Service-Oriented Integration Using a Model-Driven Approach

Philip Hoyer, Michael Gebhart, Ingo Pansa, Aleksander Dikanski, Sebastian Abeck
Research Group Cooperation & Management
Karlsruhe Institute of Technology
Zirkel 2, 76131 Karlsruhe, Germany
{ hoyer | gebhart | pansa | a.dikanski | abeck } @ kit.edu

***Abstract* — Provision of processes supported by Information Technology (IT) spreading around several different units of one organization requires the integration of existing distributed legacy applications. Typically the part of the application's functionality used in a process is offered through proprietary interfaces, complicating the integration. A possible solution to this issue is to construct standards-based, service-oriented interfaces offering only the required functionality. Existing approaches within this field mostly focus on the technical issues of the integration using Web services and hardly consider the integration from the perspective of the IT-supported processes. In this article, we introduce a development approach for modeling an IT-supported process which is enhanced by the automatic generation of necessary Web service artifacts. 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

Due to fast changing markets and emerging requirements, an organization's Information Technology (IT) needs to be flexible in order to quickly provide new functionality. Usually this flexibility is required at the high level of rapidly changing business processes, which necessitates the implementation of IT-supported business processes. Several applications already exist and are in practical use within today's organizations, providing basic functionality and data. Often these existing legacy applications can hardly be replaced or enhanced with new functionality due to high costs or the associated high complexity. If new functionality has to be added, it should reuse existing functionality in order to reduce costs. Thus, the realization of new functionality requires the integration of these existing applications. Using existing applications in an integration scenario is complicated by the proprietary interfaces these applications provide. Additionally not all of the functionality of an application might be used in a new IT-supported process. To overcome these issues, a service-oriented architecture (SOA) is a widely accepted approach for process-oriented integration scenarios, but still the development of standardized interfaces is carried out by hand, leading to high development costs as well as to long and error-prone development cycles.

In a typical top-down integration approach, in which the needed functionality of the legacy applications is determined by the IT-supported process to be implemented, the process has to be formally modeled beforehand. It describes the flow of actions that have to be performed for the new functionality. Each action represents a functionality provided by one of the existing applications. The workflow that integrates the existing applications can be formalized using Activity Diagrams of the Unified Modeling Language (UML) [2]. In a next step, the existing applications have to be made accessible to reuse existing functionality. For this purpose, adapters have to be developed to perform the integration on a technical level. They enable the access to existing functionality in a convenient way. Figure 1 illustrates this approach of developing new functionality by integrating existing functionality. As technology to realize the adapters, Web services can be used as they represent a standardized technology that is platform and programming language neutral. Web services are de facto standard in the context of service-oriented architectures. To provide the functionality using Web services, the interface description and the data schemas have to be developed using the Web Service Description Language (WSDL) [3] and XML Schema (XSD) [4]. Providing the functionality as a Web service enables a convenient usage of it within different contexts. To decouple the Web service from the existing applications, it is desirable to abstract from existing data types within the existing application and to create the data schema used within the Web service from a more conceptual view.

Existing integration approaches mostly focus on solutions on technical levels. That means that they focus on the development of Web service adapters including technical aspects such as the usage of various Web service standards. The conceptual issues such as the modeling of an IT-supported process that can be effectively realized are subject of further investigation. This requires the analysis of the existing applications to avoid unnecessary data transformations when realizing the workflow that integrates the existing applications. For example, the data types available within the existing application can provide an indication about how to design an appropriate IT-supported process.

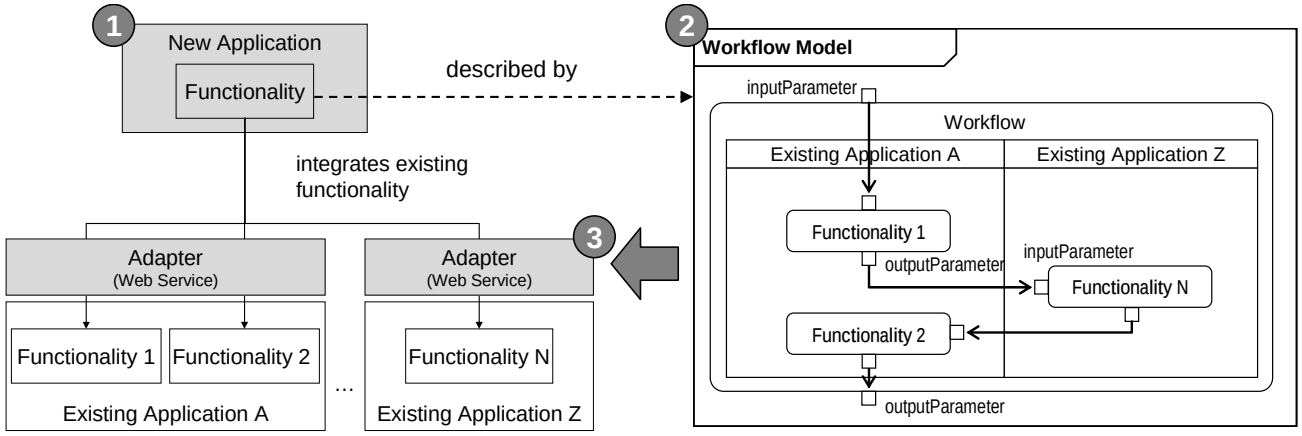

Figure 1. Integration of Existing Functionality

In this article we propose a development approach that supports the developer in creating the IT-supported process and helps to speed up the workflow that integrates the existing applications by automatically generating Web service adapters for existing applications and a Web service for the newly developed functionality. The presented approach in this paper extends our model-driven development method described in [1]. To get a better understanding of the IT-supported process, we start with mock-ups and sketches of graphical user interfaces (GUI). They enable the identification of required functionality. As a complement a domain model is created, allowing the derivation of conceptual data types that are independent of existing applications. Afterwards, an analysis of existing applications is performed that provide the required functionality identified with the help of the GUI sketches. This analysis enables the creation of the process with a reduced set of data transformations. The process is formalized using Activity Diagrams of the Unified Modeling Language (UML). In a next step, the Web services and data schemas are automatically derived from the formalized process and the domain model using an enhanced version of our model-driven development method [1]. The enhancement lies in the usage of the domain model which enables the decoupling of the Web service providing the new functionality from existing applications by creating data types that are independent from existing applications.

The derivation of the Web service artifacts is a two step process, starting with a platform-independent model representing the service interfaces and data schemas on a conceptual level and the final platform-specific realization using Web services. The approach targets a service-oriented integration in which functionality is provided as a service. The approach itself is model-driven, which means that the required Web service interface descriptions using WSDL and the data schemas using XML are generated automatically from the workflow that integrates the existing applications and the domain model. To realize our approach, existing work in the context of developing Web service adapters is reused. Also, existing guidelines how to derive data schemas from a domain model are applied.

