Business Process Automation and Web Service Choreography

Project Work

Submitted by:

Sa Liu

Information and Media Technologies

Matriculation Number: 24060

Supervised by:

Prof. Dr. J. W. Schmidt

STS - TUHH

M.Sc. Miguel GARCIA

STS – TUHH

Technical University Hamburg Harburg, Germany

2004-06-29

I declare that:

this work has been prepared by myself,

all literally or content-related quotations from other sources are clearly pointed out,

and no other sources or aids than the ones that are declared are used.

Hamburg, 29.06.2004

Sa Liu

Abstract

The idea of Web Service Choreography is to integrate and coordinate different service partners in a business process, so that the workflow can be automated. BPEL4WS provides an XML-based syntax to describe the execution of business process.

This paper focuses both on the conceptual investigation of Web Service Choreography and practical implementation of business processes. The technical requirements of Web Service Choreography and Orchestration are to coordinate, manage and monitor the flow. Currently a few of products have been developed to fulfill these requirements. Collaxa BPEL solutions are taken as examples in this paper.

Collaxa Orchestration Server makes it possible to model, execute, and manage complex business processes. It includes a graphical editor, the Collaxa BPEL Designer, which enables the business analysts and developers to visually model and define the processes. The BPEL Designer is integrated in Eclipse platform. Collaxa Orchestration Server provides an execution environment for the process. It supports concurrent invocation of multiple Web Services, asynchronous messaging, dehydration (de-activation) of process instances, as well as run-time error and events handling. Collaxa Console is a Web tool that enables the developers to view, debug and manage running processes. Monitoring of a process can be either graphic view or text audit.

A scenario of a business process is realized based on Amazon Web Service and Barnes&Noble Web Service. The case study also suggests the implementation phases of a BPEL project with Collaxa products. A set of Flash demos are developed to illustrate the usage of the tools, thus making easier to evaluate finer aspects of the tool support.

Table of Contents

Table of Figures

Table of Examples

1. Introduction

1.1. Background

As Web services rapidly emerge to be the most effective approach of integrating customer, vendor, and partner applications in a platform- and language-independent manner, more and more companies begin to develop and publish their individual Web services. This brings new challenges to integration of enterprise applications, both internally and externally. These web services must be connected and the collection of Web services should be specified to implement more complex functionalities, typically a workflow or business process.

Business collaborations require long-running interactions driven by an explicit process model. However, existing approaches to business process modeling are not designed to interact with components crossing organizational boundaries. Failing to satisfy the requirements of collaboration between those individual applications may lose the significance of Web services. Therefore, the current trend is to express the logic of a composite Web service using a business process modeling language.

Accordingly, the main technology companies start to converge the existing business process languages to a new standard. Early works include Web Services Conversation Language (WSCL), IBM Web Services Flow Language (WSFL) and Microsoft's XLANG and so on. WSCL allows defining the abstract interfaces of Web services, focused on modeling the sequencing of interaction between Web services. WSFL was an IBM protocol specifying two types of Web services composition 1) an executable business process known as a flowModel, and 2) a business collaboration known as a globalModel [Virdell, 2003]. The flow model represents the series of activities in the process, while the global model binds each activity to a specific web service instance.

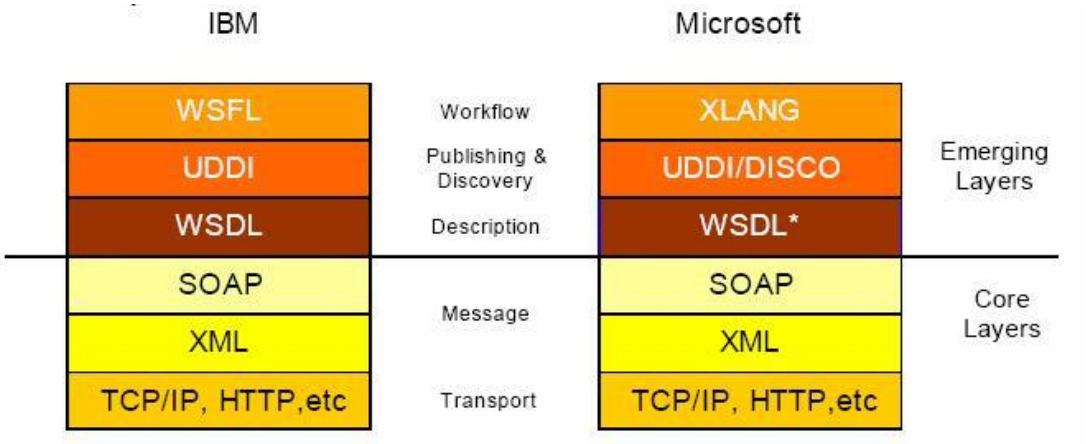


Figure 1 Emerging Standards of Web Service in 2002 [Greenwood, 2003]

