another important requirement is the capability to check the results of a refinement step for consistency with the results from the step before. This requires a precise definition of the development approach as well as a precise definition of the result notation. After the last development step it should be possible to generate all required configuration data or code for the integration platform. For that purpose, developers should check out the meta data from the central documentation system.

Our next scenario, *management of changes and maintenance*, is quite similar to the second scenario. This scenario requires the support of two different alternatives, changes on meta data or changes in code. The first one is similar to the new-development scenario and includes the documentation of changes into the repository, which leads to several refinement steps until generation of program code or configuration data. The second one is a scenario which is more difficult to handle. In this scenario, developers do their changes within the code or configuration data of the integration product, e.g. to fix a problem quickly. Here, mechanisms must be available, that detect these changes and propagate them to the central documentation system. After that, consistency checks must follow and must implicate well-defined steps to propagate these changes backwards to the higher development levels. Subsequently, the central documentation system should guarantee that all documentation is consistent again.

A very common scenario is our last scenario, the *migration of integration products*. From time to time, enhancements to integration technology require the migration of an integration product to another version, or to another product from another vendor. We mentioned in Section 2.1 the problem of different meta data, which results often in a complete reimplementation.

With the existence of a consistent central documentation as we pointed it out, it is possible to use the content as an interchange format between integration products. Together with applicable generators and adapters for the target product, which are required anyway for the scenarios before, this information shortens the implementation for the migration project.

3. RELATED WORK

In this section, we discuss commercial approaches and scientific work, which partially solve some of the problems we identified in Section 2. This discussion shows that even though there are some good approaches to address the problems of application integration, they can not solve these problems all-embracing.

3.1. Commercial Approaches

Software producers of commercial EAI products have integrated various mechanisms into their bundled development tools, which discard the common bottom-up approach for application integration. Instead, they provided tools which allow or force developers to use a top-down approach for development. This solves some of the problems mentioned in Section 2, but some requirements are not implemented yet.

One of these requirements is the lack of common standards, which makes it difficult to replace one EAI product by another product from a different vendor. This drawback reduces the agility especially of large enterprises, because different requirements for integration projects implicate the usage of different products, as well as the evolution of IT systems implicate heterogeneity. Finally, this means for development, documentation and maintenance of integration solutions, that the incompatibility among EAI products anticipates the

implementation of common practices for integration development, based on the packaged tools. Consequently, existing technology cannot solve satisfactory the problems mentioned in Section 1.

3.2. Scientific Work

There are some research projects which try to address the figured problems from different point of views.

The Model Integrated Computing (MIC) approach [3] describes a method how developers can model all relevant information about a system in development with domain specific modeling. The approach describes possibilities how to derive new models from existing models, and how to generate code from models with generators. Related to our needs, one drawback of this approach is both the lack of a well-defined domain model for EAI and a standardized modeling language.

Nevertheless, the idea behind this approach is one of the fundamentals of Model Driven Architecture (MDA) [4]. MDA is a standard from the Object Management Group (OMG), and is in contrast to the MIC approach more specific concerning the modeling language. The Unified Modeling Language (UML) [5] is the standard modeling language for MDA, but this does not eliminate other languages for MDA. Methodically, MDA specifies a top-down process model with three steps. The result of each step is a set of models in a specific abstraction. MDA distinguishes between platform independent models (PIM's), platform specific models (PSM's) and code. At present, many vendors support MDA with their tools, or plan to support MDA in the near future. Follow-up OMG standards of MDA are for example the Query/View/Transformation (QVT) specification [6], which is important to describe model transformations between the MDA abstraction layers in a unique way. A second example is the new Architecture-Driven Modernization (ADM) approach [7], which is driven by nearly the same motivation as our RADES approach.

In the beginning of our research we thought that MDA is exactly the standard we need for a clean software engineering process for system integration. After doing some research, it became clear that we need an additional abstraction layer to meet all our requirements. This fits seamless into the Model Driven Engineering (MDE) approach, which was outlined by Stuart Kent, as stated in [8]. We can say that MDE is a superset of MDA. Consequently, in comparing the concepts behind these two terms, MDA can be seen as an instantiation of MDE. For example, as MDA postulates three abstraction layers (PIM, PSM, Code), MDE postulates in a more common manner multiple abstraction layers. In MDE, each abstraction layer is generally spoken a refinement of its predecessor, which is a layer with higher abstraction [9].

