Journal of Universal Computer Science, vol. 14, no. 8 (2008), 1368-1387
submitted: 15/11/07, accepted: 25/3/08, appeared: 28/4/08 © J.UCS

A Service-oriented Process to Develop Web Applications

Fábio Zaupa
(Departamento de Informática/Universidade Estadual de Maringá, Brazil
zaupa@embusca.com.br)

Itana M. S. Gimenes
(Departamento de Informática/Universidade Estadual de Maringá, Brazil
itana@din.uem.br)

Don Cowan
(Computer System Group/The University of Waterloo, Canada
dcowan@csg.uwaterloo.ca)

Paulo Alencar
(Computer System Group/The University of Waterloo, Canada
palencar@csg.uwaterloo.ca)

Carlos J. P. Lucena
(Departamento de Informática/Puc-Rio, Brazil
lucena@inf.puc-rio.br)

Abstract: Web applications are widely disseminated, but, traditional development methods for this type of application still require a substantial amount of new modeling and programming. Current methods do not take significant advantage of reuse techniques, such as software product lines (PL). This paper presents the WIDE-PL environment focusing on its application generation process, called Application DEvelopment based on Services - ADESE. This environment is an evolution of WIDE - Waterloo Informatics Development Environment. The WIDE-PL environment supports the generation of Web applications based on the Service-oriented Architecture (SOA) and the product line approach. Our solution encompasses a general software architecture, an application generation process, and a set of mandatory and optional services. Examples of applications from two different domains were developed using ADESE to evaluate its feasibility. The results show that the process offers several advantages including an increase in reuse and an explicit separation between the services and the business process connecting those services.

Key words: Web applications, Web-based services, Product line, Business Process
Categories: D.2.m

1 Introduction

Many organizations are developing Web-based information system as they foresee the possibility of new business opportunities. Web technologies include browsers, servers, programming languages, databases and support protocols. In this context, information systems are commonly referred to as Web applications. The number of Web applications has been continuously increasing, but the quality of the applications

does not appear to be increasing proportionally. Quality attributes such as reliability, availability, efficiency and usability are not being met. The work by Casati and Shan [Casati and Shan 2000] states that the Web façade hides huge inefficiencies, manual and error-prone operations, and slow, complex, inflexible, and unmanageable systems. It is not difficult to find evidence that the quality attributes of Web applications are not being fulfilled. There are several reasons for this lack of quality, but lack of development methods that comply with current time-to-market requirements, and taking proper advantage of reuse techniques are two key ones

Web applications can be either single applications or a composition of services that can require inter-organizational cooperation. An example is a travel agency that calls services offered by organizations such as ticketing and reservations, and car rental. The service-oriented architecture (SOA) offers the technological infrastructure to allow applications to be defined as a composition of electronic services. The SOA specification defines the composition of services as a business process made of reusable and interoperable service components.

Despite the dissemination of Web applications, it can be observed that traditional methods applied to their development, such as OOHDM [Schwabe and Rossi 1998] and OOWS [Pastor et al. 2003], still demand excessive modelling and programming as they do not take much advantage of reuse. In order to tackle this issue, the Computer System Group (CSG) of the University of Waterloo proposed a group of service and control-structure frameworks to support the development of applications, called WIDE (Waterloo Informatics Development Environment) [Cowan et al. 2007]. WIDE-PL is an evolution of WIDE that follows the principles of software product lines. The objective of WIDE-PL is to support the generation of Web applications based on SOA. In WIDE-PL, the services used by the applications can be specified and composed using a specific composition language.

This paper presents WIDE-PL focusing on its application generation process, called Application DEvelopment based on Services (ADESE). This process consists of a set of activities to: (i) define the application domain; (ii) model the services based on feature models of the product line approach; (iii) instantiate the feature models; (iv) map the instantiated feature model to a corresponding implementation diagram; (v) implement the service from the implementation diagram; and, (vi) generate applications based on the defined services. ADESE uses WSDL [Christensen et al. 2001] to specify services and WS-BPEL [OASIS 2006] (the former BPEL4WS [BEA Systems et al. 2003]) to specify business processes. Two examples of applications from different domains were developed using ADESE to evaluate its feasibility. The results show that the process offers advantages regarding: reusability, explicit separation of concerns between the business process and the services, reduction of time and development costs, interoperability and maintainability.

This paper is organized as follows. Section 2 presents a view of SOA and Web architecture based on services. Section 3 presents the fundamentals of product lines used in the proposed approach. Section 4 and 5 presents WIDE-PL and the ADESE process, respectively. Section 6 presents examples of applications, from the domains Renting a video and Inventory control, developed using ADESE, and lessons learned. Section 7 presents related work and Section 8 presents the conclusions.

2 Service-oriented Architecture (SOA)

The Web can be defined as a collection of services connected by communication protocols over the Internet. SOA uses electronic services as fundamental elements to develop distributed applications [Papazoglou and Georgakopoulos 2003]. A Web service is a specific kind of electronic service. It can be considered as an open component whose main properties include: self-description, reliance on well-known standards, high interoperability, and easy access from a URL. The main standards used in Web service technology are XML [XML 2006], WSDL [Christensen et al. 2001], SOAP [SOAP, 2006] and UDDI [UDDI, 2006]. In summary, a Web service can be defined as an application that has an interface specified in WSDL, registered in a directory via a UDDI, and that interacts with its clients exchanging XML messages encapsulated in SOAP envelopes that are transported by the HTTP protocol [Fantinato et al. 2005].

Business processes are used to compose Web services. A business process is a sequence of activities designed to achieve a business goal, possibly involving multiple cooperating organizations. There are languages to specify business processes where WS-BPEL is the most common. WS-BPEL allows the specification of a business process including primitives for the description of participating services, process variables, data and control flow, and exceptions. In this paper, a Web application is composed of a set of mandatory and optional services. The business process of an application is specified in WS-BPEL.

3 The Product Line Approach

A software product line consists of a set of software systems, also called an application family, that share a set of manageable features from a well-defined market segment or mission [SEI 2006] [Clements and Northrop 2001]. The applications (or products) of the family, also called members, are developed from a set of core assets that compose the architecture of the product line. This architecture has to be generic enough to represent all members of the product line. A member of the family is generated by customizing the core asset based on member's requirements. New components can be added to the product line as it evolves.