Our approach is exemplified by a university scenario at the Karlsruhe Institute of Technology (KIT). In this scenario, the goal is to provide a new feature that allows students to gain insight into their current study progress, by combining and visualizing their data from disparate sources. The required functionality and data is shared across several existing applications, so that an integration workflow is necessary, which accesses these applications and combines the collected data. In a first step, the requirements are analyzed. The domain model for the study progress scenario and several GUI sketches are created. Based on these requirements, the required applications are identified and the process is created and formalized. Afterwards, the process and the domain model are transformed into a description of the required services. In a last step, the necessary Web service interfaces using WSDL and data schemas using XML Schema are created.

The rest of the article is structured as follows: Section 2 introduces the background and gives general information about integration issues and their solutions. In Section 3 the concept of our approach of a service-oriented integration using a model-driven approach is described. The practical implications and issues of our approach are exemplified by the aforementioned case study of a study progress visualization in Section 4. Section 5 presents the most relevant related work in the context of integrating existing functionality, modeling workflow with the UML and transformation into Web services. Section 6 concludes this article, discusses the achieved results, and presents an outlook as well as suggestions for future research work.

II. BACKGROUND

The necessity for building integrative solutions comes along with the evolution of the way Information Systems are used. We thereby define an Information System (IS) as all the components and algorithm that are necessary for enabling IT-based computation of information. In the very beginning of supporting business activities through IT, the typical architecture of an IS was structured using two logical layers – a layer representing the client side and a layer representing the centralized business logic and data stores –

typically on a single tier or host. The availability of today's high performance networks connecting datacenters all over the world has lead to a multi-layer and multi-tier structuring of distributed business logic and distributed data stores. Figure 2 shows a comparison of these two approaches (based on [20]). It is obvious that while in the late 1960's one centralized datastore and computation logic served multiple clients, we are faced not only with distributed computation logic today but also distributed data stores that might be connected with many-to-many relationships amongst together.

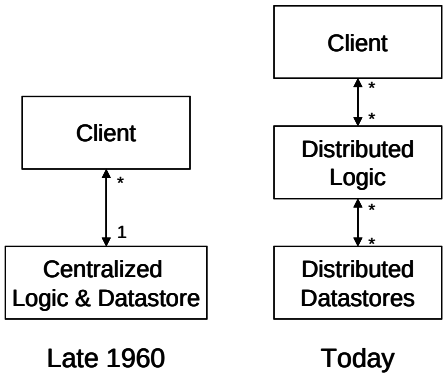

Figure 2. Evolution of Information Systems

Although separating the business logic from the data stores and enabling access to these components in a distribution way has many advantages, a logically holistic access to the information these systems contain is desired. Furthermore, not only access to information is needed but also an evolution of functionality is desired. Therefore reasons for aiming for integrative solutions in order to capture a holistic access to information are manifold: increasing the efficiency of business-related tasks, extending existing applications with new functionality, reusing existing applications, saving of software investments, avoid the costs of introducing a new software system. According to [21] the term "application integration" describes a strategic approach to couple different existing applications for the purpose of simplifying business-related tasks. The benefit of integration within these scenarios is more than just a conciliation of existing functionality. Rather, the idea is to leverage synergetic effects and thus to increase efficiency of IT-supported business activities.

For the purpose of constructing software-based solution logic, a clear distinction of the necessary development steps is necessary. There exists several different classification schema for describing levels and approaches for application integration [20],[21]. A common agreement to a basic classification scheme comprises four categories: information-oriented application integration (IOAI), service-oriented application integration (SOAI), business process-oriented application integration (BPOAI) and user interface-oriented application integration, which is also referred to as portal-oriented application integration (POAI). For a better understanding of the conceptual contribution of our work, at least some basic knowledge of the fields of application of each of the single possibilities is necessary, which is why we briefly describe each single category and highlight the main points of interest.

A. Information-Oriented Application Integration

Information-oriented integration is a simple approach for integrating several existing systems by considering the extraction of information of a source system and deciding how to convict this information into one or many different target systems. Almost any information system that is used today in a typical business scenario follows an n-tier architecture based on a database or a data store component enabling the information-based integration to easily be introduced. This is often the only possibility if changes to the business logic of an existing information system cannot be performed.

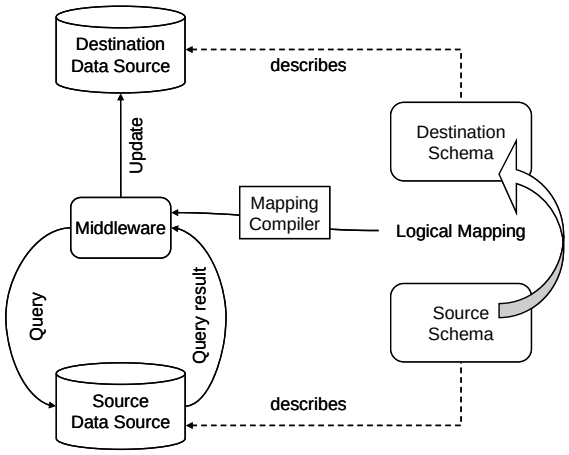

Figure 3. Concepts in Information-Oriented Application Integration

Figure 3 outlines the principles of this approach. The different data sources are described using different schema descriptions. These descriptions have to be logically mapped to each other, which can either be done at runtime or in less frequent cases, at design time.

Although the advantages of this approach are its simplicity and often fast-to-develop solution, this approach bares a couple of disadvantages. Often it is not clear in advance to what extends the desired solution needs to be based on integrated data stores. This leads to a couple of single integrative island with a total amount A where $A = 1/2 * \nu * (\nu+1)$ connections for ν different data stores ending in solutions that are hard to maintain or to evolve.

Further, this approach is not aligned with requirements derivable from the overall business processes, making changes at the business level hard to be propagated to the supporting IT level.

B. Service-Oriented Application Integration

Often it is not sufficient to only consider the information that existing applications operate on but also to enable access to functional capabilities the existing applications offer. By focusing the functionality in means of interface-oriented

semantics, one can think of a service a distinct application offers, leading to a new aspect of layering the possibilities of integration: service-oriented application integration (SOAI).

Simply speaking, SOAI allows applications to share common business logic [21]. The idea is to identify objects of reuse within an organization and enable access to these reusable objects through standardized service interface. Although the concept of reuse can hardly be assumed in advance, in the last years a couple of technologies and platforms to realize this vision have surfaced, namely Web service technology with its standardized interface descriptions and access methods using WSDL [3], XML [4] and SOAP [22].

