Development of SOA-Based Software Systems—
an Evolutionary Programming Approach

Christian Emig, Jochen Weisser, Sebastian Abeck
Cooperation & Management, Universität Karlsruhe (TH)
{emig | weisser | abeck}@cm-tm.uka.de

Abstract

A software application has strong relationships with the business processes it supports. In the analysis phase those parts of the processes are analyzed in which the software system is applied by its future users. If an object-oriented approach is taken, the Unified Modeling Language (UML) is often used to model the relevant aspects of the business processes. In the design phase these models must be manually mapped to the business layer of the software application. The Service-Oriented Architecture (SOA) and related technologies offer a promising new approach: The business process is described in a programming language [1], i.e. a process language which can be automatically mapped to an execution language and executed by a process engine. In this article it is shown how Programming in the Large can be practically applied in a software engineering process. The Business Process Model Notation (BPMN) is used as a process programming language. A BPMN description can be mapped to the Business Process Execution Language (BPEL) which is a widely accepted standard to compose Web services.

1. Introduction

When a software application is developed the future users’ requirements for the application are the starting point for a systematic, goal-driven software engineering approach. User requirements concern the question in which tasks and for which purposes a user wants to make use of the software, whereas these tasks are part of an overall business process.

In software engineering user requirements are analyzed in the analysis phase which represents the first phase of the application development process. The results of the analysis phase provide the input for the software design phase which is followed by the implementation and test phase. These phases can be found in all the different software engineering approaches (waterfall, RUP, etc). An important goal of a systematic and efficient software engineering approach is to make sure that the results gained in each phase can be efficiently used in the next phase.

A widely accepted language to support the analysis and design phase is given by the Unified Modeling Language (UML, [2]). UML provides a specific diagram type, namely use case diagrams, to model the view on a software system from the perspective of its (future) users. A use case is a part of a business process which is supported by the software system that helps the users to carry out specific tasks. These tasks can be described as activities in UML. Thus, a use case can be refined by another UML diagram, the activity diagram. Both types of diagram, use case and activity, together contain the business logic of the software application. In the design phase the business logic will be mapped to components (e.g., a business process control and a number of use case controls [3]) of the application architecture.

Figure 1: Modeling Approaches in the Analysis and Design Phase
As shown on the left-hand side of Figure 1, the mapping of the business logic to the components in the architecture has to be done manually. This leads to the following major deficiencies:

1. Inefficiency: Many aspects of the models that have been described in the analysis phase could be directly and automatically transferred into the architecture.

2. Inconsistency: A change of a model on the analysis level or design level will lead to an inconsistency if the change is not manually propagated.

3. Inflexibility: The system is not designed to be able to react on a changing process in a flexible way.

To overcome these deficiencies a different approach to describe the processes has to be taken. Until now, UML does not provide an adequate concept to map the use case and activity diagrams to a process execution language. Although some work has been carried in this area [4], there are good reasons to choose another language to "program" the business process related aspects. The most important reasons according to [5] are:

Firstly, UML is alien to most business analysts. Secondly the language is object-oriented - it is not based on a business process centric approach. Thirdly the mapping of UML to a business process execution language is not supported.

A process language that fulfills all these requirements is the Business Process Modeling Notation BPMN [6] standardized by the BPMI.org. In our approach of business process oriented programming we use the BPMN to produce an executable process description based on the Business Process Execution Language (BPEL, [7]) which is standardized by OASIS. Figure 1 gives an overview of the approach we will describe in the next chapters in detail. Therefore, the rest of the article is organized as follows: In Chapter 2, BPMN and its (graphical) language elements are introduced by means of a simple business process. In this example process a student orders a so-called transcript of records (ToR, a special kind of report) from the university administration. In Chapter 3 we will show how the BPMN description (i.e., the process-oriented program) is mapped (i.e., compiled) to BPEL code which can be executed by a BPEL process engine. The BPEL code needs to be executed as well - we take care of this in Chapter 4. Core Web services which are not a composition of other Web services (like executable BPEL processes) mark the border of two complementary types of programming: The composition of Web services is called "Programming in the Large" while the development of a Web service is called "Programming in the Small" [8]. In the Outlook in Chapter 5, we will point out that the business process oriented programming (Programming in the Large) will not replace component-oriented and object-oriented programming (Programming in the Small) since both types of programming are complementary.