A common representation of the product line domain is the feature model. This model allows the representation of the common and variable capabilities of the product family. The concept of features come from domain engineering [Kang 1990] and has been evolving to support the demands of software product line [Sochos et al. 2004, Czarnecki et al. 2005, van Gurp et al. 2001]. Features are usually represented by feature diagrams that are tree structures annotated with the types of features of the product line [Sochos et al. 2004, Czarnecki et al. 2005].

In this paper, the concepts of product line are applied to support the modelling of domain services of WIDE-PL. These services are configured to compose Web applications generated using the environment.

4 The WIDE-PL Environment

The Computer System Group (CSG) from the University of Waterloo has been developing technology to build Web sites since early 90s [Cowan et al. 2007]. This technology has been used to generate a series of Web sites called Community Learning Spaces. The environment used to support the application development is called WIDE (Waterloo Informatics Development Environment). WIDE consists of an articulated set of XML-based service and control-structure frameworks to support the development of Web applications. Service frameworks include one for: mapping, diagramming, reporting, database construction, content management, access control, indexing and searching, notification and agents. WIDE-PL (WIDE-Product Line) is an evolution of WIDE that has the objective of generating Web applications based on SOA using the principles of software product lines [Kang 1990, Czarnecki et al. 2005]. In WIDE-PL, the services used in Web applications are defined and composed based on a specification and composition language. WIDE-PL consists of domain services that have an associated set of data and metadata. The services are coordinated by the business process of the application. Thus, the concerns between the services and the business process are separate leading to more flexible applications as the process can change not affecting the services. WIDE-PL adopts a principle similar to the environment proposed by Ceri et al. [Ceri 2003], but offers the possibility of generating applications from customized domain services and an explicit business process.

The work presented in this paper has used the Web service technology to specify and compose services related to current standards and support infrastructure. This technology allows the specification of services in WSDL [Christensen et al. 2001] and the composition of them as business processes specified in WS-BPEL. The advantage of using WS-BPEL as the language to compose services in applications, either local or distributed, is to rely on a homogeneous technological foundation that minimizes the complexity of heterogeneous applications. However, a specific composition language can be designed for WIDE-PL which can take advantage of levels of compositional requirements. In WIDE-PL, an application can be represented as a set of high-level features [Kang 1990, Czarnecki et al. 2005] as illustrated in Figure 1.

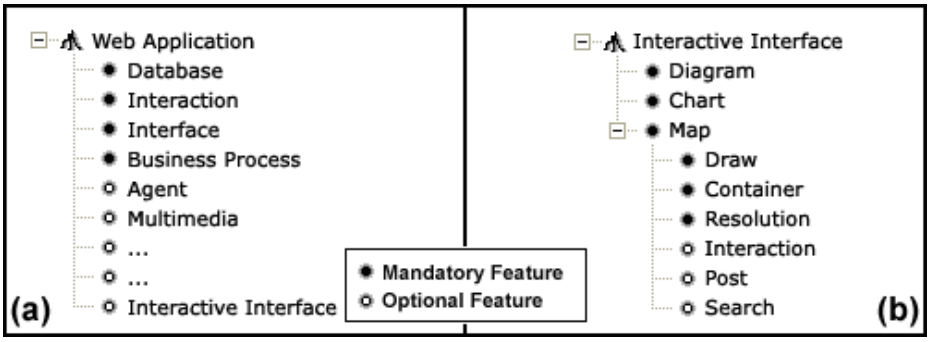

Figure 1: (a) Feature Model of Web applications in WIDE-PL. (b) Feature Model of the Domain Service Interactive Interface.

The diagrams shown in Figure 1 were produced by the tool Feature Plug-in [Antkiewicz and Czarnecki 2004]. Feature models presented in this paper follow this notation. Features are abstractions to represent the capabilities of Web applications. Each high-level feature is realized through a domain service. Each service encompasses the operations and data needed to realize the associated feature [Kang et al. 2002]. Figure 1(a) illustrates the feature model of high-level applications in WIDE-PL. A Web application has as a minimum the following services: database, interaction, interface, and business process. Thus, these features are represented as mandatory. Optional domain services can be added to WIDE-PL based on the evolution of the environment and the application requirements. Examples of domain services that can be added are: interactive interfaces, multimedia and agents. The features of each domain service can be refined in such a way that enables developers to configure the services according to the application needs. Figure 1(b) presents an example of the refinement of the Interactive Interface domain service model. This service allows the definition and execution of several types of interactive interfaces such as maps, graphics and diagrams. The Map feature is also refined to present the features Draw, Container and Resolution as mandatory as they are required in any map. It also includes as optional features Interaction, Post and Search, which can be configured as required by the application.

The following sections present the mandatory services and the logical architecture of an application generated by WIDE-PL.

4.1 Mandatory Services

The mandatory services of WIDE-PL, corresponding to the mandatory features shown in Figure 1, are those essential for the execution of Web applications. They are:

- Database service is responsible for any interaction between the business process and the application database. This service supports the creation and updating of the application database structure as well as manipulating and recovering their data.
- Interaction Service is responsible for the interaction control between the business process and the application users. It receives a page script generated by the interface service and exhibits the page to the user through a browser. It also gets user requests and forwards them to the business process service.
- Interface Service is responsible for receiving the input data and formatting the page to be exhibited to the user. This service defines the application style including fonts and colors to be used in the page. It also defines the content of the page including headers, menus and page body. The page body is prepared to receive, treat and exhibit the item lists and the form requests from the business process service. After inserting the configuration, a page script can be generated.
- Business Process Service is responsible for coordinating the execution of the application defining its behavior. The service contains: the business rules; the invocation sequence of the participating services; the actions to control exceptions; and, the definition and use of functions and variables. This is the only service that can interact with the domain services used by the application, thus any request or response must go through this service.

4.2 Logical Architecture of a WIDE-PL application

The logical architecture of applications generated by WIDE-PL is presented in Figure 2. The architecture aims at defining an explicit business process service which is separate from the domain services. Thus, an application becomes more adaptable as the business process can change without necessarily changing the services. In addition, services can be added or modified with a minimum impact on the business process.