Service-orientation is rather a design decision than a concrete solution to all the integration issues. Many questions are still open, for instance, it often is not clear how to design services in order to fulfill certain quality attributes such as loose coupling.

C. Business Process-Oriented Application Integration

While IOAI occurs at the level of data exchange [21], the concept of business process-oriented application integration (BPOAI) can be seen as an advancement of the IOAI and SOAI by focusing not only the data level of each single application but considering the overall process that each of the single applications participate with. Therefore, BPOAI takes the flow of information on a more abstract level, enabling an admission to the integration issue on a level that is more independent of the concrete data schemas.

Having the existing applications organized using service-oriented interfaces can be seen as a requirement for an efficient support of the business processes. Typical implementations of service-oriented integration scenarios are based on Web services, thus enabling the descriptions of interfaces and exchanged data schemas using XML. Such a standardized approach enables the usage of models for the descriptions of the processes and is proposed to lead to a more formal approach to solve integration issues, as models can easily be reused. Therefore, great efforts have been made to introduce modeling languages that are based on formal meta models in order to support direct transformations from the modeled business process to a concrete architecture supporting these modeled processes. Examples of these languages include the UML Activity Diagrams or the Business Process Modeling Notation (BPMN) [23] with its upcoming release 2.0 but also approaches that are based on mathematical formalism such as Petri nets [24] or event-driven process chains [25].

D. User Interface-Oriented Application Integration

Focusing the user interface for solving integration issues aims at enabling a single point of access to a multitude of existing user interfaces the existing applications have. While the first three approaches (IOAI, SOAI, and BPOAI) consider the exchange of information for the purpose of automating parts of business processes, the application integration on the level of user interfaces takes the human factor into account. Still many steps of business processes cannot be fully automated today, thus the need for enabling access to information or operations distributed applications provide are still a requirement. One of the biggest problems with user interface-based integration is the fact that almost any graphical user interface (GUI) of traditional business applications is tightly coupled with monolithic frameworks making it all but impossible and infeasible to make the functionality a GUI offers available to external software artifacts.

In the recent yeas, the concept of portals evolved for Web-based applications. User interface-oriented application integration therefore is often called portal-oriented application integration (POAI). A portal thereby is a Web browser-based approach for constructing a distributed architecture consisting of a portal server, a framework for generating and operating pluggable parts (portlets) of the user interface and a couple of connectors to access the existing applications.

E. Discussion

We currently observe a shift from simple information-based approaches to service-oriented approaches because of the several advantages of SOAI and because of the disadvantages of the other approaches pointed out in the previous descriptions. Although IOAI can be the solution to choose in small scenarios, SOAI proposes to have a better acceptance due to the enablement of business process-oriented aspects. A clear separation of IOAI and SOAI can be given by arguing that IOAI solely focuses the exchange of information, while SOAI considers not only the information but also the methods that operate on the information. As a major goal of the integration project we applied our development approach on was to create a solution that is accepted by a wide range of students having different skills in using Web-based information systems, the integration of the existing applications should lead to a single user interface that can be extended with further functionality on demand. Figure 4 relates the different aspects of integration of our approach, showing the flow of information across all elements of the architecture to be integrated.

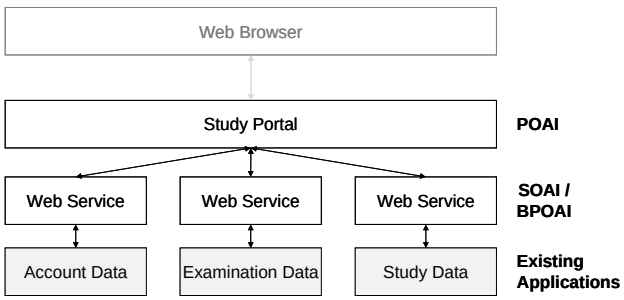

Figure 4. Different Aspects of Integration in our Approach

Our development approach therefore focuses the construction of service-interfaces for supporting both a flexible realization of business processes and usage of portal technology. For the purpose of simplifying the development method, we equivalently use the term “business process” and “workflow” as we focus on the technical representation of a business process which we consider to be a workflow [36].

This is not a constraint in our opinion as we argue that a technical representation of a business process can be found and the relevant automatable steps can be identified. For increasing the quality of the engineered solution, we use models and model transformations to omit unnecessary manual steps in code generation. Figure 5 outlines the steps in our development process that are based on model transformations and shows which of the development steps have to be performed using manual model transitions.

III. SERVICE-ORIENTED INTEGRATION USING A MODEL-DRIVEN APPROACH

In this chapter we describe our model-driven software development approach for a service-oriented integration of legacy applications using the Unified Modeling Language (UML). The overall goal of the development process is to develop a new high-level service by integrating legacy applications via service adapters (low-level services) and specify a workflow to compose those low-level services in such a way that they create the high-level service. Since existing applications used in the composition might be substituted in the future by newer applications, which provide new or better functionalities, the final solution must consider the adaption to a changing IT-landscape with reasonable time and effort.

The proposed development process contains six steps but does not cover the whole software lifecycle. For example, the test and deployment step is omitted as it does not significantly differ from other software development processes. Furthermore our solution should be adapted to the requirements of the integration project, adding new steps or leaving out steps described here.

Our development process starts with the definition of the requirements (Figure 5: A). Due to our experience we suggest the development of a prototype of the graphical user interface, but other methods can also be used. After capturing the requirements the next step is to develop a domain model (Figure 5: B). It contains the entities derived from the requirements. The domain model is of conceptual nature and independent from concrete applications. Having completed the domain model, the development process continues with the design of the workflow according to the defined requirements (Figure 5: C). In this step we regard the available legacy applications but also use the entities from the domain model created in the previous step. This allows us to avoid dealing with data transformation issues, since in the legacy systems the entities might be represented as different data types. Afterwards, model-driven transformation techniques are applied, generating formal interface descriptions and executable workflow definitions by transforming the workflow model and the domain model into a service model (Figure 5: D). All created and generated models so far are independent from a concrete technology. A final transformation step generates the necessary Web service artifacts – WSDL for service interfaces, XML Schema for data types and BPEL for workflow definition from the service model (Figure 5: E). Note that we use Web service technologies for the integration solution since it is most common, yet the transformations can be rewritten to generate artifacts based on other technologies than Web services using the same models (as suggested by the concept of Model-Driven Architecture). At last the Web service adapters are implemented based on the generated WSDL interfaces, which is still done by manual work (Figure 5: F). The adapter can either accesses a database used by the legacy application or uses a native interface provided by the legacy application.