Both, MDA and MDE, have one big problem. The methods how to model a good MDA or MDE model in a specific abstraction layer are not precisely specified. There are scientists who try to give answers for that problem by providing a MDE megamodel [10], which is a model of MDE concepts that is meant to be a starting point for a sound and complete MDE theory. For our approach, we restricted the application domain on system integration, similar to the MIC approach [3]. With this restriction, we think that we are able to develop a sufficient precise modeling specification. Similar to [11], we use libraries in each abstraction layer to avoid that developers must build all model components from scratch, and to make succeeding model transformations more powerful.

There are other approaches that focus on the topic of EAI and MDA. In [12] for example, the authors present a five model approach to realize a Model-Driven Architecture for EAI. Each model represents a different aspect of the EAI problem. Nonetheless, it lacks of a clear definition how the models relate to each other, as well as of a prototypical implementation.

In the next section, we give answers why we think that our four-layer modeling process is necessary for developing solutions for system integration, which meets our requirements in Section 2.

4. THE RADES APPROACH

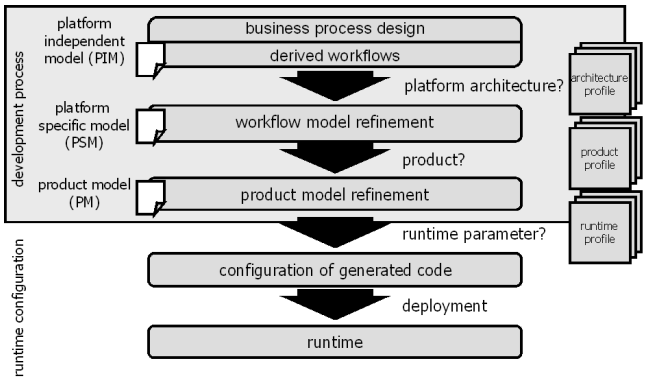

Figure 1: RADES Process Model

RADES (Reference Architecture for the Development and Support of EAI Solutions) is a research project founded in cooperation with DaimlerChrysler Research & Technology. The project's goal is the development of a reference architecture to implement, document and maintain integration scenarios. The reference architecture includes concepts for both development methods and tools.

One part of this approach is the usage of a central repository, which contains all necessary information about applications and systems being part of an integration landscape. A well-defined development process is supposed to guarantee the consistency of all development data, including documentation, models, source code, and binaries (see Figure 1). Each stage in the whole lifecycle postulates on the one hand results in a specific syntax and semantic, but allows developers on the other hand a flexible organization of the tasks inside of each lifecycle stage.

4.1. RADES Development Stages

Similar to MDA, we first defined three development stages for the RADES development process (PIM, PSM and Code). During our work, we decided to extend the development process to four stages by introduction the concept of a product model (PM). In the next sections, we discuss the reason for this decision.

4.1.1. Business process and workflow modeling

One task, which is often disregarded, is the design of the business process, that defines the underlying workflow of an integration scenario between applications and systems. This task is, from our point of view, one of the most important ones to finish an integration project successfully. After mapping the business processes to one or more workflows, developers are enabled to make a decision about the integration strategy. They are able to decide which integration technology is appropriate for interconnections between previously not connected applications or systems. In some cases, they may desire to extend an existing interconnection between systems with the so far used EAI technology. In other cases, they may decide to replace such an existing integration technology with another one.

Therefore, the RADES approach defines the modeling of platform-independent workflows in the first development stage. Developers have to model applications, systems and resources, as well as their logical dependencies to each other. In this stage, it is sufficient to model only components whose functions and data are required for workflow execution from the application perspective. The collection of these models is called the platform-independent model (PIM), and it reflects an overview over the integration landscape.

4.1.2. Detailing workflows

In the next development stage, developers have to refine the platform-independent model to a platform-specific model. A platform-specific model is a description of the integrated systems within the targeted integration architecture without determining on a specific integration middleware technology or product. An example for a possible integration architecture is a "hub and spoke"-architecture, which is most commonly used, or a bus architecture . By choosing an integration architecture, developers limit on the one hand possible integration products to a certain product family, but does not commit to a specific product on the other hand.

To get an initial platform-specific model, we use model transformations. The generator software, which is used for transformations, gets its input from the PIM and from a specific transformation profile. The transformation profile describes the integration architecture of the targeted integration middleware product. After the transformation, developers detail the models to get a complete product-independent description of the integration landscape. For example, to get a precise description of activities during workflow execution, developers have to detail the sequence and activity diagrams from the PIM.