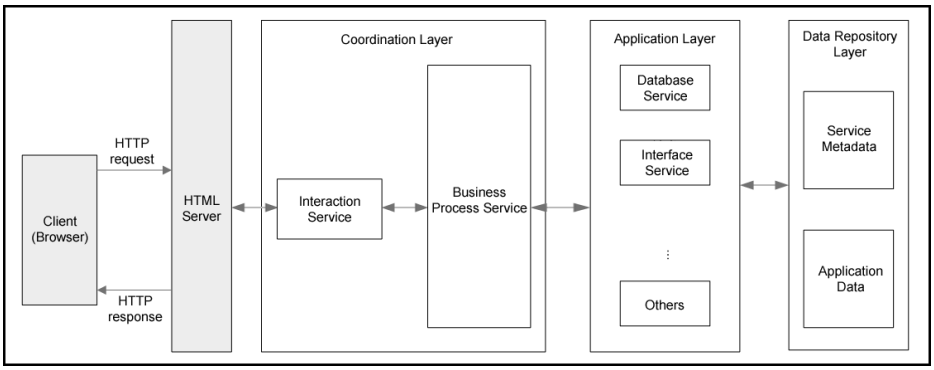

Figure 2: Overview of the logical architecture of WIDE-PL applications

The architecture is composed of three layers as follows:

- Coordination is responsible for the coordination of the business process. It contains the business process and the interaction service. An application execution starts at the coordination layer by triggering the business process service. This service executes the application control flow invoking services of the application layer. After recovering the page data the business process service invokes the interface service to generate the script of the page to be exhibited to the user. This script is then sent to the interaction service which shows it to the user through a browser. The user interacts with the interface sending requests to the interaction service that handles them and forwards them to the business process service to continue with the control flow of the application.
- Application contains the configuration of the domain services either mandatory or optional that were selected for the application. These services can access the services of the data repository layer searching for data and metadata of the application
- Data Repository contains the data and metadata that can be used by the application services.

The WIDE-PL application coordination is carried out only by the business process service, currently following the orchestration approach [Peltz 2003]. The business process service contains the execution engine and the other services have only a basic infrastructure to execute their services and to communicate with the business process.

5 ADESE Process

ADESE is the process used to develop a Web application in WIDE-PL. Figure 3 presents a graphical representation of ADESE with SADT [Ross and Schoman 1977]. The process contains 3 stages described as follows.

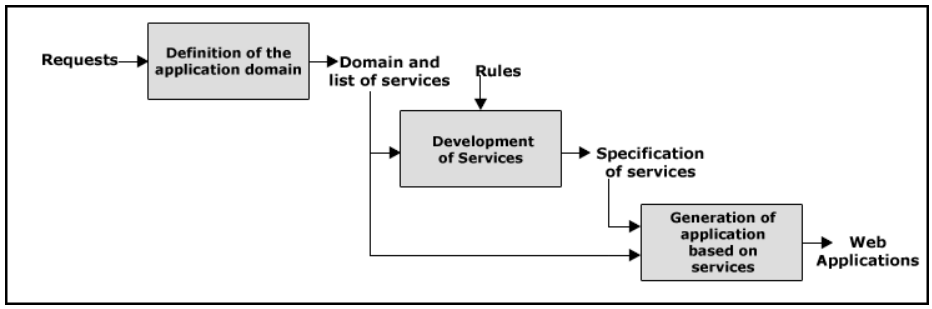

Figure 3: Stages of ADESE Process.

Stage 1 – Definition of the application domain

This stage defines the application domain based on a textual description. This description is used to undertake a domain analysis that will generate a list of domain services needed for the Web application development, and the respective feature models. These services can be either mandatory or optional. For instance, a travel agency domain might list services for air ticket reservation, car reservation, as well as basic service such as calendars and weather forecasting. The feature model represents these services defining their mandatory or optional features.

Stage 2 – Development of services

This stage is developed when a new service is required for a Web application. If the required service is available in WIDE-PL, the developer can proceed to the next step. In order to develop a new service it is necessary to:

- Define the feature model of the service consists of capturing the mandatory and optional features and their attributes.
- Instantiate the feature model consists of selecting a configuration of the feature model, which contains the mandatory and optional features selected as appropriate for the Web application.
- Map the feature model to an implementation model consists of mapping the feature model to a corresponding implementation model identifying the classes, relationships, attributes and operations.
- Implement the service from the implementation model consists of three steps: (i) implement the code of the implementation model; (ii) map the implementation model to services; and, (iii) specify the interface of the service.

The mapping of the feature model to the implementation model defines the physical infrastructure to implement the services. We have mapped the feature model to a class diagram. This has dealt with the mapping problem at a low level of granularity. However, we recognize that further research is needed to address frameworks of higher granularity levels.

The rules defined to map feature models to class diagram are as follows:

- Identify classes. The feature diagram is a tree, thus the first step creates a class for each feature of the model configuration which is not a leaf.
- Identify relationships. Add an aggregation relationship between classes that have a relationship in the feature model. The aggregating class is the one, which is at a higher level than the referred relationships in the feature model.
- Identify operations. Transform the leaf features into operations of the class that represent the respective feature.
- Identify the operations of the main class. The root feature of the feature model is the main class of the class diagram. The operations included in this class will be mapped to service operations. Thus, if any other class contains operations that will be a service operation; they have to be included in the root class to act as interface to the operations of the other classes.
- Include additional support operations and class attributes. Additional operation and attributes such as getters and setters and class attributes are included.

Stage 3 – Generation of the application based on the defined services

This stage aims at generating the Web application based on services according to the selected domain defined in Stage 1. The activities of this stage are:

- Creation of a synchronous business process consists of creating the basic control infrastructure of the application. This infrastructure contains the startup definitions of the process, variables, partner services, and an empty definition for the process flow. As a synchronous process, its execution starts from the first operation of the process and ends after its control flow is completed.
- Service addition consists of adding the necessary services to the business process of the application. The steps to add services are: (i) import the specification of each service necessary for the application to the development tool selected by the developer; (ii) configure the specification of each service necessary to the application generation - a WSDL extension has to be added to the service code in order to facilitate the communication between the WS-BPEL process and the respective services; and, (iii) add the configured service to the business process.
- Business process specification consists of completing the specification of the business process with the application control flow according to the rules of the process language, the rules of the business domain and the execution rules that establish the sequence of the service invocations according to the WIDE-PL logical architecture.

6 Examples of applications developed using ADESE

This section presents an example of the use of the ADESE process developed to evaluate its feasibility. This example consists of two independent Web applications originating in different application domains. One application is from the domain of renting a video and the other one from the inventory control domain. The applications use the same mandatory services, but their respective business processes are specified according to the specific business rules of each domain. In addition, these applications have optional services that are specific for each domain to illustrate the development of additional services in the WIDE-PL environment. Simple services were used to illustrate the application of ADESE process; however, we notice that more complex services will only require a more complex implementation, which is carried out only once, when the service is added to WIDE-PL.

