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

Christian Emig, Jochen Weisser, Sebastian Abeck
Cooperation & Management, Universität Karlsruhe (TH), Germany
{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 in which the software system is applied by its future users are analyzed. Taking an object-oriented approach, 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) offers 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. This article shows 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 widely accepted Business Process Execution Language (BPEL).

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 [2]. User requirements concern the question for which tasks and for which purposes a user wants to utilize the software, bearing in mind that these tasks are part of an overall business process.

In software engineering, user requirements are evaluated in the analysis phase, 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 are 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.

The Unified Modeling Language (UML), a widely accepted language, supports the analysis and the design phase. 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 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 diagrams, use case and activity, together contain the business logic of the software application. In the design phase, the business logic is mapped to components (e.g., a business process control and a number of use case controls) of the application architecture.

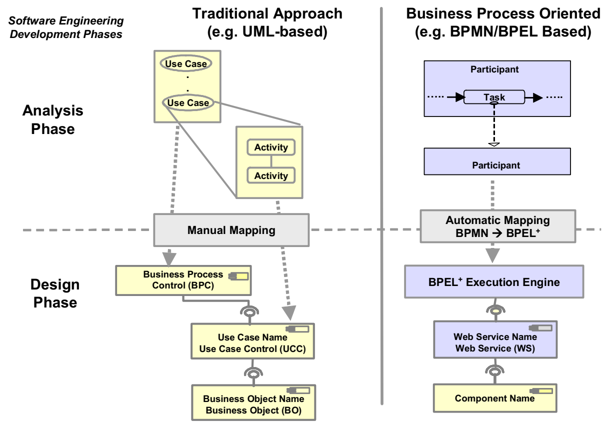

Software Engineering Development Phases
Traditional Approach (e.g. UML-based)
Business Process Oriented (e.g. BPMN/BPEL Based)
Analysis Phase
Design Phase
Use Case
Use Case
Activity
Activity
Participant
Task
Participant
Manual Mapping
Automatic Mapping BPMN → BPEL+
Business Process Control (BPC)
Use Case Name Use Case Control (UCC)
Business Object Name Business Object (BO)
BPEL+ Execution Engine
Web Service Name Web Service (WS)
Component Name

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 in a model at the analysis level or the 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 to a changing process in a flexible way.

To overcome these deficiencies a different process description approach 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 out in this area [3], there are good reasons to choose another language to "program" the business process related aspects. The most important reasons according to [4] are:

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

A process language that fulfills all these requirements is the Business Process Modeling Notation (BPMN, [5]) standardized by the Business Process Management Initiative (BPMI). In our approach to business process oriented programming BPMN is used to produce an executable process description based on the Business Process Execution Language (BPEL, [6]) which is standardized by OASIS. Figure 1 gives an overview of the approach which is described in detail in the next chapters. The rest of the article is organized as follows: In Chapter 2, BPMN and its (graphical) language elements are introduced using 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 it is shown how the BPMN description (i.e., the business 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 – this is described 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" [7]. In the Outlook in Chapter 5, it is pointed 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 are complementary.

2. Programming of a Process Using BPMN

The Business Process Modeling Notation (BPMN) [6, 8] which has been created by BPMI.org 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, continuing to the technical developers responsible for implementing the technology used to perform these processes. The second major goal of BPMN is to reduce the gap between the business process design, the focus of the analysis phase, and the process implementation being looked at in the design and implementation phase. This is ensured by setting up on a mathematically based, internal model that enables the mapping from BPMN's graphical elements to the underlying constructs of (XML-based) executable business process languages like BPEL as illustrated in Figure 1. The tight connection between analysis and implementation provides major advantages to other modeling languages, 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 non-competitive 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 techniques. 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, which is a generic term for work that has to be performed. The second is called 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 basic *Flow Objects*, a business analyst can model a large variety of business process.

To ensure the intuitive understanding of a BPD, 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 a process that can be decomposed into sub-processes. The minutest 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 each of which has a different focus on business processes:

1. Private processes are internal to a specific organization and 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 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 [5].