Microsoft's XLANG was the Business modeling language for BizTalk, which is a

component of .NET that enables EAI. XLANG focused on the creation of business processes and the interactions between web service providers. The specification provided support for sequential, parallel, and conditional process control flow. 0 gives a layered view of Web service standards supported by IBM and Microsoft.

In May 2003, Microsoft, IBM, Siebel Systems, BEA, and SAP authored the 1.1 release of the specification of Business Process Execution Language for Web Services (BPEL4WS). BPEL4WS is the cooperative merging of WSFL and XLANG, for Web services choreography, workflow, and composition. It provides an XML-based syntax to describe the control logic, how the coordinate Web services participate in a business process. BPEL4WS is essentially a layer on top of WSDL, with WSDL defining the specific operations allowed and BPEL4WS defining how the operations can be sequenced [Peltz, 2003].

BPEL4WS supports both executable and abstract business processes. An executable process models the behaviour of participants in a specific business interaction. Abstract processes, also named as business protocols in BPEL4WS, specify the public message exchanges between parties, without exposing the internal details of a process flow.

BPEL itself is defined as a Web service, whose interface is described by a WSDL, so that recursive composition with other Web services is supported. The specification provides execution of sequential, parallel, and conditional process control flow. It also includes a robust exception handling facility, with support for long-running transactions through compensation. In April 2003, BPEL was submitted to OASIS to obtain a broader industry acceptance and open standardization

1.2. Objective of the study

The aim of this report is to investigate the idea of Web Service Choreography and Orchestration. A set of emerging technologies, including BPEL4WS (Business Process Executable Language for Web Service), Web Service Choreography and Orchestration will be studied thoroughly in this report. To help understanding the theory, concrete design and execution of BPEL using BPEL editor and engine should be performed, especially the Collaxa products. The Collaxa BPEL Designer provides a visual editor to design and modify business processes from a developer's point of view. The Collaxa BPEL Server provides a scalable and reliable run-time environment for deploying, executing and managing BPEL processes. Demos using Collaxa BPEL Console to deploy a business process will be taken as an example for illustrating the result of the investigation.

According to the three main requirements of Web Service Choreography, the objectives of this study can be summarized as follows:

- *Coordination*: using open standards (Java/J2EE, JMS, XML, SOAP, WSDL) to build loosely coupled services and to support non-linear asynchronous interactions.
- *Management*: to coordinate, store and manage the state of each conversation while using asynchronous messaging; to handle business level exception and to manage non-linear transaction, cancellation and compensation.

- *Monitoring*: to provide business visibility on the state of the conversations and to trace the execution of the business process; also to supply detailed reports relating to the process execution.

1.3. Overview of the report

The report is organized into the following chapters.

Chapter 2 starts with the introduction of Web Service Choreography and Orchestration. Three technical requirements are discussed in detail. The current vendors supporting Web Service Choreography are briefly introduced.

Chapter 3 provides detailed investigation of the Collaxa BPEL Designer and Server. The main features of the BPEL Server are discussed in detail, along with some business scenarios. Throughout the study, the absence of some capabilities to support BPEL should also be discovered.

The case study is illustrated in Chapter 4 by going through a concrete deployment of a business process. The case study should better explain and support the analysis and statements in Chapter 3. Some screenshots and code snippets help to demonstrate the whole procedure when applying the BPEL Designer and Server.

Finally a conclusion is driven in Chapter 5, summarizing the strength of the BPEL Server, lessons learned from the study, future prospect of Web Service Choreography, etc.

2. Web Service Choreography / Orchestration

2.1. Introduction to Web Service Choreography / Orchestration

"Web services choreography, orchestration, and general business process management are the programming equivalents of ballroom dancing" [Rhody, 2003]. As these terms like Choreography and Orchestration occur more and more often in Web services technical articles, their original meanings are almost ignored. But actually they are visualized analogy to the execution of business processes. Since Web service itself is making multiple applications work together, choreography defines the behaviors, the timing and ordering of each individual "dancer " – each single Web service. While a group of dancers perform in a coordinated manner, the choreographed performance depends on their physical interactions. Variations or errors brought by each individual may cause changes to the others. Thus, choreography is associated when tracking the sequences and public messages exchange among multiple parties.