The tools used to support the example process applications are: (i) Netbeans IDE 5.5 [Netbeans 2006] to codify, compile and execute the services and the business process; (ii) Feature Plug-in [Antkiewicz and Czarnecki 2004] to specify the feature model; and, (iii) MySQL [Mysql 2006] to store and recover application data.

6.1 The Video Renting Application

In this section, we explain the application of the stages of ADESE process to develop the application related to renting a video.

Renting a Video: stage 1 – Definition of the application domain

The application should control the process of renting videos including: registering clients, films and media; reservation; location, payment and access control. The development of the application requires the mandatory services available in the WIDE-PL environment plus a `Calendar` service which is optional. This service is responsible for providing information including: current date and time; differences between dates, and adding days to a certain date.

Renting a Video: stage 2 – Development of services

In this stage, we present the development of the optional service `Calendar` in order to illustrate the development of a service not initially available in WIDE-PL. Thus, the steps followed are: define the feature model, instantiate it, map it to a class diagram and implement it. Figure 4(a) presents the feature model obtained for the `Calendar` service. The root feature of this model is `Calendar`. The other features are divided into two groups `Date` and `Time`. The `Date` feature includes `Today`, `Day`, `Month` and `Year`; and the `Time` feature includes `Now`, `Hour` and `Minute`. The sub-features `Today` and `Now` are mandatory whereas the others need to be instantiated based on the application requirements. In this case, all optional features are selected.

The instantiated feature model is then mapped to its corresponding class diagram, according to the rules defined in section 5 (Stage 2). The class diagram obtained for `Calendar` is presented in Figure 4(b).

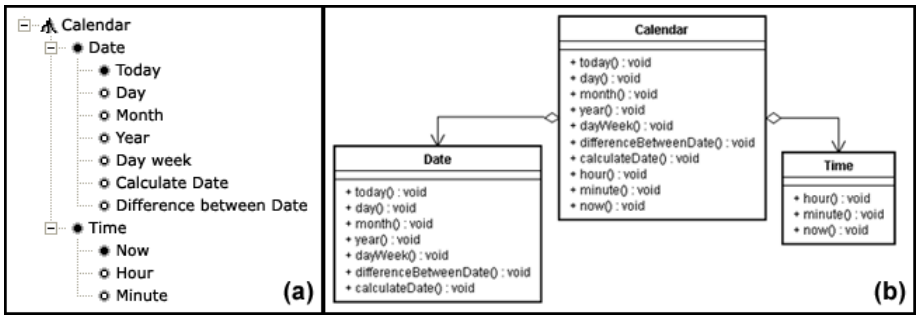

Figure 4: (a) Feature model of the Calendar service. (b) Respective class diagram.

The rules applied consist of identifying classes, relationships, attributes and operations of the diagram as follows:

- Class identification: `Calendar`, `Date` and `Time` were added as classes to the class diagram because they are not leaves of the feature diagram.
- Relationship identification: The aggregation relationships between `Calendar` and `Date;` and, `Calendar` and `Time` were added as they are corresponding relationships in the feature diagram. `Calendar` is the aggregating class as it is a high level feature in the feature diagram.
- Identification of operations: Operations corresponding to the respective leaves of the feature diagram were added to the classes already in the class diagram. Thus, the operations: `today`, `day`, `month`, `year`, `dayWeek`, `differenceBetweenDate` and `calculateDate` were added to the class `Date`. Similarly, the operations `now`, `hour` and `minute` were added to the class `Hour`.
- Identification of the operations of the main class: The main class of the class diagram is `Calendar`, thus its operations will become service operations. These operations are: `today`, `day`, `month`, `year`, `dayWeek`, `differenceBetweenDate`, `calculateDate`, `now`, `hour` and `minute`.

The next step is to generate the service from the class diagram by coding and compiling the classes. This step is carried out manually by the developer supported by a development tool, which in this example is Netbeans. In what follows, the classes are packed into a service. This also depends on the support tool.

Figure 5 shows the structure created by Netbeans for the `Calendar` service. After developing the `Calendar` service, it is necessary to specify its interface in WSDL. Netbeans supports the automatic generation of this interface.

Renting a Video: stage 3 - Generation of the application based on the defined services

This stage presents the activities needed to generate the Web application according to section 5 (Stage 3) illustrated with segments of WS-BPEL specifications.

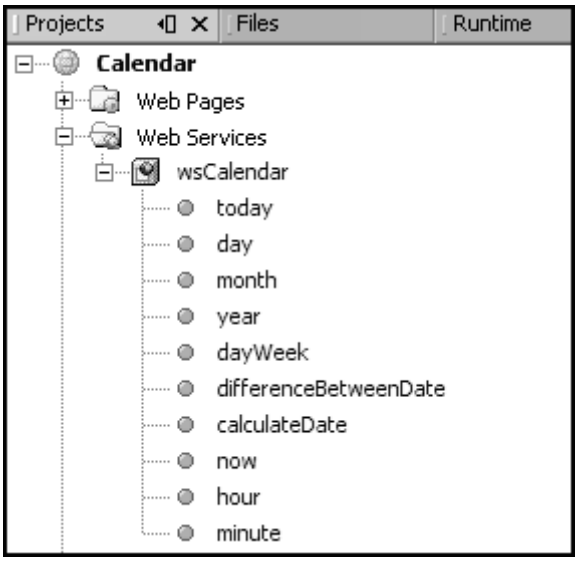

Figure 5: Structure of the Calendar service generated by Netbeans.

Figure 6 represents the overall business process of the Renting a Video application at the end of this stage, based on the graphical representation of Netbeans. Parts of this figure are explained in more detail in the following paragraphs.

Creation of a Synchronous Business Process

The first activity is the creation of a synchronous process corresponding to the mandatory service that represents the Business Process service of WIDE-PL. The basic infrastructure of the Business Process first generated is presented in Figure 6(a). The process created, named `BusinessProcess`, contains only one operation, named `operation1`, which aims at starting up the Renting a Video application. This basic infrastructure contains a WS-BPEL description including initial variables, service partners and an empty process flow. The business rules are inserted between start and end.

Service addition

The addition of required services to the initial business process consists of incorporating the customized services to compose the application. The services needed are then copied to the development tool to be configured. This configuration consists of adding to the service an extension of the WSDL language in order to establish the communication between the WS-BPEL business process and the service. The final step is to add the configured service to the business process. Figure 6(b) shows the mandatory services whereas Figure 6(c) shows the optional ones with their respective operations.