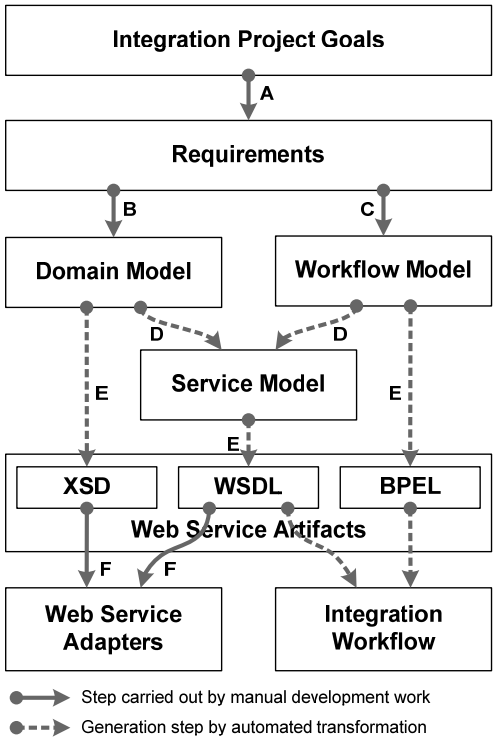

Figure 5. Model-Driven Development Process Overview

In the following the development process is described in detail.

A. *Capturing 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 way.

Some traditional techniques for requirement elicitation are introspection, questionnaires, interviews or brainstorming [11]. Representation-based techniques use descriptions of scenarios or use cases. Our development approach does not prescribe a concrete technique but rather allows the developer to choose one that fits best to the project. Since we propose an approach for integration scenarios, an important part after the requirement elicitation is to analyze existing applications and systems, which are required to fulfill the functional requirements.

A common approach that works well in 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. The requirements can than be deduced from that prototype.

B. Creating the Domain Model

Based on the defined requirements, the next step is to create a domain model. A domain model is a static model that contains relevant entities of that domain and their relations [26]. It does not contain dynamic issues, like sequences or information flows. Consequently we use UML class diagrams so that each entity in the domain model is formalized as an UML *Class* with typed *Properties.* Furthermore we use *Associations* and *Generalizations* (note that all UML meta classes are written in *italic*) to specify relations between entities. All *Properties* should be typed as primitive data types (like string, integers, boolean values etc.). Relations between entities are formalized by modeling *Associations* between the corresponding *Classes*. *Packages* can be used to classify entities into logical groups.

In contrast to the approach described in [1] this approach propose to design the entities rather abstract by understanding the domain the integration project is settled in and not derive them from the existing legacy applications. Nouns and noun phrases in the requirements specified in the previous step can help to identify relevant entities. Although this seems to be a rather complex and time-consuming task than deriving the entities directly from the legacy applications, the effort put into the domain model will pay off later when new or changing requirements have to be implemented or new applications substituting the legacy applications have to be integrated into the existing solution. We also recommend using or building upon existing standardized domain models, such as ebXML in the domain of electronic business [35], complementing our use of open standards for describing the service interfaces and data types. Using such standardized domain models simplifies the integration of functionality of third party services as well as it provides the ability to offer the service to third parties.

With the domain model, our integration solution use data types that are independent from the concrete legacy applications. The workflow only operates with data objects from the domain model and therefore does not have to cope with transformations of different data types representing the same entity that otherwise would be necessary to implement in the workflow. At runtime the data transformations between the legacy applications and the workflow are carried out in the service adapters. Hence the service adapter transforms native data objects from the legacy application into data objects as defined in the domain model and vice versa. The legacy applications can be replaced by new applications at the cost of rewriting the relevant service adapters, but the workflow does not have to be adapted, since the interface of the service adapter remains the same.

C. Designing the Workflow Model

After the completion of the domain model, the next step is to design the workflow model in a bottom-up way. During the execution of the workflow, the legacy applications are invoked and provide required data or execute actions. In contrast to the domain model, the workflow model is a dynamic model. The workflow model makes use of the entities defined in the domain model though. In the model the workflow itself is represented by an UML *Activity* (c.f. Figure 8: Source model, *Activity* "Wf").

Many workflows require some initial data transferred from the invoking application before the workflow can be executed. Also after the completion of most workflows some data is returned to the application that has called the workflow. To specify those data objects, the activity can have *ActivityParameterNodes* attached to it (Figure 8: "wfIn", "wfOut"). An *ActivityParameterNode* always has a reference to a *Parameter*. The *Parameter* is either typed with a primitive type or a concrete *Class* from the domain model. Furthermore the Parameter requires a direction type that indicates if the value of the Parameter is passed into the workflow or from the workflow.

To represent the legacy applications in the workflow model that are invoked at runtime *ActivityPartitions* are used (Figure 8: "AppX"). *ActivityPartitions* are usually used to group *Actions* in an *Activity* that share some common characteristics, e.g., belonging to the same organization unit. The *Activity* representing the workflow must contain at least one *ActivityPartition*, because otherwise there would be no legacy application to call.

To call a legacy application, *CallOperationActions* are modeled (Figure 8: "OpX"). *CallOperationActions* are more 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* (Figure 8: "xIn"). In contrast, *OutputPins* represent the data returned from an application (Figure 8: "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.

D. Transformation to Service Model

To generate standardized Web service-based interface descriptions and data types, the next step is to generate a new model by using model-driven transformation techniques. From the domain model and the workflow model a transformation specification generates a service model [7], which, among other details, specifies the interfaces for each legacy application and the study progress workflow itself.

Since the transformation to the service model generates a new model from existing models, the transformation rules are formalized in the transformation language "Queries, Views, Transformations" (QVT) [12], a standard specified by the Object Management Group (OMG). Several QVT implementation exists, e.g. in Borland Together [27], but also as plug-ins for the Eclipse IDE (mediniQVT [28], smartQVT [29]). Simply speaking, the transformation rules are described by mapping the elements of the source meta model to elements of the target meta model. Since the source meta model and target meta model is the UML Superstructure [2] 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. Figure 8: Target model, "AppXService" and "AppX"). Each *CallOperationAction* contained in an *ActivityPartition* results in an *Operation* of the created *Interface* (Figure 8: "+opX()"). Figure 6 shows this transformation in the graphical notation of QVT.