In general, Web Service Choreography focuses more on the external view, as illustrated in 0 The external flow describes the information exchange among multiple participants, including suppliers, customers and partners.

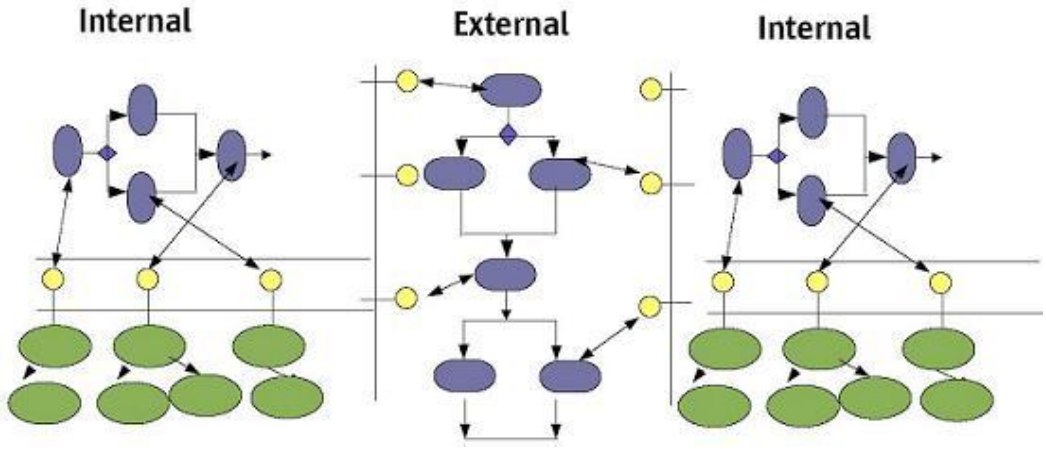


Figure 2 Web Service Choreography [McDonald, 2003]

While Web Service Orchestration describes how web services can interact with each other at the message level, including the business logic and execution order of the interactions. This concentrates more on an internal view, tracking the interactions with certain timing and order, which result in a long-lived, transactional, multi-step process model. It is like a musical director imposes order and timing individually on a set of musicians in order to coordinate their performance in a concert.

It is said that Orchestration is more static, rule-based automation of workflow and Choreography is more dynamic and people-based. But the distinctions between them are actually blurring and the two viewpoints are converging. Merging the two viewpoints means that the ability of workflow software to flexibly handle varied and unexpected

circumstances becomes very important [Virdell, 2003].

As has been point out in 0, Orchestration lays on top of the WSDL layer and the SOAP messages. This gives a reasonable answer to the next question: Why do we need Web Service Choreography and Orchestration?

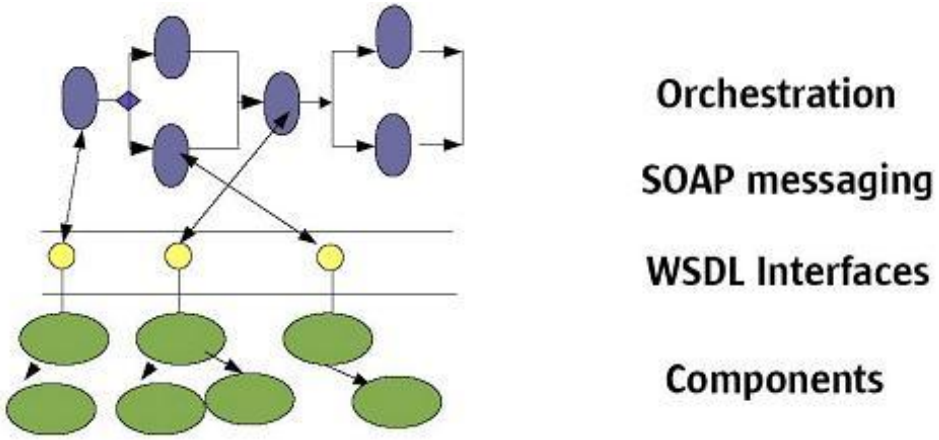


Figure 3 Web Service Orchestration [McDonald, 2003]

Enterprise applications are modelled in components with their public interfaces exposed. WSDL describes these interfaces through standardized XML semantics and defines protocols and messaging end points of Web services. SOAP makes the platform- and language-independent XML format messaging possible. Above all these layers, the automated business processes are realized by Web Service Orchestration and Choreography.