Code 1 presents the WS-BPEL specification corresponding to the process shown in Figure 6. The partners shown in this code are the services used by the application.

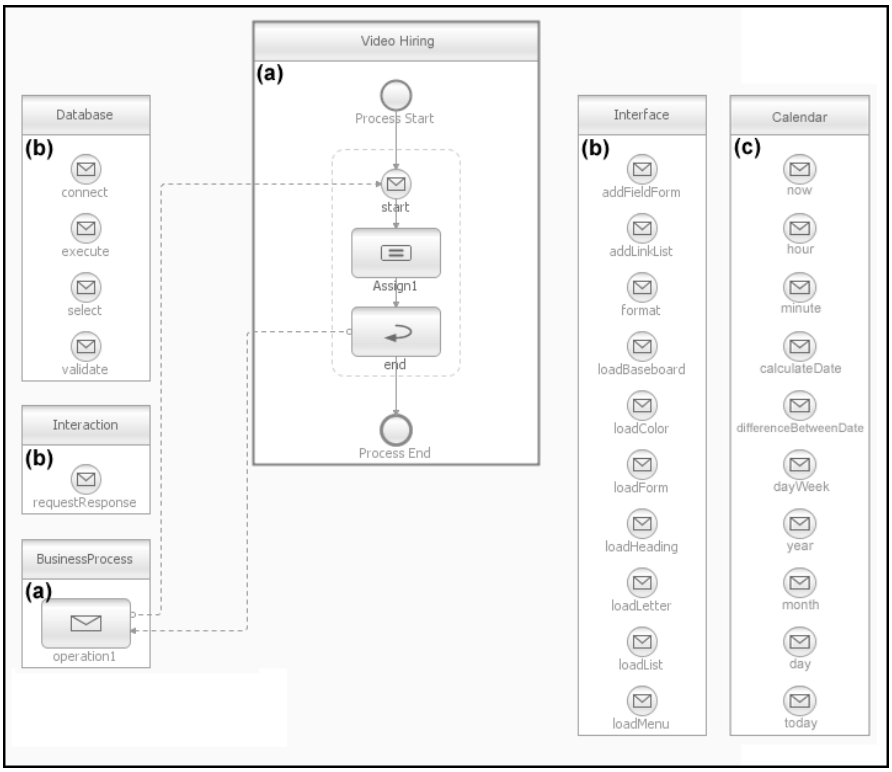

Figure 6: Renting a Video application with services added.

```
<partnerLinks>
    <partnerLink name="BusinessProcess" partnerLinkType="ns1:partnerlinktype1"
myRole="partnerlinktyperole1"/>
    <partnerLink name="Database" partnerLinkType="ns2:Database"
partnerRole="wsDatabaseSEILinkType"/>
    <partnerLink name="Interface" partnerLinkType="ns3:Interface"
partnerRole="wsInterfaceSEILinkType"/>
    <partnerLink name="Interaction" partnerLinkType="ns4:Interaction"
partnerRole="wsInteractionSEILinkType"/>
    <partnerLink name="Calendar" partnerLinkType="ns5:wsCalendarSEILinkType"
partnerRole="wsCalendarSEIRole"/>
</partnerLinks>
```

Code 1: Structure of Renting a Video in WS-BPEL.

Specification of the business process

The business process specification consists of defining the application flow in WS-BPEL according to the business process rules of the application. This specification establishes the sequence of service invocations as defined in the WIDE-PL logical architecture. Initially the variables are declared and values are assigned to the

elements used throughout the application execution such as database connection values, report headers and footers, menus and page style.

Code 2 presents a WS-BPEL excerpt in which values are assigned to the variables `login`, `password`, `localhost` and `drive`. Further, in the operation `connect` the database service is invoked in order to send the variable instances as parameters to validate the database connection.

```
<assign name="Assign1">
   <copy><from>string('')</from><to>$ConnectIn1.parameters/S1</to></copy>
   <copy><from>string('')</from><to>$ConnectIn1.parameters/S2</to></copy>
   <copy><from>string('jdbc:mysql://localhost/m')</from><to>$ConnectIn1.parameters/S3</to></copy>
   <copy><from>string('com.mysql.jdbc.Driver')</from><to>$ConnectIn1.parameters/S4</to></copy>
</assign>
<invoke name="ConnectDB" partnerLink="Database" operation="connect"
portType="ns2:wsDatabaseSEI" inputVariable="ConnectIn1" outputVariable="ConnectOut1"/>
```

Code 2: BPEL excerpt that invokes the operation connect to the database service.

The body of a Web page should be updated with information related to the user's interaction. Thus, it is necessary to send this information to the Interface service. After that action, the operation `format` formats the data and generates the script of the page to be invoked. This script is returned to the Business Process service that forwards it to the Interaction service to show the page to the user. Code 3 illustrates this scenario.

```
<invoke name="Page1" partnerLink="Interface" operation="format" portType="ns2:wsInterfaceSEI"
 inputVariable="FormatFalse" outputVariable="Page"/>
<assign name="Ass3">
   <copy><from>$Page.result/result</from><to>$ResultInteraction.result/result</to></copy>
</assign>
<invoke name="Interaction1" partnerLink="Interaction" operation="requestResponse"
portType="ns3:wsInteractionSEI" inputVariable="PageInteraction" outputVariable="ResultInteraction"/>
```

Code 3: WS-BPEL excerpt that formats a Web page and sends it to the interaction service.

Interactions with the users are handled and forwarded to the business process to continue its flow. Similar procedures are carried out for each application function such as register, locate, pay and reserve films. The cases that recover data from the database service require an additional invocation to obtain the data and send them to the Interface service. The application ends when the business process reaches the end of the flow.

6.2 The Inventory Control Application

This section presents the activities needed to generate the Inventory control application. Figure 7 shows a graphical representation of the business process and services for this application. This Figure contains the mandatory services plus the

`Basic mathematic` service. In the following paragraphs, parts of this figure are explained and excerpts of the code are presented.

Inventory control: stage 1 – Definition of the application domain

The inventory application controls product buying and selling in a company. It includes functions for: client and provider registry, product catalog, shopping control, and user access control. The application requires a `Basic mathematic` service in addition to the mandatory ones. This service is used to update the available product quantity after shopping occurs.