2. Programming of a Process Using BPMN

The Business Process Modeling Notation (BPMN) [6, 9] which has been created by the Business Process Management Initiative (BPMI) pursues two objectives: First the notation should be easy to understand for every role participating in the development process, beginning with the business analysts who describe the processes from the business perspective, but as well by the technical developers responsible for implementing the technology that will be used to perform these processes. The second major goal of BPMN is to reduce the gap between the business process design being in focus in the analysis phase and the process implementation being looked at in the design and implementation phase. This is ensured by setting up on an mathematically based, internal model that enables the mapping between BPMN’s graphical elements to the underlying constructs of (XML-based) executable business process languages like BPEL [7] as illustrated in Figure 1. The tight connection between analysis and implementation provides major advantages to other modeling languages, like for instance UML which takes an object-oriented approach to the modeling of applications, while BPMN follows a business process oriented approach to the modeling of IT solutions. This different focus makes UML and BPMN not competing notations but they propagate different views.

The BPMN defines both the (graphical) notation and the semantics of a so-called Business Process Diagram (BPD), which is based on flowcharting technique. The small set of core elements of a BPD comprises the so-called Flow Objects: First of all, the Activities, represented by a rounded-corner rectangle, being a generic term for work that has to be performed. Secondly the Events, which are diagrammed as a circle and just “happen” during the execution of a business process. They usually affect the process flow. Last but not least Gateway Objects represented by a diamond symbol are used to control the divergence and convergence of a sequential flow. By means of these
To ensure the intuitive understanding of a BPD, a hierarchical modeling is strongly recommended. At first, the business process is modeled at a high level where activities in the BPD usually aggregate sub-processes, which are graphically evaluated in another BPD. A [+] sign inside an activity denotes the fact that a process can be decomposed into sub-processes. The finest granularity of activities is described by tasks forming the lowest-level process in a BPD.

BPMN supports three basic types of sub-models within an end-to-end BPMN model that have a different focus on business processes:

1. Private (internal) processes that are internal to a specific organization and that form the class of classical workflow processes. BPMN uses the rectangular symbol of a Pool representing the organization’s boundary where many so-called Lanes can be included as sub-partitions to organize or categorize activities. A single private business process may be mapped to one or more BPEL documents.

2. Abstract (public) processes that take care of the interactions between a private business process and at least one participant. Only those activities that are used to communicate outside the private business process represented in its single pool, plus the appropriate flow control mechanisms, are included in the abstract process. The focus is on the message flow between the separate business organizations. The mapping between a BPMN abstract process and its BPEL equivalent is not yet specified in [6].

3. Collaboration (global) processes can be shown as two or more abstract processes communication with each other. This is the most powerful model which cannot be mapped to BPEL but to various collaboration languages such as ebXML BPSS [10] or RosettaNet [11]. The example process we will now have a look at is such a collaboration process.

The business process shown in Figure 2 is taken from the higher education area, for example a university. The business case concerns a student who needs consultation, for example after having failed an exam. In the BPD the two participants are modeled in two pools with their own internal sequence flow. The pools are connected by the exchange of messages. Both pools consist of explicitly modeled start and end events. At the right side of the diagram all those activities are grouped that the business analyst decided to be supported by computer systems. Most of the activities are quite high-level and have a more detailed representation in another BPD. This is exemplary shown by zooming into the task “Get ToR” at the bottom of Figure 2. At the granularity of activities as tasks, BPMN offers the possibility to add further attributes to the task like the information whether a task is a so-called Service Task or a User Task. Service Tasks provide some sort of service which could be a Web service or an automated application, whereas User Tasks that a user performs manually or with the assistance of a software application. Furthermore, a task can also be a sender or receiver, just sending or receiving one message to or from a different pool and processing it afterwards.