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 can be mapped to various collaboration languages such as ebXML BPSS [9] or RosettaNet [10]. The example process now looked at is such a collaboration process.

The business process shown in Figure 2 is taken from the higher education sector, 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* each 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 should be supported by computer systems. Most of the activities are quite high-leveled and have a more detailed representation in another BPD. The explanation is found 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 are performed manually. Furthermore, a *Task* can also be defined as a sender or receiver, just sending or receiving one message to or from a different *Pool* and processing it afterwards.

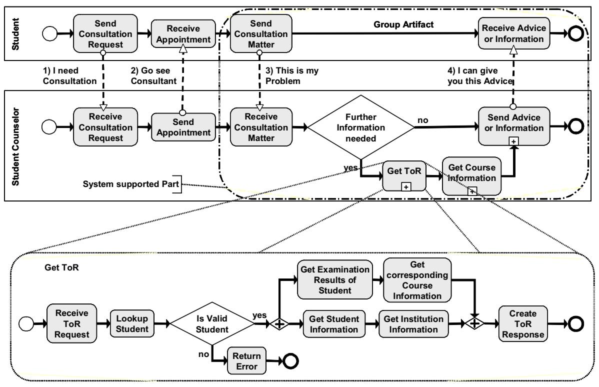


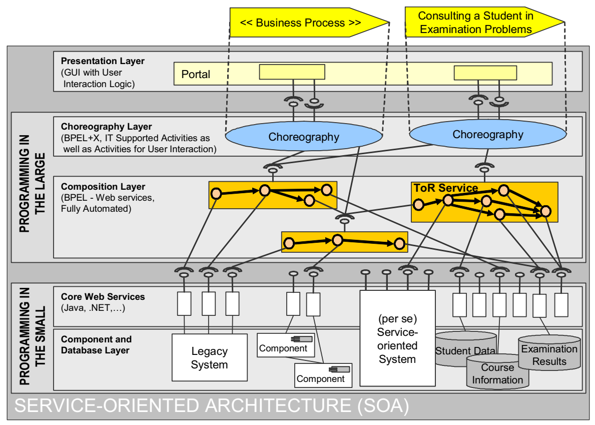
Figure 2. Hierarchical modeling with the BPMN

After the decision gateway where the process determined that further information has to be gathered, 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 this 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 just one *Pool* is concentrated on, BPMN allows us to neglect the outer borderline. This sub-process creates a Transcript of Records in university environments and is used not only in the business process of student counseling but can be part of 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 decision gateway "Further information needed" in Figure 2, esp. a ToR. Two bold circles which denote the possible end points in this sub-process are seen. According to this, the process finishes with either an error, for example because the student does not exist, or the regular return to the calling processes, in particular a valid ToR. Inside the process there is a parallel (AND) forking and joining. The four parallel 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. Before mapping parts of the process to executable elements, first the architecture which is going to be used for the deployment of the service-oriented and process-focused elements must be introduced. Figure 3 shows a SOA design approach from a different point of view than the conventional one focusing at an enterprise service bus [11, 12].

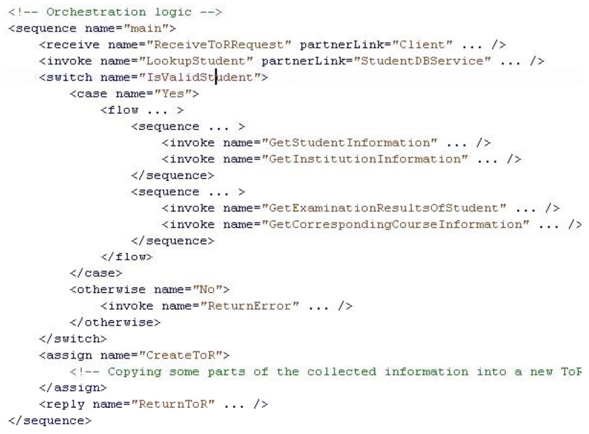


<< Business Process >>
Consulting a Student in Examination Problems
Presentation Layer
(GUI with User Interaction Logic)
Portal
Choreography Layer
(BPEL+X, IT Supported Activities as well as Activities for User Interaction)
Choreography
Choreography
PROGRAMMING IN THE LARGE
Composition Layer
(BPEL - Web services, Fully Automated)
ToR Service
PROGRAMMING IN THE SMALL
Core Web Services
(Java, .NET,…)
Component and Database Layer
Legacy System
Component
Component
(per se) Service-oriented System
Student Data
Course Information
Examination Results
SERVICE-ORIENTED ARCHITECTURE (SOA)