Inventory control: stage 2 – Development of services

The inventory control application requires the development of the `Basic Mathematic` service following the same steps as those applied to the Renting a Video services. The feature model of this service has the sub-features: `Add`, `Subtract`, `Multiply` and `Divide` as basic operations. Figure 7(c) shows the corresponding structure of this service.

Inventory control: stage 3 - Generation of the application based on the defined services

The creation of the business process and addition of services to the inventory control application followed the same steps as the Renting a Video application. The difference between applications developed based on the ADESE process is in the business process rules. Each application will have to configure colors, fonts, database connection and so on. The WIDE-PL logical architecture provides a uniform structure that facilitates reuse of services. Code 4 presents a WS-BPEL excerpt of the business process of the inventory control application where values are assigned to variables that must be presented in the header, footer and menu of the application.

```
<assign name="Assign1">
...
   <copy><from>string('Inventory Control')</from><to>$LoadHeadingIn1.parameters/String_1</to></copy>
   <copy><from>string('... Web application for work of Masters ...')</from>
   <to>$LoadBaseboardIn1.parameters/String_1</to></copy>
   <copy><from>string('Client;op=1//Supplier;op=2//Product;op=3//Buy;op=4//Sale;op=5//Quit;op=9')</from>
   <to>$LoadMenuIn1.parameters/String_1</to></copy>
...
</assign>
<invoke name="LoadHeading" partnerLink="Interface" operation="loadHeading"
inputVariable="LoadHeadingIn1" portType="ns2:wsInterfaceSEI" />
<invoke name="LoadBaseboard" partnerLink="Interface" operation="loadBaseboard"
inputVariable="LoadBaseboardIn1" portType="ns2:wsInterfaceSEI" />
<invoke name="LoadMenu" partnerLink="Interface" operation="loadMenu"
inputVariable="LoadMenuIn1" portType="ns2:wsInterfaceSEI" />
```

Code 4: WS-BPEL excerpt that assigns values to variables of the header, footer and menu used as parameters of the Interface service.

The variable instances are further sent as parameters to the `Interface` service in order to configure the service (e.g. `partnerLink="Interface" operation="loadHeading" inputVariable="LoadHeadingIn1"`).

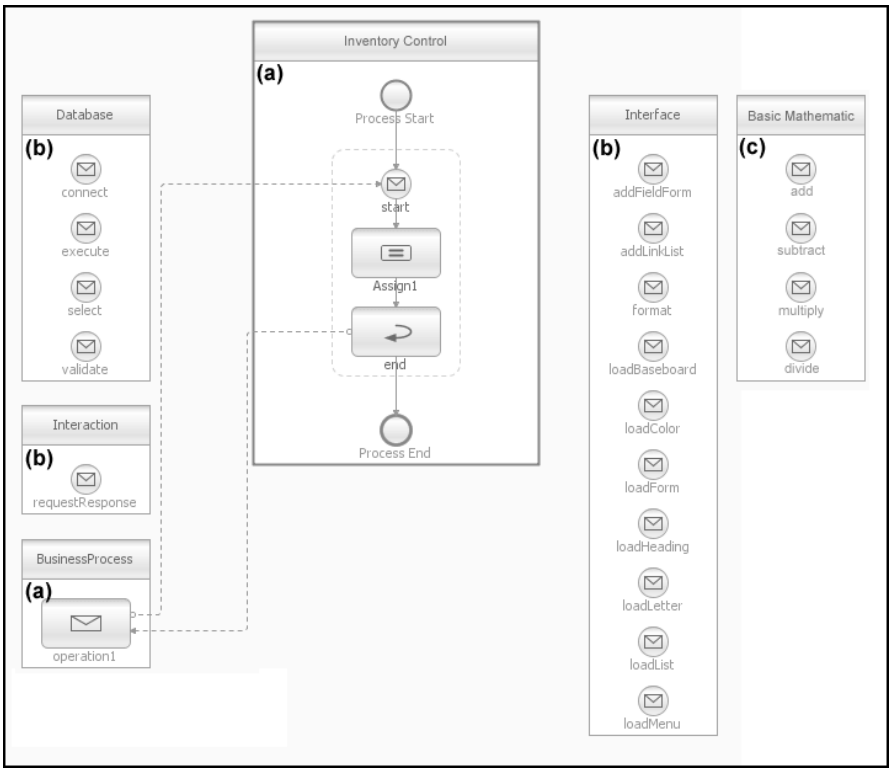

Figure 7: Business process and service for the inventory control application.

6.3 Application execution

A WS-BPEL engine is necessary to complete the execution of the example and a number of these engines, such as Netbeans 5.5 [Netbeans 2006], Bexee [BEXEE 2007] and PXE [PXE 2007], were tested in the context of this work. However, at the time of the development of the examples, we have opted for simulating the process execution in a Java servlet from which the services were called following the sequence of the WS-BPEL business process. The Java servlets of the applications were activated by referring to their addresses in the browser. Code 5 presents an excerpt of the servlet that simulates the Renting a Video application. More recent solutions allow an automatic translation from WS-BPEL to Java [B2J 2007]. Currently our research group is using Active BPEL [Active Endpoints 2007].

```
// Set web application
try {
   database.WsDatabase wsDatabase = new database.WsDatabase_Impl();
   database.WsDatabaseSEI _wsDatabaseSEIPort = wsDatabase.getWsDatabaseSEIPort();
   _wsDatabaseSEIPort.connect("","","jdbc:mysql://localhost/m","com.mysql.jdbc.Driver");
} catch(Exception ex) {
}
try {
   interfacew.WsInterface wsInterface = new interfacew.WsInterface_Impl();
   interfacew.WsInterfaceSEI _wsInterfaceSEIPort = wsInterface.getWsInterfaceSEIPort();
   _wsInterfaceSEIPort.loadColor("#FFFFFF","#9EEB95","#DDDDDD","#9EEB95","#EEEEEE","#000000","#FF0000");
   _wsInterfaceSEIPort.loadHeading("Video Hiring");
   _wsInterfaceSEIPort.loadBaseboard("... Web application for work of Masters ...");
   _wsInterfaceSEIPort.loadMenu("Client;op=1//Film;op=2//Tape;op=3//Lease;op=4//Payment;op=5//
   Reservation;op=6//Quit;op=9");
   _wsInterfaceSEIPort.addFieldForm("TEXT","E-mail","email","","");
   _wsInterfaceSEIPort.addFieldForm("PASSWORD","Password","password","","");
   _wsInterfaceSEIPort.loadForm("Access","");
   page=_wsInterfaceSEIPort.format(false);
} catch(Exception ex) {
}

// Loop until access
while (!access){
   try {
      interaction.WsInteraction wsInteraction = new interaction.WsInteraction_Impl();
      interaction.WsInteractionSEI _wsInteractionSEIPort = wsInteraction.getWsInteractionSEIPort();
      result=_wsInteractionSEIPort.requestResponse(page);
   } catch(Exception ex) {
   }
   try {
      database.WsDatabase wsDatabase = new database.WsDatabase_Impl();
      database.WsDatabaseSEI _wsDatabaseSEIPort = wsDatabase.getWsDatabaseSEIPort();
      access=_wsDatabaseSEIPort.validate("select * from inventory_user where email=\""+result[0]+"\" and
      password=\""+result[1]+"\"");
   } catch(Exception ex) {
   }
}...
```

Code 5: Excerpt of the servlet that simulates the WS-BPEL business process.