4.1.3. Detailing the product model

The third development step starts similar to the second step. First of all, developers have to transform the PSM to the product model (PM). As before, the input for the transformation engine are both the PSM models and a transformation profile describing the targeted integration product, for example IBM WebSphere Business Integration Message Broker. Subsequently, developers have to refine the generated PM so that the PM is well prepared for the code generation engine.

4.1.4. Configuration and runtime

The last step in our development process is code generation for the targeted integration platform. For that purpose, developers need an appropriate code generation engine, which is enabled to compile the PM to executable configuration data or runtime code for the targeted integration product. Due to the fact that the PM is a very detailed description of the integration product's metadata, the code generation is a simple mapping task in general. However, the reason for the necessity of the PM development stage before is simple. The PM describes all necessary interfaces, messages and flows of a specific integration product very detailed – but it does not describe a concrete configuration of this product. Therefore, this must be done by code configuration in this development stage.

4.1.5. Deployment

This development stage is not part of the RADES development approach. Usually, developers use tools for deployment which are bundled with the integration product. Nevertheless, it is possible that deployment fails. In this situation, it is necessary to go one or more development steps backwards to solve the problem on a higher abstraction layer. After that the succeeding development steps have to be done again.

4.2. Modeling

Modeling techniques become very important in the context of model-driven engineering. To get a better understanding of how we address this issue, we explain our modeling technique by giving an example. This example shows a very simple PIM model, which we transform to a PSM model by applying an appropriate architecture profile.

As a basic principle, the RADES approach does not commit to a certain modeling language or modeling tool. However, it is required that all models can be transformed into a unified model interchange language to process the models with different tools. For our research and prototype development, we use UML for modeling, and the XML Metadata Interchange (XMI) format [13] for model interchange.

For RADES, we defined a modeling syntax and semantic subset based on the "UML for Enterprise Application Integration" profile from the OMG [14]. The advantage of this definition is the strong semantic explanatory power. This subset provides two typical modeling approaches for UML, collaboration modeling and activity modeling.

Collaboration modeling expresses how applications and systems interact through the exchange of messages. An example is the modeling of dependencies between application and systems, or the modeling of message flows. Developers model collaborations with class diagrams and collaboration or sequence diagrams.

Activity modeling expresses business processes between applications and systems on each abstraction layer. This can be done by modeling control and message flows in consideration of execution order and temporal aspects.

Although the notation is still work in progress, we give a simple example for collaboration modeling in the PIM and PSM layer.

The class diagram modeled in Figure 2 expresses that Application_1 is dependent on Application_2. The corresponding collaboration diagram in Figure 3 expresses that Application_1 sends messages to Application_2. The semantic of the arrow used here means that the communication between Application_1 and Application_2 is synchronous.

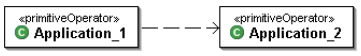

Figure 2: Simple PIM class diagram that depicts a dependency of Application_1 on Application_2

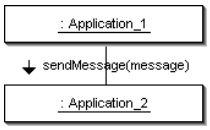

Figure 3: PIM Collaboration diagram depicting that Application_1 can send messages to Application_2

While there is no further modeling necessary in this example to get a rudimentary PSM collaboration model, it is required to make a decision about the integration architecture. In this example, we decide to choose a hub-and-spoke architecture.

As we describe later in Section 4.3, we want to transform the PIM model from Figure 2 and Figure 3 to a rudimentary PSM model. Therefore, it is necessary to create a transformation profile, which contains the necessary information of how a hub-and-spoke architecture is organized. The class diagram of such a profile is shown exemplified in Figure 4. For our example, we need the components marked with the numbers ❶ (Call-Adapter), ❷ (Router), ❸ (Request/Reply-Adapter) and ❹ (EAIBroker,