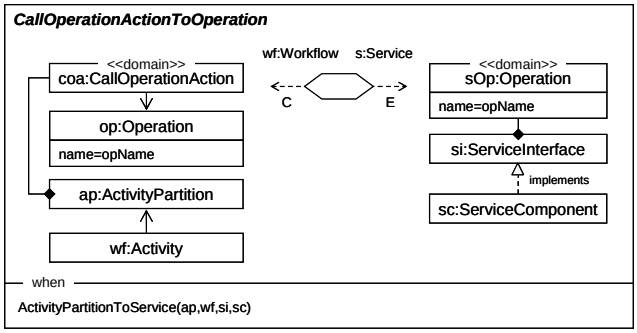

Figure 6. QVT Transformation of CallOperationAction

Finally, *InputPin*s and *OutputPin*s of the *CallOperationActions* are converted into *Parameter*s of the *Operation* (Figure 8: "wfIn" and "wfOut"). The *direction* property of each *Parameter* is set to "*in*" if it is an *InputPin* and no corresponding *OutputPin* of the same type and name is attached to the same *CallOperationAction* (Figure 7). 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.

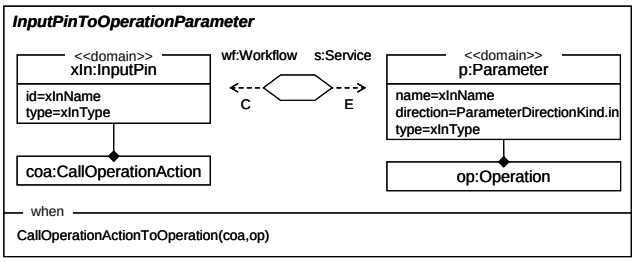

Figure 7. QVT Transformation of InputPin

In order to invoke the workflow itself an additional *Interface* and *Component* are generated from the *Activity* (Figure 8: "WfService" and "Wf"). The *Interface* contains exactly one *Operation* named "execute<ActivityName>" (Figure 8: "+executeWf()"). The *Parameters* for this *Operation* are generated according to the *ActivityParameterPins* attached to the *Activity* (Figure 8: "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*.

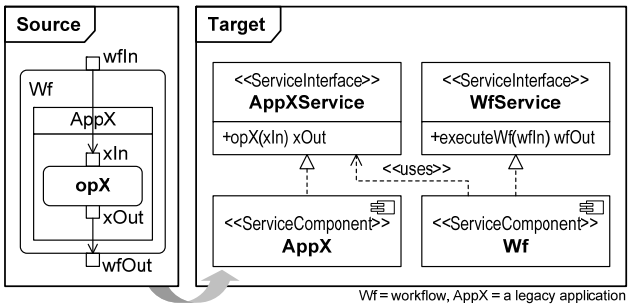

Figure 8. Transformation to the Service Model

It is not required to transform the entities from the domain model. Still, the specified 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*.

E. Transformation into Web Service Artifacts

As the final modeling step, we transform the three UML models into concrete XML artifacts. The transformation converts the domain model into XML Schema definitions [4], the service model into WSDL documents [3], and the workflow model into a BPEL process [17, 30]. As far as we know the relatively new "MOF Model to Text Transformation Language 1.0" [31] specified by the OMG is currently not implemented in common UML tools. As we also prefer a more established Model-to-Text transformation language, we decided to use Xpand [32], a templated-based approach from the openArchitectureWare toolkit, which is now part of the Eclipse IDE.

TABLE I. DOMAIN MODEL TO XML SCHEMA

UML	XML Schema
Package p	`<xsd:schema>` target namespace is derived from *p* and its parents

Class c	`<xsd:complexType>`
	name of complex type is name of *c*
	`<xsd:sequence>`
	model group for elements

Attribute a	`<xsd:element>` name of element is name of *a*, type of element is used from *a* (Primitive types are matched to similar build-in XSD types), minOccurs and maxOccurs of element is cardinality of *a*
Association s	`<xsd:element>` name is set to name of *s*, type of element is complex type of *Class* at *AssociationEnd*, minOccurs and maxOccurs of element is cardinality of *s*

Enumeration e	`<xsd:simpleType>`
	name is set to name of *e*
	`<xsd:restriction>`
	base type of restriction is set to "string"
	`<xsd:enumeration>`
	for each *EnumerationLiteral el*, value is name of the *el*

Generalization g	`<xsd:complexContent>`
	as child for complex type of specialized *Class*
	`<xsd:extension>`
	base type is set to complex type of generalized *Class*

The entities defined in the domain model and specified as UML *Classes* are transformed into XML Schema definitions (XSD) [3]. Most transformation rules are mainly straightforward (Table I). The name of the model and the structure of UML *Packages* (if present) are used to generate the target namespace of the XSD. UML *Classes* are transformed into XSD complex types with a sequence model group and all *Properties* of *Classes* into XSD elements of the generated model group. These XSD elements are typed depending on the kind of the UML *Property*: If it is an *Attribute* with a primitive type, a build-in XSD data type is used. If the *Attribute* uses a custom UML *Enumeration*, an additional XSD simple type with a restriction of base type set to "string". For each literal of the Enumeration an enumeration with the value set to the name of the literal is generated. If an *Association* is used, the XSD element is typed with the corresponding XML complex type of the associated *Class*. Furthermore the cardinalities of *Properties* are considered by using the "minOccurs" and "maxOccurs" attributes in the XSD element definition. UML *Generalizations* are also supported by complex content and extensions in XSD, using the complex type of the generalizing *Class* as "base" attribute of the "extension" element.

TABLE II. SERVICE MODEL TO WSDL

UML	Web Service Description Language (WSDL)
Component c	`<wsdl:definition>` name of definition is name of *c*
Interface i	`<wsdl:portType>` name of port type is name of *i* appended with "PortType"

Operation o	`<wsdl:operation>`
	name of operation is name of *o*
	`<wsdl:input>` and `<wsdl:ouput>`
	name of message is name of *o* appended with "Request" or "Response"
	`<wsdl:message>`
	name of message is name of *o* appended with "Request" or "Response"
	`<wsdl:part>`
	name is "parameters" and element is corresponding XSD element name
	`<xsd:element>`
	name of element is name of *o*, appended with "Response" once
	`<xsd:complexType>`
	`<xsd:sequence>`
	model group for *Parameters*

Parameter p	`<xsd:element>` name of element is name of *p*, containing model group of element depends on *p.direction* ("in" or "out"/"reply"), type of element is type of *p* (either a build-in XSD type or a complex type)

The WSDL documents are generated from the service model (Table II). Each UML *Interface* is transformed into a WSDL document, importing the generated XSD files in the "types" section of the WSDL document. Each interface is transformed into an abstract part of a WSDL file with one port type. The port type contains the operations as the UML *Interface* specifies. The generation of the messages for the input and output messages of the Web service depends on the WSDL style. Since it is most common and recommended by WS-I [13], we use the style "document/literal-wrapped" [14]. For this style, each message element in the WSDL document must contain 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 and an embedded complex type

defining a sequence of child elements, which represent Parameters. To generate the concrete parts of the WSDL file, the proposed service model uses UML *Components* and attached *Ports*, as in [6], [9]. A *Port* acts as a WSDL binding, specifying a name and location information about the service. It refers to the *Interface* as provided interface.