2.2. Technical requirements of Web Service Choreography

There are many important technical requirements that must be addressed when designing business processes involving long-lived multiple web services. Knowing of these requirements will help in positioning the various standards that have been introduced for Web Services Choreography.

2.2.1. Coordination

As the main objective of Web Service Choreography is to manage composition and collaboration between web services, variability during the process becomes a challenge. For instance, new components are integrated into the application, business level exceptions and timeouts, conditional branching and parallel execution, etc. These requirements demand great flexibility and reliability. Invoking services asynchronously allows achieving the non-linear interactions between services. With asynchronous support, a business process can invoke web services concurrently rather than sequentially in order to enhance performance. For example, in a ticket booking system, the customers may want to interact with multiple supplier web services at the same time, looking for the one who can offer the lowest price, earliest shipment date and best service. Asynchronous messaging can be achieved through web services by applying various correlation techniques.

2.2.2. Management

Web Service Choreography composes both local, tightly coupled components and remote, loosely coupled components. These components may be invoked synchronously or asynchronously. To realize sophisticated management of choreography, the following aspects should be considered:

- The logic embedded in a process-centric application includes various building blocks, interfaces, and protocol behaviors. Efficient design and deployment guarantee a successful execution later, but still the version control should be supported in order to trace the steps and roll back in case necessary.
- Long-running processes should consider not only exceptions, but also timeout and transaction integrity. The system must react in a proper way if it does not get a response after some time. Also it has to be taken into account, how the transactional integrity is managed. Here the traditional ACID rules may not satisfy the requirements, since the transaction is talked for distributed applications. Compensation is performed in case of cancellation.
- Components may need to be upgraded gracefully, without interfering with their execution and human intervention may be needed for handling exceptions, timeouts, and altering the application's functional execution. This requires the system to be scalable and adaptable.

2.2.3. Monitoring

The life cycle of the process-centric applications need be monitored, from specification through maintenance. Different audience may require visibility at their respective level of involvement and comprehension of the process functionality. For example, business analysts want to visualize a high-level view of the business process functionality. Developers need to trace the execution of the business process to debug during developing phase. Quality managers may need the detailed auditing of the process and business owners require viewing reports relating to the process execution to derive business intelligence and achieve ongoing process optimization.

2.3. Current Vendors Supporting Web Service Choreography

To investigate Web Service Choreography, a glance should be taken to the current vendors and products. The following is an overview of the BPEL editors and servers supporting Web Service Choreography and Orchestration. Information about these tools was collected from their white papers and some technical articles.

- Collaxa: provides Collaxa BPEL designer for creating and editing BPEL, Collaxa Orchestration Server for business process execution and Console for monitoring and management.
- IBM: offers BPEL4J editor and engine. The editor, as same as Collaxa Designer, is a visual tool for designing BPEL. The runtime engine can be deployed on both Tomcat 4.0.1+ and WebSpehere 4.x and 5.x. BPEL4J also includes a web-based interface for viewing deployment of processes.

- Microsoft: BizTalk Server 2004 contains a superset of BPEL4WS capabilities, including nested processes, long-running transactions, correlations and mapping between messages. It orchestrates business processes and seamlessly integrated with MS Office 2003 and Visio. The graphic process design from a business perspective is fulfilled by Visio, then it is exported to Visual Studio .Net and the orchestration is built on BizTalk Server. Finally MS Excel is used to monitor data of running process.
- OpenStorm and Sonic also announced for their abilities to support Web Service Orchestration and Composition in their products.

In this paper, Collaxa products are taken as an example and will be further inspected in Chapter 3.

3. Collaxa BPEL Solutions

Collaxa provides a standard-based software infrastructure to design, deploy and execute collaborative business processes.

The Collaxa Designer is a visual editor of a BPEL project, which provides a simple user interface for creating and modifying BPEL files. The process developer can drag and drop BPEL activities to a "process map" and edit the attributes of each element using the corresponding wizard. By building the BPEL project, the process flow is compiled and then can be deployed on the Collaxa Orchestration Server.

