A Model-Driven Development Approach for Service-Oriented Integration Scenarios

Philip Hoyer, Michael Gebhart, Ingo Pansa, Stefan Link, Aleksander Dikanski, Sebastian Abeck
Research Group Cooperation & Management
Karlsruhe Institute of Technology
Karlsruhe, Germany
{ hoyer | gebhart | pansa | link | dikanski | abeck } @ kit.edu

***Abstract*—The establishment of IT-supported processes within organizations requires the integration of existing distributed legacy applications. Therefore, Web services can be generated as wrappers to flexibly integrate existing distributed legacy applications using a standardized interface. Existing approaches mostly focus on the technical issues of the integration using Web services and do not support the developer during the description of an integration processes. Thus, in this paper we introduce a development approach that supports the developer in describing an integration process and finally allows a model-driven transformation of the prior defined integration process into Web service interfaces and XML schema definitions. Our approach is exemplified by a scenario at the Karlsruhe Institute of Technology (KIT) that implements a process to visualize the study progress of a student.**

Keywords—model-driven development; service-oriented integration; Web services; Unified Modeling Language

I. INTRODUCTION

The integration of existing distributed legacy applications is one of the major tasks when establishing new processes within organization that are expected to be IT-supported. Therefore, the required applications have to be identified and the integration process has to be described. Finally, appropriate adapters have to be developed that realize the technical aspects of the integration.

To formalize the integration process, activity diagrams in the Unified Modeling Language (UML) [1] can be used. They describe the flow of actions that have to be performed and each action represents a functionality of an existing distributed legacy application. As technology to develop the adapters, often Web services are used to provide the required functionality in a standardized manner. The Web service adapters and the according data schemas are mostly developed manually, using the Web Services Description Language (WSDL) [2] and XML Schema (XSD) [3].

Existing approaches mostly focus on the technical aspects of the integration. That means that they focus on the development of Web service adapters and the according data schemas. However, the integration step is a complex task that requires knowledge about existing applications and existing data types to create an appropriate integration process. The input and output data types enable the determination of a possible wiring of functionality without unnecessary data transformation. Additionally, it shows if functionality can be executed in parallel or needs a sequential execution.

Thus, in this paper we propose a development approach that is built upon the existing work about technical integration using Web services. The development approach supports the developer in creating the integration process and allows an automatic generation of code skeletons for the Web service adapters. The development approach is on the one hand service-oriented, which means that the purpose is to provide functionality as a service. On the other hand, it is model-driven, which means that the created integration processes are transformed into the required Web service interface descriptions using WSDL and the according data schemas using XML Schema.

Our approach is exemplified by a scenario at the Karlsruhe Institute of Technology (KIT). Existing distributed legacy applications are integrated to provide a new functionality that allows students to gain a visualized insight into their current study progress. The existing applications and their data types are used to support the developer in creating the integration process. Finally, the integration process is transformed into the required Web interfaces for the Web service adapters.

The paper is structured as follows: Section 2 represents the most relevant related work in the context of modeling workflows with the UML and the transformations into interfaces for Web services. Section 3 illustrates the service-oriented and model-driven approach. Section 4 exemplifies our approach by an integration process at the Karlsruhe Institute of Technology (KIT) to visualize the study progress of students. Section 5 concludes the paper and makes some suggestions for future research work.

II. RELATED WORK

As our approach targets a wide area of different artifacts supporting a model-driven development approach (service model, WSDL and Web services), there are several related studies.

Considering the overall development approach, starting with formal requirements and leading to a set of executable code, Meijler, Kruithof et al. illuminate the advantages of model-driven integration aligned with service-oriented principles [4]. An integrated approach combining both top-down (requirements to software components) and bottom-up (existing tool assets) approaches is proposed. Therefore, we

decided not to follow strictly a top-down development approach that would hamper the integration of existing applications, but to follow a combined middle-out approach enabling the description of existing applications early in the transformation process.