Figure 3. SOA – Architectural design

The entry to the SOA 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,* a less influential layer is located, where composition can take place as well as the interaction with only one participant (no collaboration) – this is subject to being handled by a default BPEL engine. Below this *Composition Layer*, the layer of the core Web services can be found. Core Web services either act as adaptors to wrap existing interfaces of legacy systems making them Web service capable or are Web service interfaces of a per se service-oriented systems.

The analysis and design can be done by using BPMN. When the implementation phase is reached, a mechanism or concept is needed to automatically generate BPEL code for the business process out of a BPMN representation. Before illustrating and discussing a possible concept, let us take a brief look at the BPEL code shown in Figure 4.



```
<!-- Orchestration logic -->
<sequence name="main">
    <receive name="ReceiveToRRequest" partnerLink="Client" ... />
    <invoke name="LookupStudent" partnerLink="StudentDBService" ... />
    <switch name="IsValidStudent">
        <case name="Yes">
            <flow ... >
                <sequence ... >
                    <invoke name="GetStudentInformation" ... />
                    <invoke name="GetInstitutionInformation" ... />
                </sequence>
                <sequence ... >
                    <invoke name="GetExaminationResultsOfStudent" ... />
                    <invoke name="GetCorrespondingCourseInformation" ... />
                </sequence>
            </flow>
        </case>
        <otherwise name="No">
            <invoke name="ReturnError" ... />
        </otherwise>
    </switch>
    <assign name="CreateToR">
        <!-- Copying some parts of the collected information into a new ToF
    </assign>
    <reply name="ReturnToR" ... />
</sequence>
```

Figure 4. BPEL code

The displayed code in Figure 4 is fragmentary. Most of the attributes, variables and assigns are omitted for better readability. The complete error handling is also left out.

The general structure of any BPEL process looks likes this: Inside the *process* tag which is the outer element of such an XML document there are basically three different sections. In the first section *partners* and *partnerLinks* are defined. The *partnerLinks* announce the external programs which are involved such as 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 to be discussed later.

The second section contains all the variables of 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 further here. The third section is the orchestration 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 Object* 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 participating Web service description.

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.

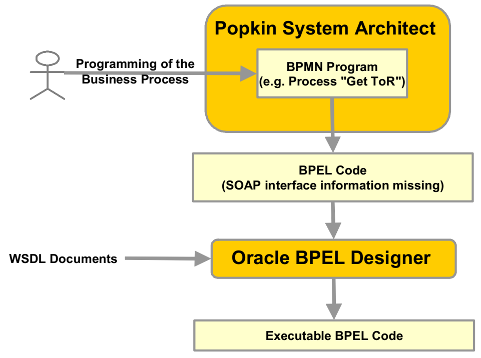

Popkin System Architect
Programming of the Business Process
BPMN Program (e.g. Process "Get ToR")
BPEL Code (SOAP interface information missing)
WSDL Documents
Oracle BPEL Designer
Executable BPEL Code

Figure 5. The approach taken to support BPMN-based process programming

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 [5]. Some BPMN modeling tools like Popkin's System Architect offer functions which automatically generate BPEL code out of BPMN. But there is one drawback. 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 to using such a tool is to add the remaining information manually. Most of the design tools allow importing the 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. For more extensive verification of Web service compositions, there exist various tools like for example the LTSA-Eclipse Tool described in [13].