After the decision gateway where the process determined that further information is needed by the student counselor for in-depth assistance, a so-called transcript of records (ToR) is created by the supporting system within the activity called “Get ToR”. A transcript of records is a standardized aggregation of a student’s achieved results at a university. The activity “Get ToR” is marked with a [+] that holds the information that there is a decomposed view on this activity.

The bottom part of Figure 2 shows the detailed sub-model for the high-level activity “Get ToR”. It contains only Service Tasks and can therefore be directly mapped to a BPEL process definition which is executable in a BPEL engine. If we concentrate on just one pool, BPMN allows us to neglect the outer borderline. This sub-process takes care of creating a transcript of records in university environments and is used not only in the business process of student counseling but can be part in student self service operations as well. All activities in this diagram are modeled as (elementary) tasks. This whole BPD can be
mapped to a BPEL construct whilst the single activities directly map to so-called Core Web Services. This diagram has one defined start event, which is triggered by the gateway decision “Further information needed” in Figure 2, esp. a ToR. We see two bold circles which denote the possible end points in this sub-process. According to this, the process finals with either an error, for example because the student does not exist, or the regular return to the calling process, in particular a valid ToR. Inside the process there is a parallel (AND) forking and joining. The four parallelly executed tasks (Figure 2, top right) are Core Web services, for example implemented in Java.

3. Compilation of a BPMN Program Using BPEL

This section shows how to map the BPMN process depicted in Figure 2 to an executable BPEL process. This can be either done using software tools that already helped in creating the BPDs like for example the System Architect by Popkin Software but can as well be done manually using the in-depth described mapping rules in [6]. Before mapping parts of the process to executable elements, what is normally done in the implementation phase, we first want to introduce the architecture, which we are going to use for the deployment of our service-oriented and process-focused elements.

The entry to Figure 3 is the Choreography Layer, the layer where all kinds of business processes can be deployed. Above that a role-based portal for any kind of user interaction is located. Below the Choreography Layer we see a less mighty layer, where Composition can take place as well as the interaction with only one participant (no collaboration) – this is subject to being handled with a default BPEL engine. Below the Composition Layer the layer of the Core Web services resides. Core Web services either act as adaptors to wrap existing interfaces of legacy systems making them Web service capable or could be Web service interfaces of a per se service-oriented developed system.

The analysis and design can be done by using BPMN. Reaching the implementation phase a mechanism or concept is needed to generate BPEL code for the business process out of a BPMN representation in a mostly automatic way.

Before illustrating and discussing a possible concept, we take a brief look at the BPEL code of Figure 4.

![Figure 4: BPEL Code](image)

It should be mentioned that the displayed code is fragmentary. Most of the attributes, variables and assigns are omitted for better readability. Manifested by the suppressJoinFailure attribute being assigned with the value false (see Line 2 in the BPEL code), the complete error handling is also left out.

The general structure of any BPEL process looks like this: Between the process tag which is the outer element of such an XML document there exist basically three different sections. In the first section partners and partnerLinks are defined. The partnerLinks announce the involved external programs such as a clients and Web services to the BPEL process. In our example the four Web services offer access to the various databases where the ToR-relevant information is stored. In this example the client is used to transform the ToR request of a human user to a SOAP request which can be processed by the ToR process. When the BPEL ToR Web service sends back the XML ToR, the same client decodes the SOAP request and displays the received ToR to the user. These two activities of the client are reflected by the receive and reply tag discussed later. The second section contains all the variables for the process. Variables embody all messages and XML documents used in a BPEL process. These variables
can be documented in the BPD using the defined BPMN attributes. The idea is that BPMN supporting tools save these properties for automatic conversion to BPEL but do not print them in the BPD. That is why they will not be considered here further. The third section is the orchestrational logic. *Invoke* elements call external programs by using their SOAP interface. They correspond to the activities in our BPMN graph. The *switch* tag maps the gateway element of BPMN to BPEL. Sequences are sequential actions while flows correlate to the fork objects of BPMN. All tags can include many attributes which have to be filtered out of the web service description of the participating web services.