The workflow model is implemented in the Business Process Execution Language (BPEL) [30] and is provided as a Web service. The BPEL code is generated from the UML Activity Diagram, which is already handled in some works [17], [18]. The *Activity* and each *ActivityPartition* of the workflow are defined as partner links in the BPEL process and partner link types in the WSDL document of the BPEL process. The defined *ActivityParameterNodes* of the *Activity* transforms to the initial "receive" and the final "reply" action in the BPEL process. In addition corresponding variables for are generated. Each *CallOperationAction* is transformed to an "invoke" action, using the partner link derived from the *ActivityPartition* the *CallOperationAction* is placed in. Corresponding variables in BPEL for the input and output message of the "invoke" action are generated. *ObjectFlows* in the Activity are transformed into "assign" actions, which copy the content of the output variable of one invocation to the input variable of a following invocation. The sequence in which the BPEL actions occur is mainly determined by the *ObjectFlows*. However, since UML Activity Diagrams are based on graphs, whereas BPEL is structured in blocks, the generation of the BPEL processes has certain limitations [33] that we are aware of.

F. Implementing the Web Services Artifacts

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 [15]). The choice of the programming language and the framework for generating and implementing the web service adapters should be depend on the legacy application, so that existing interfaces of the legacy application can be used if possible. In addition the Web service adapter must also provide the transformation into the data objects specified in the domain model. To deploy the generated BPEL process usually a deployment descriptor has to be created, which is specific to the selected BPEL engine.

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 chapter, we apply our model-driven software integration development process presented in the previous chapter to the development of a service-oriented application to visualize 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 [9]. 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 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.

The required data for generating the study progress visualization 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.

B. Creating the Domain Model "Study Progress"

Having defined the requirements, we model the domain and needed data objects for the study progress tree map, such as examination results or information about the student. We create a UML model and model the entities as *Classes*. We also add the data structure which is needed to generate the study progress tree map (Figure 9: B).

C. 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" (Figure 9: C). 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*.

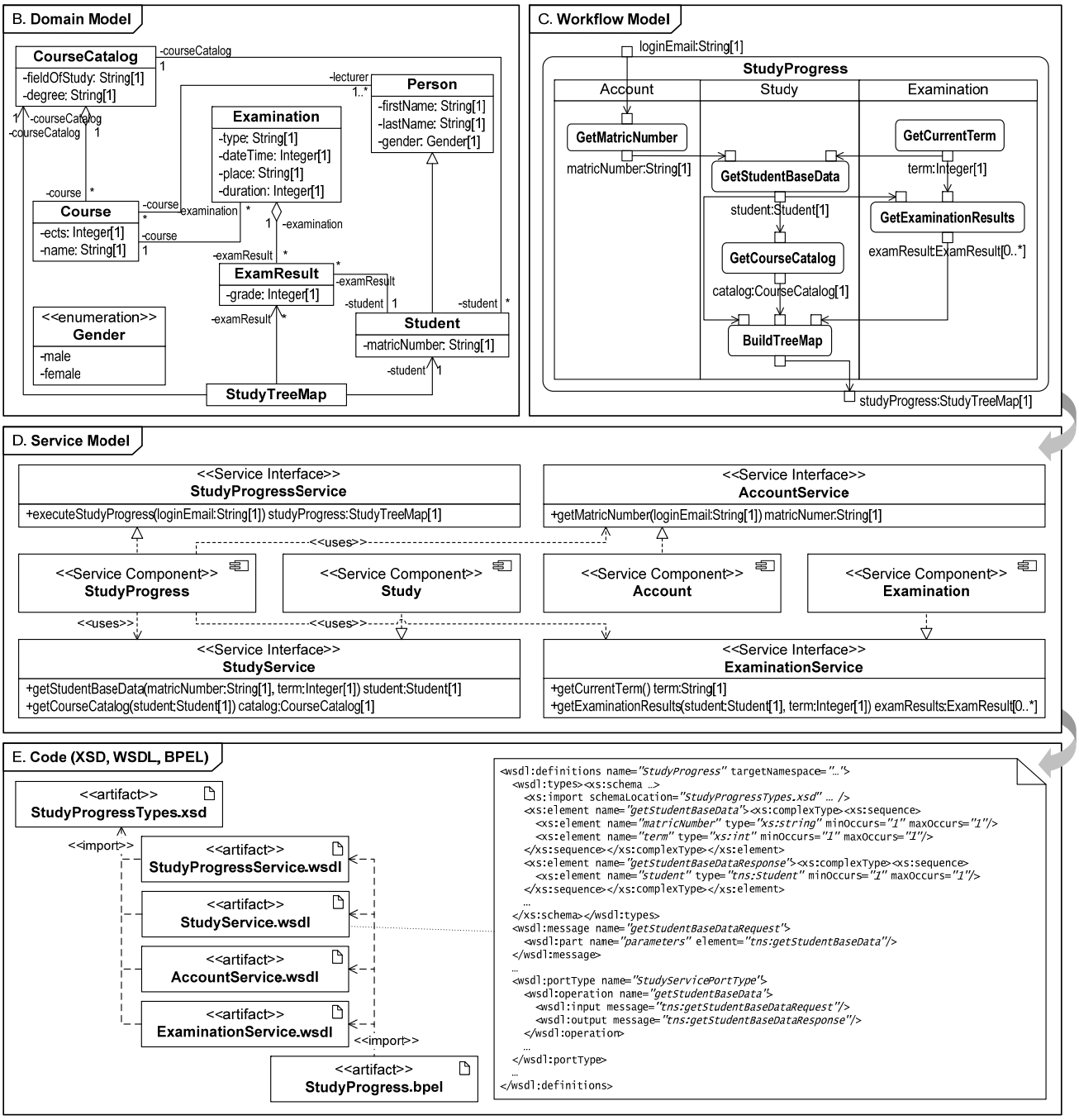

Figure 9. Model-Driven Development Process of the “Study Progress”