4. Execution of a BPEL Process

After developing the BPEL program as described in Chapter 3 the question arises as to 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 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 the BPEL code and executes the contained instructions. Examples of existing BPEL engines are ActiveBPEL [14] by ActiveEndpoints and the Oracle BPEL Process Manager [15]. 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. Conclusion and Outlook

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

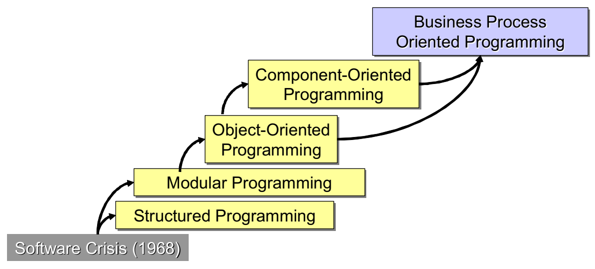

Business Process Oriented Programming
Component-Oriented Programming
Object-Oriented Programming
Modular Programming
Structured Programming
Software Crisis (1968)

Figure 6. Evolution of software engineering

Figure 6 gives an overview of the most important steps in software engineering. Caused by the software crisis, software development is no longer seen as an art but as a structured engineering process. To be able to decompose complex software problems, concepts such as modularization, object orientation, and components were introduced. This resulted in well-structured programs. Another important result of these concepts is reusability of software parts (modules, objects, components) which makes the development of software more efficient and leads to more stable software systems.

These engineering concepts essentially concentrate on the design and implementation of a software system. They do not adequately support the first phase of software engineering where business processes of the future user of the software system have to be analyzed, as pointed out in Chapter 1. Business process oriented programming fills this gap by providing executable process descriptions which are based on service-oriented elements, typically Web services. There is a separation of business-related aspects to be covered by the processes, and technical aspects to be covered by components and objects. This service is the combining element of these two types of programming, the "Programming in the Large" and "Programming in the Small". Although business process oriented programming is still in a very early stage, the high potential of this evolutionary step in software engineering is obvious. Standards like BPMN and BPEL are available to apply this concept to practical software problems as demonstrated in the paper.

6. References

[1] Christian Emig, Christof Momm, Jochen Weisser, Sebastian Abeck: Programming in the Large based on the Business Process Modeling Notation, In: Proceedings to Informatik 2005 – Informatik LIVE!, http://www.informatik2005.de, 2005.

[2] Ian Sommerville: Software Engineering – Seventh Edition, Addison-Wesley, 2004.

[3] Keith Mantell: From UML to BPEL Model-driven Architecture in Web Services World, IBM, 2003.

[4] Martin Owen, Jog Ray: BPMN and Business Process Management – Introduction to the New Business Modeling Standard, Popkin Software, 2003.

[5] Business Process Management Initiative (BPMI): Business Process Modeling Notation (BPMN), Version 1.0, BPMI.org, May 2004.

[6] Business Process Execution Language for Web Services (BPEL4WS), Version 1.1, http://www.ibm.com/developerworks/library/ws-bpel, May 2003.

[7] Frank Leymann: Web Services — Distributed Applications without Limits, Business, Technology and Web, Leipzig, 2003.

[8] Stephen A. White: Introduction to BPMN; IBM Cooperation 2004.

[9] Business Process Specification Schema (ebXML BPSS), Version 1.01, http://www.ebxml.org/specs/, OASIS, May 2001.

[10] Rosettanet Implementation Framework, http://www.rosettanet.org/rnif, 2002.

[11] Thilina Gunasinghe, Tim Kelly: Establishing a Standard Business Process Execution Architecture for Integrating Web Services, Proceedings to IEEE International Conference on Web Services (ICWS), 2005.

[12] The Burton Group: Service-Oriented Architecture – Developing the Enterprise Roadmap, February 2005.

[13] Howard Foster, Sebastian Uchitel, Jeff Magee, Jeff Kramer: Tool Support for Model-Based engineering of Web Service Compositions, Proceedings to IEEE International Conference on Web Services (ICWS), 2005.

[14] ActiveBPEL – Open Source BPEL Engine, http://www.activebpel.org.

[15] Oracle BPEL Process Manager, http://www.oracle.com/technology/products/ias/bpel.