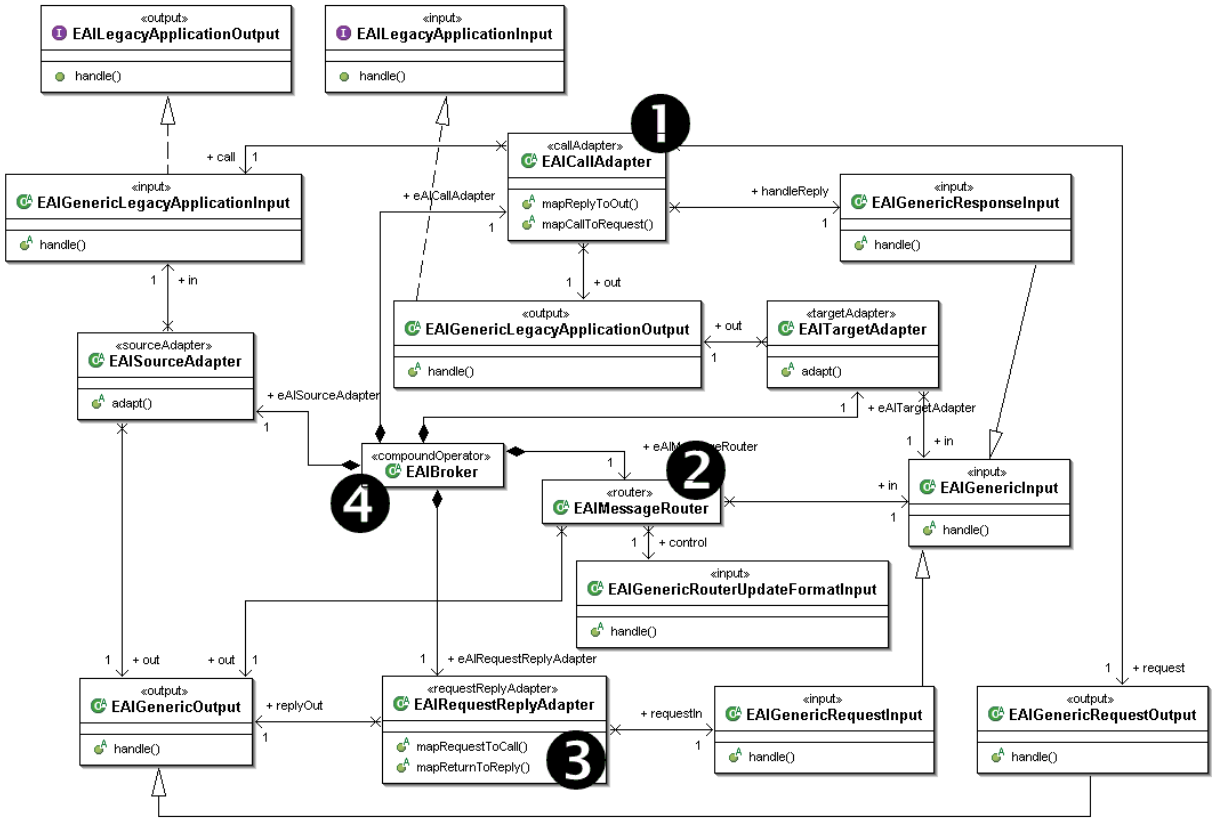

Figure 4: Hub-And-Spoke transformation profile for generating a PSM model

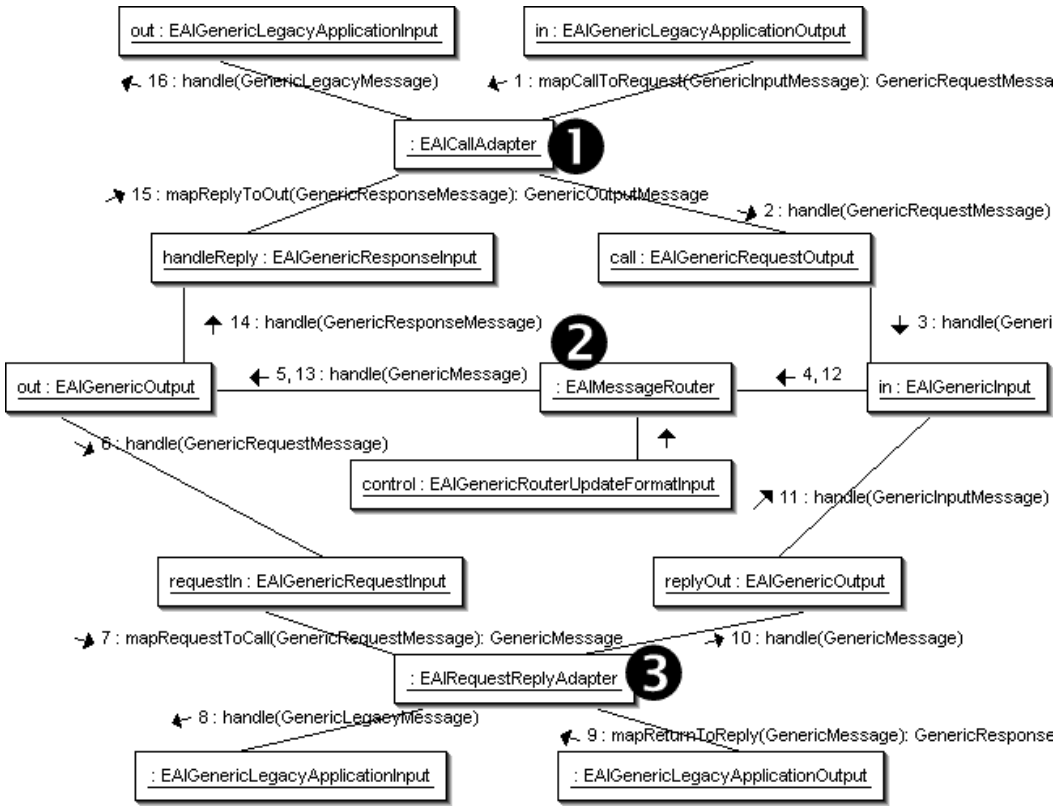

Figure 5: Collaboration Diagram of Figure 4

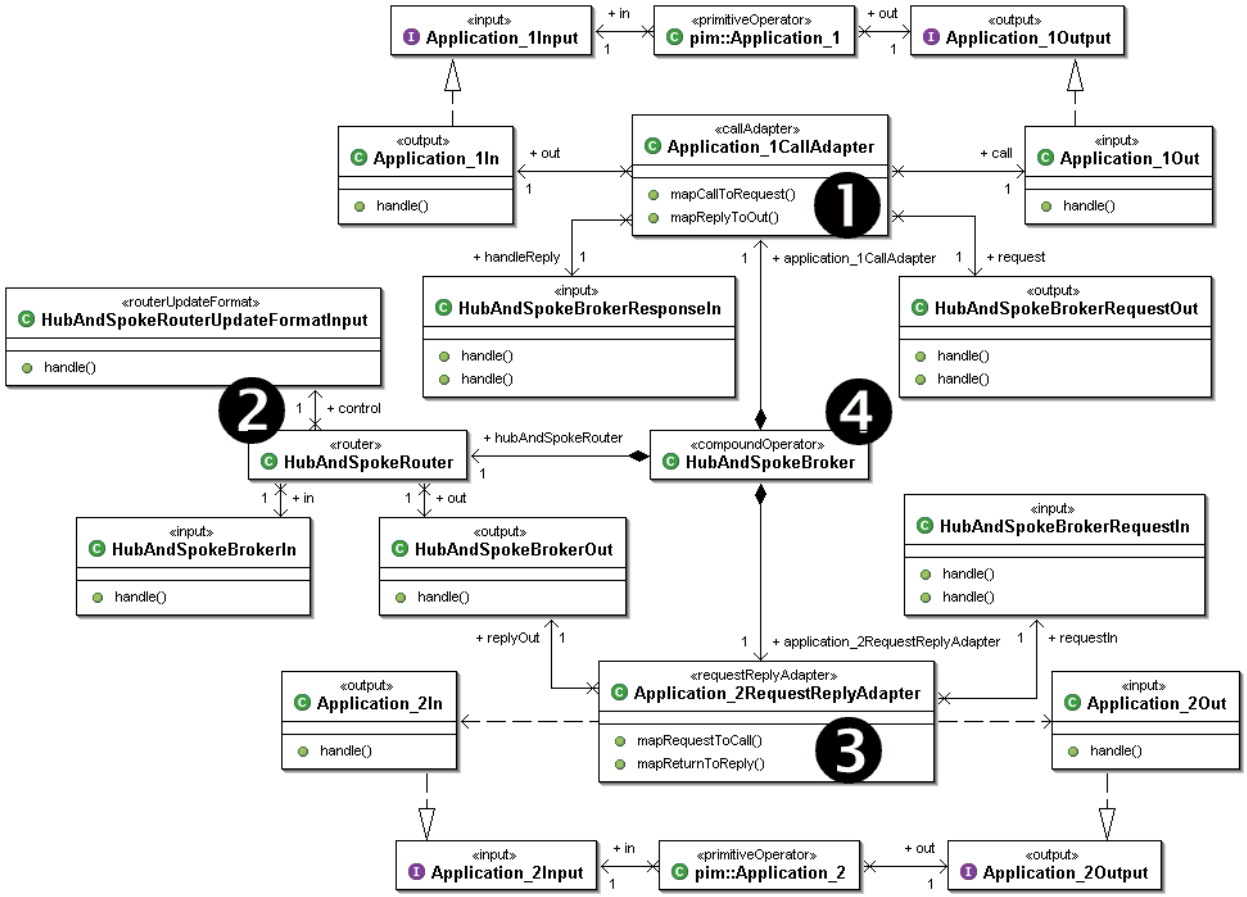

Figure 6: Class diagram (PSM) of Figure 2 and Figure 3

realized as CompoundOperator), together with their associated input and output terminals. See [14], Chapter 8 for a detailed description of the components.