The Orchestration Server provides the underlying infrastructure, handling asynchronous messaging, business transactions and processes coordination. According to the XML-based standards, the WSDL binding framework connects the existing messaging infrastructures so that it enables the interoperability between systems. The BPEL Server also supplies user task definition, which is mainly associated with portal systems. To support asynchronous conversation, the flow can be dehydrated. This means the current state is persisted in a database (to free resources), from where it can be later brought back to main memory to continue processing.

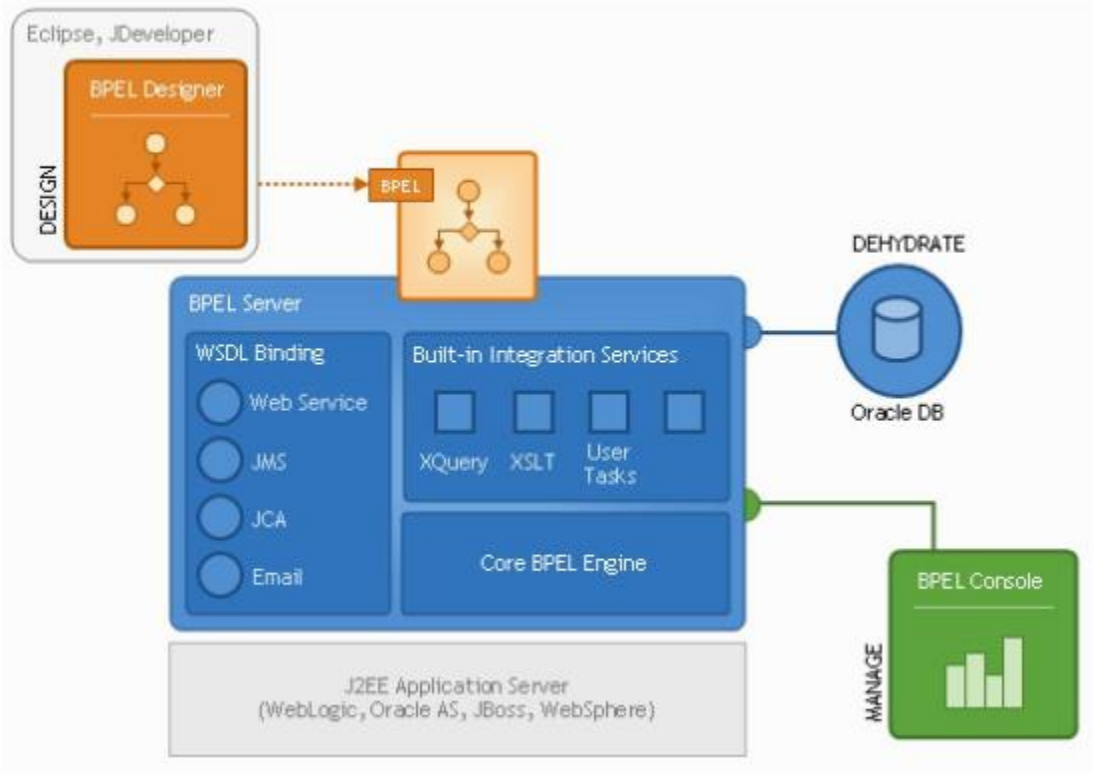


Figure 4 Collaxa BPEL Solution Schematics [Collaxa, 2004]

The BPEL Console is a management tool binding with the server. Runtime execution of a process can be monitored in the console. There are three views available in the console:

visual flow, audit instance and debug instance. Developers can trace the process execution by looking at each activity in the visual flow, view a more detailed textual representation of audit trail and debug the variables of the dehydrated instances.

Figure 4 gives architecture of the Collaxa products. In the following sections, a more detailed explanation of applying Collaxa Designer and Server is given.

3.1. Collaxa BPEL Designer

Collaxa Designer is a BPEL editor. Its current version (2.0) is integrated in Eclipse 3.0M7, with its own perspective window, as shown in Figure 5.