One main idea behind BPMN is to reduce the gap between the business process design and the implementation. The rest of this subchapter illustrates how the BPMN graphs can be converted to BPEL code under the precondition to maximize the automatically executable parts. The lack of available software products, which can translate BPMN to BPEL in a way so that this code can be deployed or even efficiently developed, causes a work-around.

![Figure 5: The Approach Taken To Support BPMN-Based Process Programming](image)

As illustrated in Figure 5 there are two different categories of relevant tools for this procedure: BPMN modeling tools to create the BPMN graph and BPEL design tools for managing the specific BPEL aspects and concerns. Let us begin with the ToR process (Figure 2) which was developed with a BPMN modeling tool. At this point it is possible to map the information represented in the BPMN graph manually by using the mapping rules defined in [6]. Some BPMN modeling tools like Popkin’s System Architect offer functions to generate BPEL code out of BPMN automatically. But there is one hitch. The generated code contains almost no information about the SOAP interface of the orchestrated elements in it. If a Web service for a particular element already exists this work can be done tool-supported as its Web service Description holds all the additionally needed information.

That is the point where a BPEL design tool like Oracle’s BPEL Designer comes into play. The alternative of using such a tool is to add the remaining information manually. Most of the design tools allow importing existing BPEL code which can be generated by Popkin’s System Architect as described. The BPEL code is parsed and then displayed as a graph. Unfortunately this graph is not in BPMN notation but in a proprietary one. Now the WSDL documents of the participating Web services can be imported and missing information like variables and messages can be added to the BPEL process. Furthermore functionalities for testing and validating the created BPEL process as well as a function for deploying the process to the used BPEL engine are available.

4. Execution of a BPEL Process

After developing the BPEL program as described in Chapter 3 the question arises how this XML-based program can be executed. First of all it has to be mentioned that the core Web services that are used by the BPEL program have to be deployed in advance. The BPEL process can only be executed if WSDL documents of each Web service that is used by the process are available. In most cases the Web service description of a Web service that is online and running can be retrieved by appending “?wsdl” to the Web service URL.

To execute a BPEL process a BPEL engine is needed which parses BPEL code and executes the containing instructions. Examples for existing BPEL engines are ActiveBPEL [12] by ActiveEndpoints and the Oracle BPEL Process Manager [13]. All engines have in common that the BPEL process, which has to be deployed itself as well, needs to be supplemented. Of course the general BPEL code is always the same regardless which BPEL engine is used because it is standardized. But in practice the deployable BPEL packages differ from engine to engine. For instance, an engine-specific so-called deployment descriptor is additionally needed in order to execute the process.

Furthermore, a BPEL package usually comprises either the WSDL documents for the involved Web services themselves or the respective URLs, where they can be found. At this point it should be mentioned that because BPEL needs *partnerLinkType* tags for each involved Web service that are not included in a WSDL file by default, even for remote WSDL documents (accessible via their URLs) the BPEL packages often
contain so called wrapper WSDL documents that add the partnerLinkType tag and import the original WSDL. Some engines even require more supplementary files in the packages that e.g. contain information about partners or the WSDL catalog.

5. Outlook

It is important to notice that business process oriented programming, as it was illustrated in the last chapters, is the next step in the evolution of software engineering. Thus, business process oriented programming does not replace but complements the existing approaches of programming.

Although business process oriented programming is still in its very early stage, the high potential of this evolutionary step in software engineering is obvious. Standards, such as BPMN and BPEL, are available to apply this concept to practical software problems as we have demonstrated in the paper.

6. References