Figure 5 depicts in which way these components are connected to each other, and in which order messages can be sent to the input and output terminals. Note that the EAIBroker (❹) from Figure 4 is not part of the collaboration diagram, because it is modeled as a composition of the components ❶, ❷ and ❸.

The model transformation engine analyzes the PIM model (Figure 2 and 3) and the generation profile (Figure 4 and 5), and after that it transforms the PIM to a rudimentary PSM model (Figure 6). Note that the numbered components in Figure 6 correspond to the same components in Figure 4 and 5.

The collaboration diagram of Figure 6 is similar to Figure 5, therefore we do not illustrate it here.

4.3. Transformation

Model transformation is one of the key features for MDE. In the last years, many people tried to find appropriate mechanisms to describe and execute model transformation. One of the results is the ATLAS transformation language (ATL) [15] as an example for a textual transformation language with descriptive and imperative language elements, as well as the UMLX transformation language [16] as an example for a graphical transformation language. Both approaches incorporated into the QVT specification of the OMG, which defines both a textual and a graphical notation for model transformation.

Since the first adopted specification of QVT became available not until November 2005, we developed and implemented a slightly different transformation mechanism in our prototype.

For our future work, we plan to integrate some of the existing or upcoming QVT implementations in this prototype. Nevertheless, in our actual work we are still using our notation and implementation, because neither ATL nor UMLX are appropriate to meet our needs with respect to the current available implementations.

To transform a model for further manual refinements (see Section 4.1) from one abstraction layer to another, we need information about the next abstraction layer. We create this information by modeling transformation profiles. These transformation profiles can not be compared with transformation rules as known from QVT. Our transformation profiles are more similar with patterns, from which the transformation engine derives the transformation rule by applying pattern matching and heuristic algorithms on the source model and transformation profile model.

Each transformation requires such a transformation profile with a specific notation. The transformation engine reads both the source model and the transformation profile to transform the source model into a target model. This target model is the foundation for further refinements done by a developer, and after that input for a subsequent transformation. Figure 7 shows a schematic example for these consecutive transformation and refinement steps. For a better readability, we both did not illustrate the manual refinements here and did not draw the models in UML-notation.

In contrast to our example given in section 4.2, this example illustrates the scenario shown in Figure 2 and Figure 3 with

asynchronous communication, as well as the entire development process from PIM to code generation. Step ① depicts a platform independent model, which models a communication between application A and B. The application A always initiates communications with B and communicates in an asynchronous manner. Step ② is a description of the integration architecture, which shall be used to implement the communication channel between the applications A and B. Step ③ is a transformation process: the model transformation engine transforms the model from ① in consideration of the transformation profile in ② to a new platform specific model (PSM), as depicted in step ④. If necessary, a developer completes this model for further transformation. After that, the model transformation engine transforms the model (⑥) from ④ by means of the product transformation profile in ⑤ to a product model ⑦. If necessary, a developer completes this model to a comprehensive model-based description of the integration landscape. In step ⑨, the model transformation engine transforms the model from ⑦ to product-specific configuration or code, according to the runtime profile ⑧ of the targeted integration product.

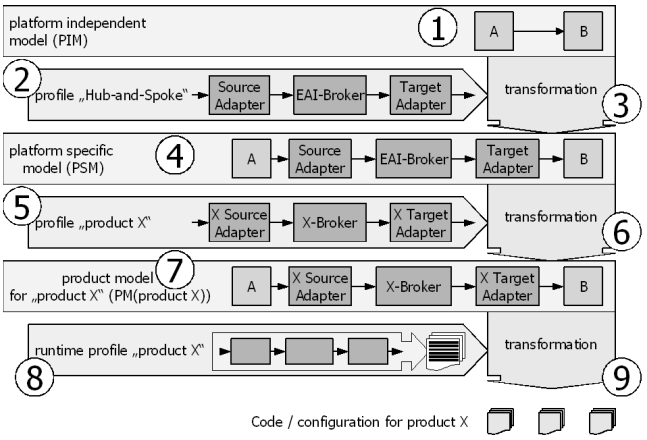

Figure 7: Schematic example illustrating model transformations with RADES

4.4. Prototype implementation

Our current work focuses on the implementation of a prototype for model transformations, based on the Eclipse framework [17]. With this prototype, we want to provide a proof of concept for both our process model as well as our modeling approach.