Model-driven development of Web services has already been discussed in several previous works, for instance in [5, 6, 7]. Based on these approaches, we focused on capturing business requirements with models and mapping these models to existing distributed legacy applications. Considering the integration of legacy applications using Web services, a generic model for application integration is presented in [8]. Since different legacy applications often use different formats and standards for describing their data schemas, a mapping of these different data schemas has to be realized additionally. The proposed approach in [8] focuses on the integration of several different data schemas by implementing adapter components realized with Web services. Within the special requirements of our scenario, not only the integration of existing data schemas but also the integration of existing business logic is needed; thus our approach considers the aspect of integration from a system-oriented direction.

Finally, the presented intermediate model for service descriptions (c.f. chapter 3) is based on the work of Emig, Krutz et al. [6]. While the approach presented in [6] targets towards a holistic and technology-independent possibility for describing service interfaces of service-oriented components, we improved the proposed development approach by the integration aspect of existing software assets. Similar to [6], Johnson demonstrates the use of a technology-independent approach for describing service-oriented software components [9]. An UML 2.0 Profile [1] as an extension to existing modeling tools is proposed, although specific modeling elements are introduced regarding the very special needs of the appointed vendor-specific tool chain.

III. Service-oriented and Model-driven Development Approach

In this paper, we present our model-driven software development approach for service-oriented integration solutions. The development process starts with the definition of the requirements. The next step is to model the data types and the workflow according to the defined requirements and the available legacy applications. Afterwards, model-driven transformation techniques are applied, generating formal interface descriptions by transforming the workflow modeled, by means of a UML activity diagram into a service model. Finally, a second transformation step is used to generate Web service interfaces in WSDL and corresponding data types in XML Schema.

A. Analysing the Requirements

The requirements needed for designing the integration solution can be captured using manifold techniques. All techniques for requirement analysis have in common that there is a close collaboration between the customer and the architect or similar roles, since only the customer knows what he expects from the final software solution, but cannot express it in an unambiguously and well-formed form.

Some traditional techniques for requirement elicitation are introspection, questionnaires, interviews or brainstorming [10]. Representation-based techniques use descriptions of scenarios or use cases. A common approach that works well due to our experience is the prototyping of the graphical user interface, since it gives the customer a “look-and-feel” of what the final solution might look like.

Our development approach does not prescribe a concrete technique but rather allow the developer to choose one. Since we propose an approach for integration scenarios, an important part after the requirement elicitation is to analyze existing applications and systems and their containing data, which are required to fulfill the functional requirements.

Based on the defined requirements, the needed data objects are specified. In almost all cases, the data objects can either be derived from the native interface description of the legacy applications, or, if such an interface is not present, reverse engineered from the database schema used by the underlying application. Hence, the desired data objects are modeled as UML class diagrams by using *Classes* with typed *Properties* and *Associations* (note that all UML meta classes are written in *italic*). Many UML modeling tools support the generation of SQL database schemas from UML class diagrams and vice versa.

B. Designing the Workflow

Having analyzed and modeled the required data objects, the next step is to design the workflow in a bottom-up way. During the execution of the workflow, applications are invoked, which provide required data or execute actions. The workflow is represented by an *Activity* (c.f. Fig. 1: Source model, *Activity* “Wf”).

To specify the starting input and the final output of data, the activity might have *ActivityParameterNodes* attached to it (Fig. 1: “wfIn”, “wfOut”). The *Activity* also contains at least one *ActivityPartition*. *ActivityPartitions* are usually used to group some elements in an activity diagram. In our case an *ActivityPartition* represents a legacy application that will be invoked during the execution of the workflow (Fig. 1: “AppX”).

To invoke an application, *CallOperationActions* are used and modeled (Fig. 1: “OpX”). *CallOperationActions* are specialized *Actions*, which have a reference to an *Operation*. As a minor restriction, it is not possible to invoke more than one application within one invocation. Therefore, each *CallOperationAction* must be contained in exactly one *ActivityPartition*. However, since one application can be invoked in many ways to retrieve different data sets, an *ActivityPartition* can contain several different *CallOperationActions*.