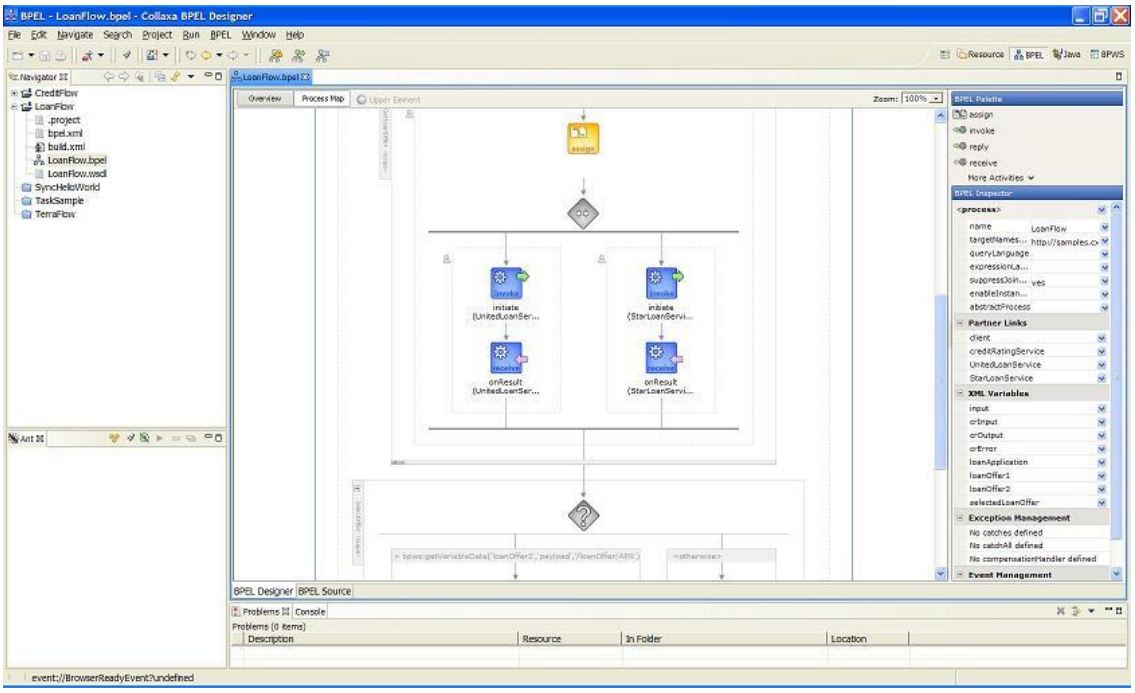


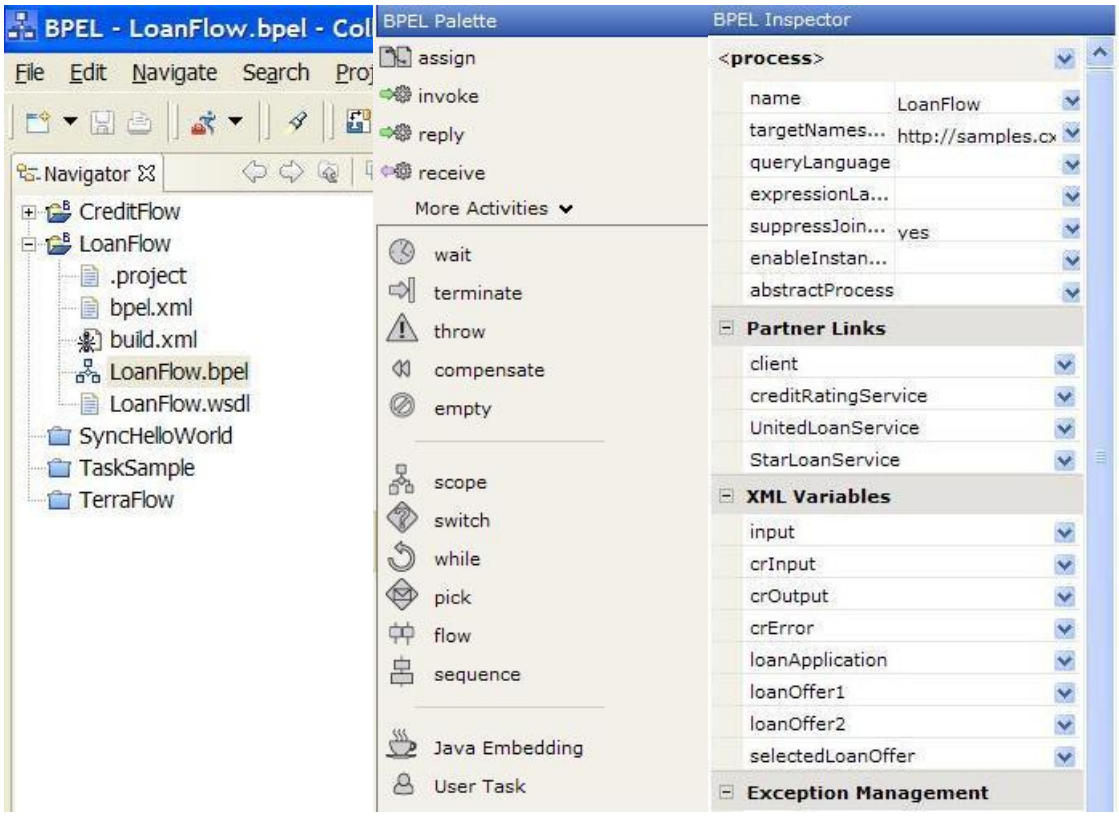
Figure 5 Collaxa BPEL Designer

The BPEL Designer includes a New Project wizard that automatically generates the skeleton of a BPEL project: a BPEL source, containing the minimum activities and definitions; a WSDL file defining the client interface for this process; a BPEL deployment descriptor, which defines the locations of the WSDL files for services called by this flow, along with other project-specific parameters; an Ant script for compiling and deploying the BPEL process. At the left side there is a small window for project navigation (Figure 6.a), listing the above mentioned files of each project.

By opening the BPEL file, an overview of the BPEL is displayed in the main window in the middle, with its client interface on the left side, global variable list in the middle and partnerLinks on the right. Something worth to be mentioned is, for each partnerLink, all the services provided by the partner are automatically loaded, through the specified WSDL of the partner.

The process map tab in the middle window gives the possibility to visualize the flow representation of a BPEL, as illustrated in Figure 5. To create a more complex BPEL,

Collaxa Designer provides the drags and drops from the BPEL Palette (Figure 6.b) on the right side of the main window, to compose sequences, flows, scopes and assignments. Additionally, using the activity User Task, a Task Service can be initialized, which is a generic service bundled with Collaxa BPEL Server and accessible via a standard Web Service interface. In this way, manual tasks can be integrated into a BPEL process just as any other Web services. It is also possible to embed java code in the process using Java <exec> activity, which utilizes the BPEL extension capabilities.



a. Project Navigator *b. BPEL Palette* *c. BPEL Inspector*

Figure 6 Creating BPEL Projects

The inspector pane on the right (Figure 6.c) allows user to initialize and edit attributes of selected activity. For example, to add a new assign activity, drag an "assign" from the palette and drop it to the proper position on the process map, then name it in the inspector. A "copy rule" should be created in the inspector by filling the specific variables, parts and XPATH Query in "from" and "to" elements respectively.