6.4 Lessons learned

A qualitative analysis of the application examples comparing some aspects of the ADESE process to the traditional methods of Web application development was undertaken.

- Reuse: ADESE brings benefits because mandatory services are available in WIDE-PL and new ones, once developed for an application become available for future applications.
- Explicit separation of concerns between the business logic of the application and the services: The approach separates the business logic of the application from the functions needed to implement the application.
- Reduction of time and development costs: As the services of WIDE-PL evolve, the major development effort concentrates on the business logic of

each new application domain. The services only need to be configured and added to the business process. This reuse of services reduces much of the necessary modeling and programming needed to generate an application.

Other examples of potential benefits of ADESE are: interoperability as functionality provided in different languages and platforms can be incorporated into applications as services; ease of maintainability as there are few dependencies between services. Examples of possible disadvantages are: (i) performance because services uses XML as a standard format for message exchange, which produces more complex formats than their equivalent binary ones; (ii) security as the risk of data interception increases due to the exchange of messages throughout the Web; and, (iii) availability because the application will be using services distributed across the Web which increases the risk of failure to find an available service.

7 Related Work

There are works related to our approach both in the area of methods for the development of Web applications and PL. There are well-known methods for the development of Web applications such as OOHDM [Schwabe and Rossi 1998], WebML [Ceri et al. 2003] and OOWS [Pastor et al. 2003]. These methods are based on the creation of conceptual models that are used as a basis for code generation. WebML has a comprehensive support tool called WebRatio [WebRatio 2007]. By using this tool one can rapidly produce a data intensive Web application. For each application model, the tool produces an executable code. The approach proposed here enhances WIDE [Cowan et al 2007], an environment which is consistent with these methods. However, we noticed that the introduction of PL concepts could improve reuse of Web applications, thus reducing the amount of modeling and programming currently required. In addition, we aimed at making the business process part of the application explicit. This was carried out using a service-oriented approach. Rossi et al. [2003] introduce a design model for treating processes as first class citizens during Web application modeling and design. Knap et al. [2004] introduces the modeling of business processes based on UML activity diagrams. However, these works did not consider a service-oriented architecture. Recent works by Brambilla et al. [2006] and Giner [2007] are also concerned with making the business process explicit in Web applications. Brambilla et al. [2006] introduces the modeling of business processes using BPMN [Owen and Raj 2003] which is than related to WebML models. Giner [2007] also uses WS-BPEL for modeling the business process. Neither of these works adopts a feature-based approach to model services as it is proposed for WIDE-PL.

Koriandol [Balzerani et al. 2005] is a PL based on FODA which aims at supporting the development of Web applications. However, it does not consider a service-oriented approach. Recent research projects in the area of PL and domain-driven approaches are consistent with our work. Gruler et al. [2007] propose an approach to model services and compose application from their configuration. Lalanda e Marin [2007] proposes a domain-configurable development environment for service-oriented application. However, these works are not directly related to the development of Web applications.

8 Conclusions

This paper presents the ADESE process for the development of Web applications in the context of WIDE-PL. The contributions include the process definition and specification, in addition to the evolution of the WIDE-PL environment. The process includes stages to develop the services and the rules to map feature models to an implementation model. There is still a need for solutions to translate feature models to lower level models. An example is the approach proposed by [Czarnecki and Antkiewicz 2005], which focuses on translating feature models to UML activity diagrams. We have also shown how to translate feature models to class diagrams. However, more work is needed to support translation of higher granularity feature models to class frameworks, thus facilitating the development of domain services.

The results of this work illustrate advantages of the proposed approach over traditional development methods. The main advantage is the explicit separation of concerns between the business logic of the application and the services logic, thus allowing Web applications to be developed and evolved in a simpler manner. The advantages also include: (i) reusability; (ii) reduction of cost and development time; (iii) interoperability; and, (iv) maintainability. The ADESE process shows that it is possible to achieve a more automated development of Web application based on services, thus it promotes a new approach where the focus is on the business process, and services are added based on the business rules of the application. Some disadvantages were observed owing to the current state of technology such as the efficiency of WS-BPEL engines as well as performance, security and availability. It was also observed that a lower-level composition language is necessary to enable the composition of lower-level services to higher-level ones before making them available as WSDL specifications to be invoke from WS-BPEL. This language would further increase reusability.

Future work includes: (i) extending the operations of the mandatory services as we have only implemented the ones necessary to carry out the case study; (ii) specifying and implementing more optional services to enrich WIDE-PL; (iii) developing an extension of WS-BPEL that supports better human interaction throughout the execution of the business process; (iv) undertaking experimental studies to collect data that support quantitative studies; (v) extending the mapping from the feature model to frameworks of higher granularity; and, (vi) studying alternative compositional languages.

References

[Active Endpoints 2007] Active Endpoints – ActiveBPEL Designer for SOA Orchestration, http://www.active-endpoints.com/, 2007.

[Antkiewicz and Czarnacki 2004] Antkiewicz, M., Czarnacki K.: “FeaturePlugin: Feature Modeling Plug-in for Eclipse”, In Eclipse '04: Proceedings of the OOPSLA, Canada, ACM Press (2004).

[B2J 2007] BPEL to Java Subproject Eclipse: http://www.eclipse.org/stp/b2j, March 2007.

[BEA Systems et al. 2003] BEA Systems, IBM, Microsoft, SAP AG, Siebel Systems: “Business Process Execution Language for Web Services Version 1.1” (2003).

[BEXEE 2007] Bexee BPEL Execution Engine: http://sourceforge.net/projects/bexee, January 2007.