The activity diagram is refined by specifying the type of data sent to or retrieved from the invoked applications. The type of data sent to an application by one invocation is modeled by adding *InputPins* and/or *ValuePins* to the *CallOperationAction* (Fig. 1: “xIn”). In contrast, *OutputPins* represent the data returned from an application (Fig. 1: “xOut”). According to the UML meta model [1], a *Pin* is

derived from the *TypedElement* and the *MultiplicityElement* meta class by Generalization. The former enables the user to type a *Pin* with a *PrimitiveType* (such as String, Integer, etc.) or one of the data objects modeled earlier as a *Class*. The later allows the collection of complex data structures in one invocation. The same applies for the *ActivityParameterNodes*.

To represent the data flow between the invocations, we add *ObjectFlows* between *InputPins* and *OutputPins*. The *ObjectFlows* also specify in which order the invocations must be executed. Additionally, if a typed *InputPin* does not have a matching incoming *ObjectFlow*, the required data has to be collected by an additional *invocation*. In such a case, we need to model new *CallOperationActions*, which return the required data and provide an *OutputPin* for that. Of course, the appropriate application which holds the data must be known in advance. Thus the application has to be added as an *ActivityPartition*, if not present yet.

The model containing the *Activity* formalizes the workflow and the legacy applications to be invoked. Due to the *ObjectFlows* it is further specified how data is processed in the workflow and in which order the invocations occur.

C. Transformation to a Service Model

To generate standardized Web-based interface descriptions and data types, the next step is to transform the model described in the previous chapter to a service model [6], which, among other details, specifies the interfaces for each legacy application and the study progress workflow itself.

The transformation rules are formalized in the transformation language "Queries, Views, Transformation" (QVT) [11]. The transformation rules are described by mapping the meta elements of the source meta model to the target meta model. Since the source and target meta model is the UML Superstructure [1] the transformation itself is independent from a concrete platform or technology and thus can be reused for other integration projects of the same kind.

The transformation uses the created *Activity* and the containing model elements as the source model and generates a target model according to a set of transformation rules. Since each *ActivityPartition* represents an application, which will be invoked during the execution of the workflow, Each *ActivityPartition* is transformed into an *Interface* (stereotyped as "ServiceInterface") and a *Component* (stereotyped as "ServiceComponent") with a *Realization* relationship between (c.f. Fig. 1: Target model, "AppXService" and "AppX"). Each *CallOperationAction* contained in an *ActivityPartition* results in an *Operation* of the created *Interface* (Fig. 1: "+opX()").

Finally, *InputPins* and *OutputPins* of the *CallOperationActions* are converted into *Parameters* of the *Operation* (Fig. 1: "wfIn" and "wfOut"). The direction property of each parameter is set to "in" if it is an *InputPin* and no corresponding output pin of the same type and name is attached to the same *CallOperationAction*. An *OutputPin* results in the direction "out". If a *CallOperationAction* has an *InputPin* and an *OutputPin* with the same name, the same type and the same multiplicity, the direction property of the Parameter is set to "inout" and the *OutputPin* is ignored

In order to invoke the workflow itself an additional *Interface* and *Component* are generated from the *Activity* (Fig. 1: "WfService" and "Wf"). The *Interface* contains exactly one *Operation* named "execute<ActivityName>" (Fig. 1: "+executeWf()"). The *Parameters* for this *Operation* are generated according to the *ActivityParameterPins* attached to the *Activity* (Fig. 1: "wfIn" and "wfOut"). In total, n + 1 *Interfaces* are generated, whereby n correlates to the number of invoked applications (or *ActivityPartitions*). Finally, the generated *Component* has *Uses* relationship to all other *Interfaces* generated from the *ActivityPartitions*.

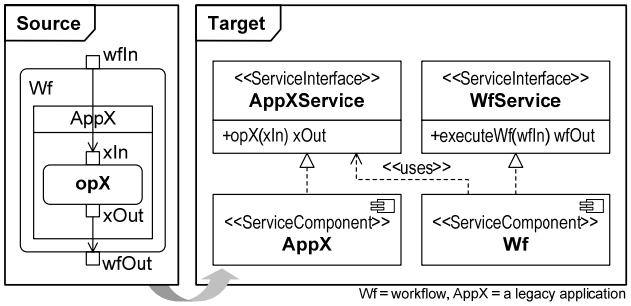

Figure 1. Transformation to the Service Model

It is not required to transform the data types modeled as *Classes*. Still, the data types are needed in the target model. Therefore the *Classes* from the source model, which represent the data types can either be imported in the target model or copied to the target model. The same applies for the *Activity* and the containing *Actions*. The property "operation" of the *CallOperationActions* can now be associated with the generated *Operations* of the *Interfaces*.