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

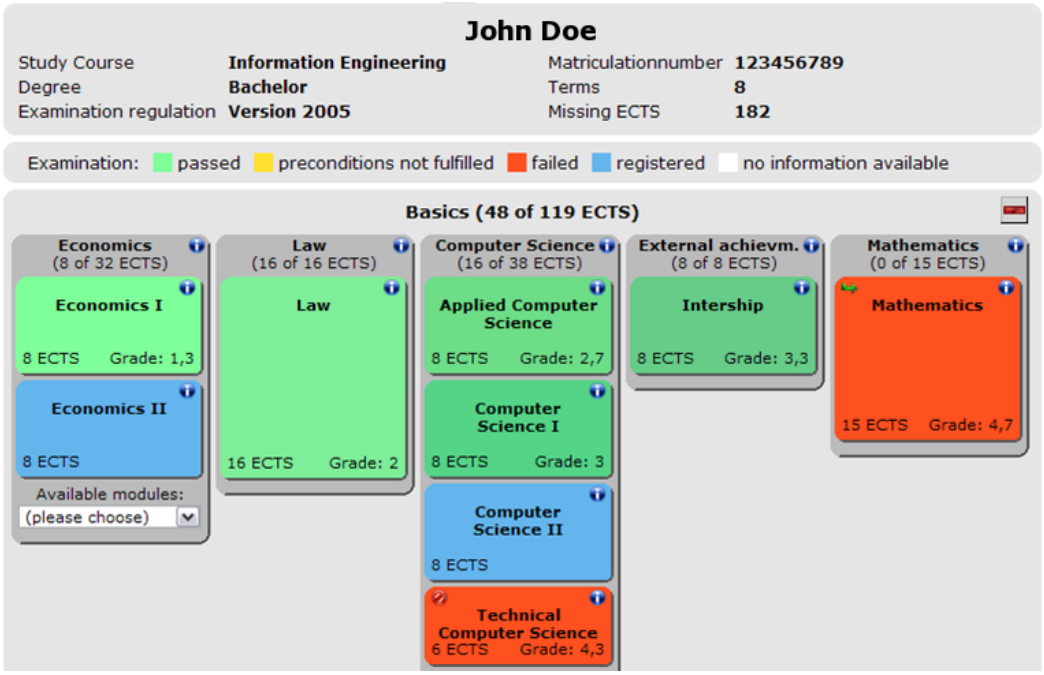

Figure 10. Screenshot of the "Study Progress"

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 how data objects flows from the outcome of a previous invocation to the input of a following invocation during the execution of the workflow. Thus the order in which invocations occur can be derived from the *ObjectFlows*.

We have formalized which applications are invoked and how the data is processed. Figure 9 shows the final workflow model as an UML Activity Diagram labeled as "Study Progress" in part C.

D. *Transformation to the Service Model*

Taking the activity diagram as a source model, we use a QVT-based model-to-model transformation to generate service interfaces using the QVT transformation rules described above. 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*.

Part D of Figure 9 shows the resulting *Interfaces*, *Components* and the relations for each *ActivityPartition* and the *Activity* itself.

E. *Transformation into Web Service Artifacts*

The model-to-text transformation creates an XML Schema file [4] ("StudyProgressTypes.xsd") from the Classes in the domain model, four WSDL files [3] (one for each service interface) from the Interfaces and Components in the service model and a BPEL file ("StudyProgress.bpel") from the workflow model. To facilitate the reusability of the XML Schema definitions the StudyProgressTypes.xsd file is imported into the "types" section of each WSDL file. Also all WSDL files are imported into the BPEL process file in order to act as partner links.

Figure 9 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.

F. *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 using Java. We further use an XSL transformation to generate XHTML from the tree map data structure defined in the domain model. Figure 10 gives the result of the engineered solution, showing a late prototype of the study process.

V. 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.

The idea of visualizing hierarchically structured information in terms of tree maps initially was published by Johnson and Shneiderman [37]. Based on their concepts, Allerding, Buck et al. present an approach using tree map concepts and focusing the requirements of students managing their studies [38]. Adapting their idea of

visualizing the study progress of a student, we used an early sketch of a tree map as input for a model-driven development approach. The execution of integration projects following a model-driven development approach based on Web service technologies is discussed in scientific and commercial communities alike.

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 [5]. 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 strictly follow a top-down development approach that would hinder 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 [6], [7], [8]. 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 [9]. 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 [9] 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. [7]. While the approach presented in [7] 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 [7], Johnson demonstrates the use of a technology-independent approach for describing service-oriented software components [10]. An UML 2.0 Profile [2] 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.

VI. CONCLUSION

In this paper, we outlined a development approach for integrating existing applications in a service-oriented manner by using a model-driven approach in order to create new functionality. In a first step GUI sketches help eliciting the requirements of the new functionality. Additionally a domain model is created statically describing the main concepts and their relationship of the domain. Afterwards, the necessary workflow, which determines the integration of the existing functionality from existing legacy applications, is modeled using UML Activity Diagrams. The workflow and the domain model are used to automatically generate artifacts which help in implementing the workflow. Such artifacts include adapters for accessing only required functionality of existing applications, relevant data types and an executable workflow for which we used Web service adapters using WSDL, data schemas described with XSD and BPEL process definitions respectively. Due to the usage of the domain model as source for the data types, the resulting services abstract from existing applications and their specific data types. This enables a wider usage of the created Web services without knowledge about platform specific details.

To exemplify our approach, we demonstrated our approach by realizing a study progress visualization at the Karlsruhe Institute of Technology (KIT). In this scenario, the goal was to provide a new feature that allows students to gain insight into their current study progress, by combining and visualizing their data from disparate sources. The required functionality and data is shared across several existing applications, so that they need to be integrated.

Even though a complete role model of an integration process is not in the focus of our work we are certain that our approach is helpful to all participants of an integration project. Our approach helps IT architects with the development of new functionality that requires the integration of existing applications. Domain experts are supported by visually describing the required functionality and integration experts can use the workflow to map the requested functionality to existing applications. Additionally the workflow is used to derive implementation artifacts by using automatic transformations, which helps in avoiding transformation errors due to human interpretations. As the approach started by gathering user requirements by means of a GUI sketch, we consider our solution user-aligned and a promising enhancement of existing integration approaches.

Although the application of model-driven approaches has several advantages, such as the convenient transformation of Web service adapters, data types and running BPEL processes from formalized design models, the usage of this modern technology is hampered by lack of a complete tool support. The application of UML Activity Diagrams enables the usage of several existing UML modeling tools and allows a formalized and visual description of the workflow. While there exist mature tools in the context of UML modeling, the development of transformations is still a complex task.
The choice of using service-oriented architecture as the integration platform proved to be the right choice for our approach, as reusing the existing business logic of the legacy applications can now achieved at a level of higher abstraction. Yet, the full potential of service-orientation such as the design of services to achieve certain design characteristics, the security of data within the workflow, the interaction with human users and the management of the services were not considered within our approach so far. These aspects are motivation for further work within this context and are part of our outlook.