We have chosen Eclipse for several reasons. First of all, it is relative simple to write plug-in modules which extend the Eclipse framework. Second, with the EMF framework [18] and the EMF-based UML2 implementation [19], the Eclipse project provides a solid infrastructure for model processing. Our transformation plug-in modules, which will implement the transformation from PIM to PSM, PSM to PM, and PM to Code, build on top of these libraries.

The principle of a transformation module implementing our methodology from 4.3 is quite simple. First of all, the plug-in gets an object-oriented representation of the UML models stored in XMI files by calling EMF and UML2 functions and builds internal graphs.

After the plug-in has built the graphs for the UML transformation profile and the input models, it uses filter classes to analyze the graphs, and applies transformation rules specified in builder classes. These filter and builder classes must be provided together with the chosen transformation profile. The filter classes search for patterns specified in the filter rules. The builder classes replace associations and generalizations in the input models through components given in the transformation profile. After transformation, the plug-in writes the packages to an XMI file, by using the EMF and UML2 interface.

Currently, we successfully implemented a plug-in module for the transformation of PIM to PSM models. For testing purposes, we used a rudimentary Hub-and-Spoke UML profile. The implementation of a transformation profile, which enables the transformation from PSM to PM models, using a transformation profile for IBM WebSphere MQ Integrator, is ongoing. Our prototype will be the platform for the implementation of further research results, for example algorithms implementing reverse engineering approaches or concepts for the generation of human readable documentation.

4.5. Evaluation

We discuss in this section in which way the RADES approach supports the scenarios introduced in Section 2.2.

The first scenario we described was the management of documentation. RADES supports it in two ways. First, RADES uses a repository for the centralized storage of models and additional documentation. In some cases, such a repository already exists for some documents, so working with RADES will complement the existing documentation. In other cases, such a repository must be set up first. Second, the usage of a unified model interchange language enables a better tool-based metadata preparation for documentation without limiting the choice of usable modeling tools.

Our second scenario, the new development of an integration solution, is supported by the RADES process model. This process model defines a thread throughout the development process, and offers various views over the integration solution by providing different abstraction layers.

These abstraction layers provide starting points for the third and fourth scenario, depending on whether changes should be reflected to workflows, architecture, or the product model.

It is well known that the collection of meta data and the ensuring of data quality is a very time intensive and difficult task. Therefore, it is obvious that the reusability of existing meta data is a feature which on the on hand shortens the development time, and on the other hand improves data quality significant. According to the RADES approach, it is a logical consequence that the more developers have modeled integration scenarios with the RADES approach, the greater is the benefit of reusing models from application systems involved in already existing integration landscapes. Although this is not the focus in this paper, it is obvious that the existence of a powerful repository for the storage of all relevant documents and data is highly important. On the one hand an appropriate repository has to offer several options to link documents and data together, whether they are of the same content type or not. One the other hand, it is important to have efficient access to the repository's content, for example by providing different views and a powerful search engine. And finally, regarding large organizations, scalability, consistency, and availability must be guaranteed.

The introduction of the additional abstraction layer, the product model, is not a radical change to the MDA philosophy. In contrast, by applying MDA to practice, there are problem domains – like Enterprise Application Integration – which require a more sophisticated view onto the problem solution. This is one of the reasons why people introduced MDE, and thus one of the reasons for us to introduce this additional abstraction layer in

RADES. In the EAI context, we are convinced that a clean MDA approach is more difficult to apply than our RADES approach.

Although first results from our research prototype demonstrate that model-driven development can strictly be applied to EAI, there are still some research topics which are unsolved to date.

First of all, it is not a trivial task to define clear rules for the decision if a model is complete and valid regarding the abstraction layer being modeled. For example, there are many degrees of freedom for modeling a platform independent model for EAI. Therefore, we want to provide a robust definition covering these degrees. We are convinced that the experiences from our prototype help us to formulate such a definition.

Another problem which is not yet satisfactorily solved is reverse engineering. Developers do not want to develop strictly top-down. There are situations where developers prefer to proceed bottom-up, for example to solve a critical problem quickly. A model-driven development approach like RADES has to consider this by providing consistency checks, which have to deliver information whether changes on a lower abstraction level affects the models on a higher abstraction level or not. An eligible goal is something like a "back transformation", which means that for example a PM can be transformed backwards to a PSM, if changes to the PM affect the PSM model. According to our experiences with reverse engineering tools, we think that current tools are still limited and do not cover these demands completely.