D. Transformation into Web Service Interface Descriptions

As the final modeling step, we transform the service model into concrete artifacts that use Web service technologies, namely WSDL [2] and XML Schema [3]. Each UML *Interface* is transformed into a WSDL document and the UML *Classes* are mapped to an XML schema [3]. The transformation rules are mainly straightforward. Each Service Interface is transformed into an abstract part of a WSDL file with exactly one port type. The port type contains the same number of operations as the UML *Interface* specified. The generation of the messages for the input and output of the Web service depends on the WSDL style. Since it is most common and recommended by WS-I [12], we use the style "document/literal-wrapped" [13]. For this style, each message acting as input or output for a Web service contains exactly one part, even if multiple UML *Parameters* are specified as input or output. To distinguish between the Parameters, XML Schema is used to build an RPC-like XML structure, using the operation name as the top XML element. This XML element contains a sequence of child elements, which represent the names and types of the parameters.

The data types specified as UML *Classes* are transformed to one XML Schema file [3], containing all needed data types as complex types. The schema file is imported by every WSDL file generated to have a common set of XML data types for different Web services.

To also generate the concrete part of the WSDL file, the proposed service model can be extended by using UML *Components* and attached *Ports*, as in [6, 9]. A *Port* acts as WSDL bindings and refers to the generated Service Interfaces as provided interfaces or if needed by composite components as required interfaces.

E. Implementing the Web services

To finalize the integration, the required Web services have to be implemented. The generated WSDL and XML Schema files are used to create skeletons for the adapter logic implementation of the web service. For this purpose, existing approaches are applied that are part of several development tools (like WSDL2Java from the Apache Axis2 framework [14]).

The final workflow is implemented in the Business Process Execution Language (BPEL) [15] and is provided as a Web service. The BPEL code can be generated from the UML activity diagram. This issue is already handled in some works [16, 17]. For the sake of simplicity, we omitted this part in the paper but will present it in a future work.

IV. Case Study "Study Progress"

The KIT offers its students the KIT-portal [18], where each student can access his/her personal data and perform actions (e.g., to register for an examination) in a simple and intuitive way. In this paper, we apply our model-driven software development process presented in the previous chapter to the development of a visualization of a student's progress in his/her studies for the KIT-portal.

The KIT-portal integrates several existing applications in a service-oriented manner using Web technologies and Web standards. At the KIT, several applications are available, each storing and providing individual data for students. However, none of the applications provides interoperable interfaces, hence preventing an easy and straightforward service-oriented integration. An important step towards service-orientation is the development of standardized and technology-neutral interfaces for accessing and manipulating the data provided by existing legacy applications [8]. These interfaces and the corresponding adapter logic have to be developed to allow the integration of existing applications.

A. Analysing the Requirements for the "Study Progess"

One feature of the KIT-portal to be developed is meant to facilitate a student's overview of his/her passed, failed or outstanding examinations in a graphical and easily understandable manner. Hence, several GUI sketches and prototypes were created prior to starting the development process, to get the look-and-feel for an adequate visualization form of the study progress. A modified version of a tree map provided the most promising results. It visualizes all the learning modules of a study course by rectangles using an equal width, but different height, depending on the amount of credit points (c.f. European Credit Transfer System, ECTS) of the module. The same applies for the examinations allocated to a module. In addition, each examination is color-coded depending on the current state or result with regard to the student.

Having defined the requirements, we extract the needed data objects for the study progress tree map, such as examination results or personal information about the student. These are persisted in two legacy systems: The study system stores the degree programs and its structures, whereas the examination system holds the data for the offered examinations and the examination results for each student. We create a new UML model and model the data objects as *Classes*. The data types are derived from the database schema used by the systems. We also model the data structure which is needed to generate the study progress tree map.