VII. OUTLOOK

Based on our latest research results presented before, there are several topics we want to investigate in more detail. Firstly all functional components are provided as services via Web service interfaces. Therefore these services have to follow design principles to allow for loose coupling or to achieve a certain granularity etc. So far our model-driven approach does not take these design principles into account during the transformation from workflow model to service model. Focusing on service design principles, different variations of transformation rules could be applied to achieve a set of services with different attributes like granularity etc. suitable for different scenarios [43].

Secondly human users have to interact within a process by, e.g. inserting some data or making a decision. Hence user interfaces have to be developed to enable theses human tasks (c.f. [44]). Since the information which is to be passed along by the human user is directly correlated to the domain model [45], an automated generation of user interfaces can be achieved. In the same manner it could be possible to automatically generate several adapters like e.g. database adapters which are commonly used among several processes.

As a third perspective one should think about the whole application itself. As it consists of several services being provided by likely different providers, the users like students need to have a centralized way to report errors and to start a problem solving process like e.g. ISO 20000's Incident Management Process [34]. Therefore the development process also has to take management information and management processes into account to allow for a manageable and sustainable service-oriented application [46].

Another important aspect of any service-oriented integration scenario is the consideration of security aspects. So far no information about the security requirements of the newly composed services is incorporated into the development approach. Even though the existing applications might be secured in the sense of e.g., a secure connection, the proposed approach would reduce it to a secure point-to-point communication between the service adapter and the application instead of providing a secure end-to-end communication. A solution to this would be to add additional information concerning security requirements into the modeled process. Furthermore it can be seen from the case study, that a user might have different identifiers to access different applications and services in an organization again using different authentication mechanisms. In the current version of our approach this leads to a significant amount of operation invocation to map the initial identifier to every necessary identifier. A better approach would be to use a global identifier at the service level and enhance the model-to-text transformation presented to generate necessary mapping code directly into the Web service adapters.

We have already presented some initial results on these topics in [39, 40, 41, 42] and will now focus on integrating our findings to our model-driven approach.

REFERENCES

[1] Hoyer P., Gebhart M., Pansa I., Link S., Dikanski A., Abeck S.: A Model-Driven Development Approach for Service-Oriented Integration Scenarios. First International Conferences on Advanced Service Computing (SERVICE COMPUTATION), Athens, Greece, November 2009.

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

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

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

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

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

[12] 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

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

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

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

[16] 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

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

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

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

[20] Conrad S., Haselbring W., Koschel A., Tritsch R.: Enterprise Application Integration: Grundlagen – Konzepte - Entwurfsmuster – Praxisbeispiele, Spektrum Akademischer Verlag, 2005, ISBN 3827415721.

[21] Linthicum D.: Next Generation Application Integration, Addison-Wesley Information Technology Series, 2004, ISBN 02018445667

[22] World Wide Web Consortium (W3C): SOAP Version 1.2, http://www.w3.org/TR/soap/

[23] The Object Management Group (OMG): Business Process Model and Notation (BPMN) 2.0 Beta 1, http://www.omg.org/cgi-bin/doc?dtc/09-08-14.pdf

[24] Valk R., Girault C.: Petri Nets for Systems Engineering – A Guide to Modeling, Verification, and Applications, Springer, 2001. ISBN 978-3540412175.

[25] Keller G., Nüttgens M., Scheer A.-W.: Semantische Prozessmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes, Heft 89, Januar 1992.

[26] S. Johnston, “Rational UML Profile for business modeling”, IBM Developer Works, http://www.ibm.com/developerworks/rational/library/5167.html, 2004.

[27] Borland, Borland Together, http://www.borland.com/de/products/together/index.html

[28] ikv++ technologies ag: medini QVT, http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77&lang=en

[29] SmartQVT, http://smartqvt.elibel.tm.fr/index.html.

[30] 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/Primer/wsbpel-v2.0-Primer.pdf

[31] The Object Management Group (OMG): MOF Models to Text Transformation Language V1.0, http://www.omg.org/spec/MOFM2T/1.0/PDF

[32] Eclipse Foundation: Xpand, http://wiki.eclipse.org/Xpand.

[33] Ouyang C., Dumas M., Breutel S., Hofstede A.: Translating Standard Process Models to BPEL, Advanced Information Systems Engineering, 18th International Conference, CAiSE 2006, Luxembourg, Luxembourg, June 5-9, 2006, Proceedings 2006.

[34] ISO/IEC, ISO/IEC 20000-1:2005: Information Technology – Service Management, www.iso.org, 2005.

[35] Organization for the Advancement of Structured Information Standards (OASIS): ebXML Technical Architecture Specification v1.04, http://www.ebxml.org/specs/#technical_specifications.

[36] The Workflow Management Coalition Specification: Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011),http://www.wfmc.org/standards/docs/TC011_term_glossary_v3.pdf, 1999.

[37] Johnson B., Shneiderman B: Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures, IEEE Computer Society Press, http://hcil.cs.umd.edu/trs/91-06/91-06.html, October 1991.

[38] Allerding F., Buck J., Freudenstein P., Klosek B., Höllrigl T., Juling W., Keuter B., Link S., Majer F., Maurer A., Nussbaumer M., Ried D., Schell F.: Integriertes Service-Portal zur Studienassistenz, Proceedings of the 38th GI Conference - Lecture Notes in Informatics, München, Germany, Munich, 2008.

[39] Gebhart M., Abeck S.: Rule-Based Service Modeling, The Fourth International Conference on Software Engineering Advances, ICSEA 2009, 20-25 September 2009, Porto, Portugal 2009.

[40] Link S., Hoyer P., Kopp T., Abeck S.: A Model-Driven Development Approach Focusing Human Interaction, Second International Conference on Advances in Computer-Human Interaction, ACHI 2009, February 1-7, 2009, Cancun, Mexico 2009.

[41] Scheibenberger K., Pansa I.: Modelling dependencies of IT Infrastructure elements, Proceedings of BDIM 2008, 3rd IEEE/IFIP International Workshop on Business-Driven IT Management, April 7, 2008, Salvador, Brazil 2008.

[42] Klarl H., Wolff C., Emig C.: Identity Management in Business Process Modelling: A model-driven approach, Business Services: Konzepte, Technologien, Anwendungen. 9. Internationale Tagung Wirtschaftsinformatik 25.-27. Februar 2009, Wien 2009.

[43] T. Erl, “SOA – Principles of Service Design”, Prentice Hall, 2007. ISBN 978-0-13-234482-1.

[44] IBM: WS-BPEL Extenstion for People (BPEL4PEOPLE), 2007, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf

[45] Link S.: Benutzerinteraktion in dienstorientierten Architekturen, Dissertation, 2009, http://digbib.ubka.uni-karlsruhe.de/volltexte/1000012354