A third research area is the role of human readable documentation. According to experience, UML models are not self-explanatory in many cases. It is therefore necessary that people write additional human-readable documentation to explain the models they developed. Without mechanisms, which guarantee that the human readable documentation is still consistent with the model it describes, the human-readable documentation may become worthless after performing changes to the related UML model. Regarding the RADES approach, we would like to have a solution enabling a documentation technique, which keeps human-readable documentation synchronous with the related UML model. The Human-Usable Textual Notation (HUTN) standard [20] may help here, but we did not evaluate it so far to confirm this.

5. CONCLUSION

Agility and flexibility is an important factor for enterprise application integration. This factor is underestimated and undervalued by many experts. The ongoing evolution of new IT concepts and technologies and their usage in enterprises leads to instable configurations of system landscapes. Developers can handle these instable configurations pretty good if they choose not to change the integration product. But if they choose to replace an integration product with a system from a different software producer, they can not just reuse the meta-information implemented in the existing systems.

The RADES approach offers a way to eliminate this drawback. We avoid an early commitment on specific integration products through high-level modeling in the beginning, and provide precise, reusable models of the integration landscape through several refinement steps. It is obvious that our approach does not make the initial implementation of an integration scenario less complex.

However, the return of investment comes with reuse, maintenance and request for standardized documentation. This is the high benefit of RADES. Practical experiences in our daily work make these benefits very important, and we think that they are worth to invest some more efforts in the first implementation.

Further work will focus especially on model transformation techniques and reverse engineering problems, because this is an essential feature for developers to adopt approaches like RADES.

6. REFERENCES

[1] Doan, AnHai; Noy, Natalya F.; Halevy, Alon Y.. *Introduction to the special issue on semantic integration*, ACM SIGMOD RECORD, ACM Press, vol. 33 (4), 2004.

[2] Halevy, Alon Y. et al., *Enterprise information integration: successes, challenges and controversies*, ACM SIGMOD: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 2005.

[3] Sztipanovits, Janos; Karsai, Gabor. *Model Integrated Computing*, Computer, IEEE, vol. 30 (4), 1997.

[4] Soley, Richard et al.. *Model Driven Architecture*, 2000.

[5] OMG. *Unified Modeling Language*. www.uml.org.

[6] OMG. *MOF QVT final adopted specification*, 2005.

[7] OMG. *Architecture-Driven Modernization (ADM) Task Force*. http://adm.omg.org/.

[8] Alanen, Marcus; Lilius, Johan; Porres, Ivan; Truscan, Dragos. *Model Driven Engineering: A Position Paper*, MOMPES 2004.

[9] Fondement, Frédéric; Silaghi, Raul. *Defining Model Driven Engineering Processes*, WiSME 2004.

[10] Favre, Jean-Marie. *Towards a Basic Theory to Model Model Driven Engineering*, WiSME 2004.

[11] Witthawaskul, Weerasak; Johnson, Ralph. *An Object Oriented Model Transformer Framework based on Stereotypes*, WiSME 2004.

[12] Al Mosawi, Adra; Zhao, Liping; Macaulay, Linda, *A Model Driven Architecture for Enterprise Application Integration*, HICSS: Proceedings of the 39th Hawaii International Conference on System Sciences, 2006.

[13] OMG. *MOF 2.0/XMI Mapping Specification, v2.1*, 2005.

[14] OMG. *UML Profile and Interchange Models for Enterprise Application Integration (EAI) Specification, v1.0*, 2004.

[15] Jouault, Frédéric; Kurtev, Ivan, *Transforming Models with ATL*, MoDELS 2005: Proceedings of the Model Transformations in Practice Workshop, 2005.

[16] Willink, E.D.. *UMLX: A graphical transformation language for MDA*, GMT Consortium 2003.

[17] Eclipse Foundation. *Eclipse Open-Source Community*. http://www.eclipse.org/.

[18] Eclipse Foundation. *Eclipse Modeling Framework (EMF)*. http://www.eclipse.org/emf/.

[19] Eclipse Foundation. *EMF-based UML 2.0 Metamodel Implementation*. http://www.eclipse.org/uml2/.

[20] OMG. *Human-Usable Textual Notation, v1.0*, 2004.