B. Designing the Workflow "Study Progress"

Next we design the workflow bottom-up. The workflow for visualizing the study progress is represented by a UML *Activity* "StudyProgress" (c.f. Figure 2). To specify the data types the workflow is called with respectively returns, the *Activity* has two *ActivityParameterNodes* attached to it. The KIT-portal invokes the study progress workflow by passing the login name from the KIT-portal (student's university e-mail address) as initial input data (*ActivityParameterNode* "loginEmail") of type string. The workflow completes by returning the output type of the workflow is the tree map data type (*ActivityParameterNode* "studyProgress"). The study system and the examination system are modeled as *ActivityPartitions* ("Study" and "Examination"). The invocation to one of the systems is modeled as a *CallOperationAction* in the corresponding *ActivityPartition* and in addition the type of data transferred to or from a system on each invocation is added as *Pins*.

For example in order to receive the student's base data from the study system we model the *CallOperationAction* "GetStudentBaseData" in the *ActivtyPartition* "Study" and add the *OutputPin* "student" of the type "Student" (the classes modeled before). The call to the study system requires the matriculation number and the current term, so we model those by adding the two *InputPins* "matricNumber" and "term". Since the portal system only knows the student's university e-mail address, which has to be entered during the KIT-portal login, we add an *ActivityPartition* for the accounting system and model the *CallOperationAction* "GetMatricNumber" inside. It accesses the accounting system, maps the student's email address to his/her matriculation number and returns the number (*OutputPin* "matricNumber"). The current term can be retrieved from the examination system. Thus, we add the *CallOperationAction* "GetCurrentTerm" in the *ActivtyPartition* "Examination" with only one *OutputPin* "term" containing the current term as an integer value.

To represent the data flow between the invocations, we add *ObjectFlows* between *InputPins* and *OutputPins* that have the same type. The *ObjectFlows* specify in which order the invocations occur.

Figure 2 shows the final activity diagram labeled as "Workflow Model" in the upper part. We have formalized which applications are invoked and how the data is processed.

C. Transformation to a Service Model

Taking the activity diagram as a source model, we use a model-to-model transformation to generate service interfaces. The transformation generates a service interface for each invoked application. In order to invoke the workflow itself from the KIT-portal, another service interface "StudyProgressService" that contains the *Operation* "executeStudyProgess" is generated. The Parameters for this *Operation* are generated according to the *ActivityParameterPins*.

The middle part of Figure 2 shows the resulting Interfaces for each *ActivityPartition* and the *Activity* itself. The grey dashed lines show some exemplary transformations from the activity diagram model elements to model elements of the Service Model. We omitted the generated *Components* and most stereotypes in Figure 2 as specified in [6].

D. Transformation into WSDL and XML Schema

On the basis of the service interfaces and the data type classes a model-to-text transformation creates four WSDL documents [2] (one for each service interface) and one XML Schema document [3] ("StudyProgressTypes.xsd"). The available operations of the port types in the WSDL documents match the operations of the service interface. To facilitate the reusability of the XML Schema definitions the StudyProgressTypes.xsd file is imported into the "types" section of each WSDL document.

Figure 2 illustrates the generated artifacts and the import of the central XML Schema definition at the bottom. Part of the WSDL document for the StudyService is also shown in detail.

E. Implementing the Web service adapters

Finally, the generated WSDL documents are used to create skeletons. We implement the adapter logic of the required Web services. The study progress process itself is implemented in the Business Process Execution Language (BPEL) [15] according to the UML Activity. We use an XSL transformation to generate XHTML from the tree map data structure defined before. Figure 3 gives the result of the engineered solution, showing a late prototype of the study process.

V. Conclusion and Outlook

In this paper, we outlined how a service-oriented integration of existing distributed legacy applications can be realized through our model-driven development approach. Our approach supports the developers in creating an appropriate integration process. Existing legacy applications and their data types in use are identified. Due to the bottom-up nature of our development approach, unnecessary data transformation can be avoided and functionality that can be executed in parallel can be identified. Afterwards the integration workflow, which is modeled using UML activity