[Balzerani et al. 2005] Balzerani, L., Di Ruscio, D., Pierantonio, A, A Product Line Architecture for Web Applications, SAC '05, March , 2005, Santa Fe, New Mexico, USA.

[Brambilla et al. 2006] Brambilla, M., Ceri, S., Fraternali, P., Process Modeling in Web Applications, ACM Transactions on Software Engineering and Methodology, Vol. 15, No. 04, October 2006, pp. 360-409.

[Casati and Shan 2000] Casati, F., Shan, M.: "Process Automation as the Foundation for E-Business", In: Proceedings of 26th International Conference on Very Large Databases, Egypt (2000).

[Ceri et al. 2003] Ceri, S., Fraternali, P., Arcebis, L., Bongio, S., Butti, S., Ciapessoni, F., Conserva, C., Elli, R., Greppi, C., Tagliasacchi, M., Toffetti, G. "Architectural Issues and Solutions in the Development of Data-Intensive Web Applications", Proc. of CIDR'03, USA (2003).

[Christensen et al. 2001] Christensen, E., Curbera, F., Meredith, Weerawarana, Sanjiva: "Web Services Description Language (WSDL) 1.1", W3C Note 15 (2001).

[Clements and Northrop 2001] Clements, P., Northrop, L.: "Software product lines: practices and patterns", 1ed, Boston: Addison-Wesley, p. 608 (2001).

[Cowan et al. 2007] Cowan, D. D., Alencar, P. S., Brown, D. B., Convey, H. D., Gimenes, I., Lucena, C. J., Malyk, W. J., Mulholland, D. W., Robins, A., Young, K. "Software System Generation from an Enterprise Service Model" David R. Cheriton School of Computer Science Report (CS-2007-04) http://www.cs.uwaterloo.ca/research/tr/2007/

[Czarnecki et al. 2005] Czarnecki, K., Helzen, S., Eisenecker, U.: "Staged configuration through specialization and multi-level configuration of feature models. To appear in special issue on Software Variability: Process and Management, Software Process Improvement and Practice", 10(2) (2005).

[Czarnecki and Antkiewicz 2005] Czarnecki, K., Antkiewicz, M.: "Mapping features to models: A template approach based on superimposed variants", GPCE 2005, v. 3676, p. 422, Springer (2005).

[Fantinato et al. 2005] Fantinato, M., Toledo, M., Gimenes, I.: "Arquitetura de Sistemas de Gerenciamento de Processos de Negócio Baseado em Serviços", Technical Report IC-05-06, Unicamp, Brazil (2005). Available at http://www.ic.unicamp.br/reltec-ftp/2005/Titles.html

[Giner et al. 2007] Giner, P., Torres, V., Pelechano, V., Bridging the Gap between BPMN and WS-BPEL. M2M Transformations in Practice, Model-Driven Web Engineering (MDWE 2007).

[Gruler et al. 2007] Gruler, A., Harhurin, A., Hartmann, J. "Development and Configuration of Service-based Product Lines", *Proceedings of 11th International Software Product Line Conference*, IEEE Computer Society, Washington, 2007, pp. 107-116.

[Kang 1990] Kang, K.: "Feature-oriented domain analysis (FODA) - feasibility study", Technical Report CMU/SEI-90-TR-21, SEI/CMU, Pittsburgh (1990).

[Kang et al. 2002] Kang, K., Lee, J., Donohoe, P.: "Feature-oriented Product Line Engineering", IEEE Software (2002).

[Knap et al. 2004] Knapp, A., Koch, N., Zhang, G., Hassler, H-M, Modeling Business Process in Web applications with ArgoUWE, LNCS 3273, Springer-Verlag 2004.

[Lalanda and Marin 2007] Lalanda, P., Marin, C. "A Domain-configurable Development Environment for Service-oriented Applications", *IEEE Software 24(6)*, IEEE Computer Society Press, Los Alamitos, 2007, pp. 31-38.

[Mysql 2006] MySql: http://www.mysql.org, April 2006.

[Netbeans 2006] Netbeans IDE 5.5: http://www.netbeans.org, April 2006.

[OASIS 2006] OASIS Web Services Business Process Execution Language (WS-BPEL) TC (2006), http://www. oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[Owen and Raj 2003] Owen, M., Raj, J. 2003. BPMN and business process management. http://www.bpmn.org/Documents/6AD5D16960.BPMN and BPM.pdf.

[Papazoglou and Georgakopoulos 2003] Papazoglou, M., Georgakopoulos, D.: "Service-oriented computing", Communications of the ACM: Service-Oriented Computing (2003).

[Pastor et al. 2003] Pastor, O., Fons, J., Pelechano, V.: "OOWS", Department of Information Systems and Computation Technical University of Valencia (2003).

[Peltz 2003] Peltz, C.: "Web Services Orchestration and Choreography", HP Company (2003).

[PXE 2007] Process Execution Engine: http://sourceforge.net/projects/pxe, January 2007.

[Ross and Schoman 1977] Ross, D., Schoman, K.: "Structured Analysis for Requirements Definition", IEEE Transactions on Software Engineering 3(1) (1977).

[Rossi et al. 2003] Rossi, G., Schmid, H., Lyardet, F., Engineering Business Processes in Web Applications: Modeling and Navigation issues, Proceedings of the Third Int. Workshop on Web-oriented Software Technology (IWWOST 2003), Oviedo, Spain, June 2003.

[Schwabe and Rossi 1998] Schwabe, D., Rossi, G.: "Developing Hypermedia Applications using OOHDM", Hypermedia Development Processes, Methods and Models, Hypertext, USA (1998).

[SEI 2006] Software Engineering Institute: "A framework for software product line practice 4.2", http://www.sei.cmu.edu/productlines/framework.html, April 2006.

[Sochos et al. 2004] Sochos, P., Philipow, I., Riebish, M.: "Feature-oriented development of software product lines: mapping feature models to the architecture", Springer, p.138 (2004).

[SOAP 2006] Simple Object Access Protocol: http://www.w3.org/TR/SOAP, April 2006

[UDDI 2006] Universal, Description, Discovery and Integration: http://www.uddi.org, April 2006.

[van Gurp et al. 2001] van Gurp, J., Bosch, J., Svahnberg, M.: " On the notion of variability in software product lines", in: Proc. The Working IEEE/IFIP, WICSA, The Netherlands (2001).

[WebRatio 2007] http://www.webratio.com, September 2007.

[XML 2006] Extensible Markup Language: http://www.w3.org/TR/REC-xml, May 2006.