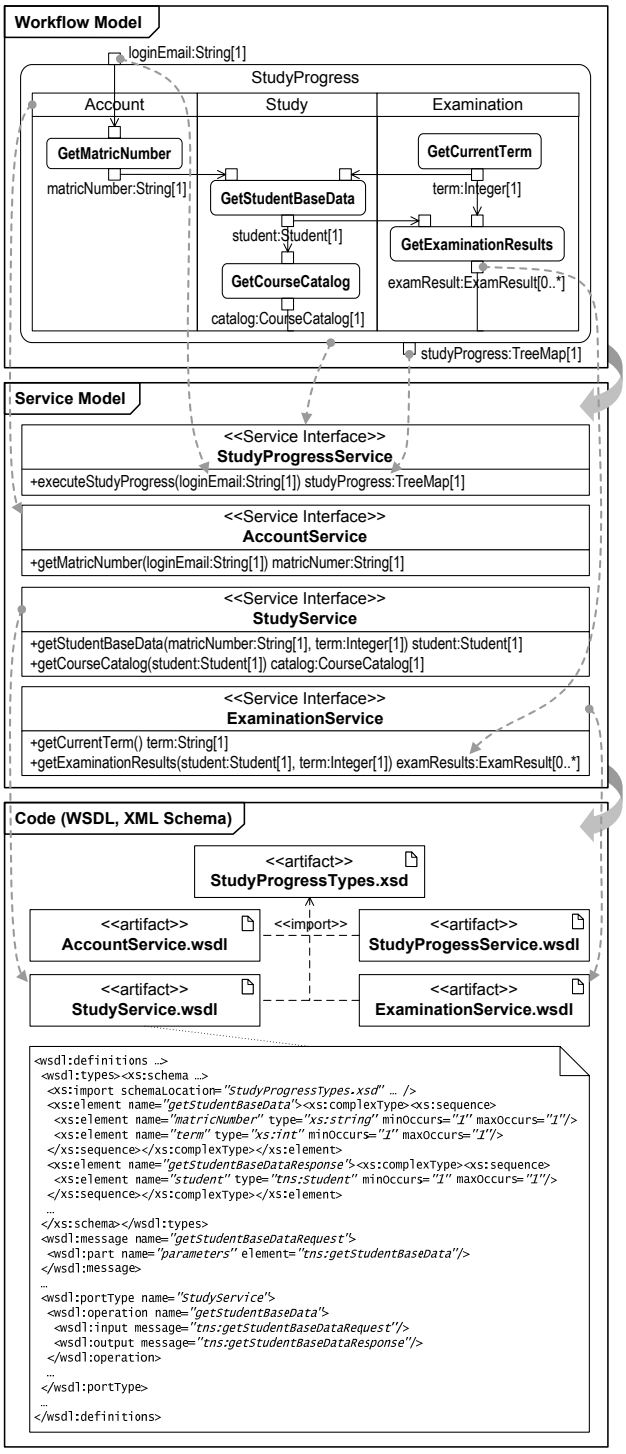

Figure 2. Model-driven development process of the "study progress"

diagrams, is automatically transformed into a technology-independent service model. This service model can be further refined and transformed into Web service interfaces and XML Schema definitions. Thus, only functionalities that are required for the solution are exposed as Web services.

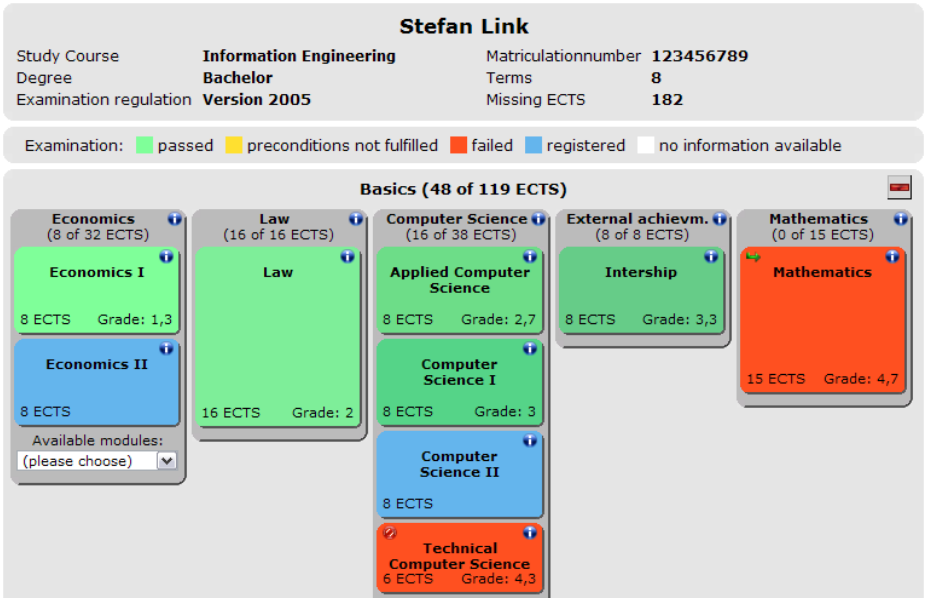

Figure 3. Screenshot of the "study progress" in a nearly final stage

The application of the standardized modeling language UML and of Web services, which allow an integration in a standardized manner, allows the usage of wide-spread modeling and development tools. The model-driven development approach targets a high level of formalization and therefore supports automatic transformations of models into more concrete models or code, which helps to avoid misunderstandings and reduces errors during the development process.

We exemplified our approach by a scenario at the Karlsruhe Institute of Technology (KIT). Here, our approach was applied to realize a visualized study progress that requires existing distributed legacy application being integrated.

Due to the successful realization of the study progress at the KIT, we plan to establish this development approach for future works as an integrated course catalog and a library that require the integration of various distributed legacy applications. Additionally, we plan to consider further design aspects when creating the Web services to create a reusable set of Web services with appropriate granularity.

REFERENCES

[1] Object Management Group (OMG): Unified Modeling Language (UML), Superstructure Version 2.2. http://www.omg.org/cgi-bin/doc?formal/09-02-02

[2] World Wide Web Consortium (W3C): Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/wsdl20/

[3] World Wide Web Consortium (W3C): XML Schema Definition Language (XSD) 1.1 Part 1: Structures. http://www.w3.org/TR/xmlschema11-1/

[4] Meijler T.D., Kruithof G., Beest N.: Top Down Versus Bottom Up in Service-Oriented Integration: An MDA-Based Solution for Minimizing Technology Coupling, LNCS Volume 4294/2006.

[5] Marcos E., Castro V., Vela B.: Representing Web Services with UML: A Case Study. 1st International Conference on Service-Oriented Computing (ICSOC), Trento, Italy, December 2003.

[6] Emig C., Krutz K., Link S., Momm C, Abeck S..: Model-Driven Development of SOA Services, Cooperation & Management, Universität Karlsruhe (TH), Internal Research Report, 2008.

[7] Gronmo R., Skogan D., Solheim I., Oldevik J.: Model-driven Web Service Development. International Journal of Web Services Research, Volume 1, Number 4.

[8] Harikumar A., Lee R, Yang H., Kim H., Kang B.: A Model for Application Integration using Web Services, Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science, July 2005.

[9] Johnston S.: UML 2.0 Profile for Software Services, IBM developerWorks http://www.ibm.com/developerworks/rational/library/05/419_soa/, April 2005.

[10] Hay D.: Requirement Analysis – From Business Views to Architecture. Prentice Hall, 2003.

[11] Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.0. http://www.omg.org/spec/QVT/1.0

[12] Web Services Interoperability Organization: Basic Profile Version 1.2. http://www.ws-i.org/Profiles/BasicProfile-1_2(WGAD).html

[13] Butek R.: Which style of WSDL should I Use, IBM developerWorks, 2003. http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

[14] The Apache Software Foundation: Code Generator Wizard - eclipse Plug-in, http://ws.apache.org/axis2/tools/1_0/eclipse/wsdl2java-plugin.html

[15] Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[16] Mantell K.: From UML to BPEL. IBM developerWorks, 2005. https://www.ibm.com/developerworks/library/ws-uml2bpel/

[17] Skogan D., Groemno R., Solheim I.: Web service compositions in UML. Proceedings of Eudth International Enterprise Distributed Object Computing Conference, September 2004.

[18] Karlsruhe Institute of Technology (KIT): The KIT study portal, http://studium.kit.edu