3.2. Compiling and deploying

The Collaxa BPEL server provides a runtime environment for deploying, running and managing BPEL processes. After creating the process model in BPEL Designer, the flow can be compiled in a command prompt using the command "cxant". This will set some environment variables and then invoke the Apache Ant utilities. "cxant" initiates the compiler "bpelc" to create an execution map for the process. "bpelc" utility generates a BPEL archive, a jar file that contains all the classes needed to create and run instances of

the process, so that it can be executed by the Collaxa Server, passivated when it is necessary and correlated with its asynchronous service invocations. Figure 7 shows an example of using “bpelc” compiler in windows system. The "-deploy" option specifies that the process should be automatically deployed to the BPEL Server so that it is available for testing and invocation. The "-rev" option specifies a revision number that is used for BPEL Process versioning.

```
C:\collaxa\samples\interop\microsoft\TerraFlow>bpelc -deploy default -rev 1.0
--------------------------------------------------
Collaxa BPEL Process Processor Version 2.0
http://www.collaxa.com/developer.support.html
Copyright (c) 2002 - Collaxa Inc (Patent Pending)
(type bpelc -help for help)
--------------------------------------------------

bpelc> validating "C:\collaxa\samples\interop\microsoft\TerraFlow\TerraFlow.bpel
" ...
bpelc> BPEL suitcase deployed to: C:\collaxa\domains\default\deploy

bpelc completed successfully.
```

Figure 7 Compiling a BPEL project

To build the project, the deployment descriptor bpel.xml file is required. It specifies where to find the WSDL files for services that are called by the BPEL Process and other run-time configuration parameters. The deployment of BPEL is not described by its specification and therefore vendor specific. To port a process running on one engine to another engine, the deployment information should be modified accordingly.

3.3. Collaxa BPEL Console

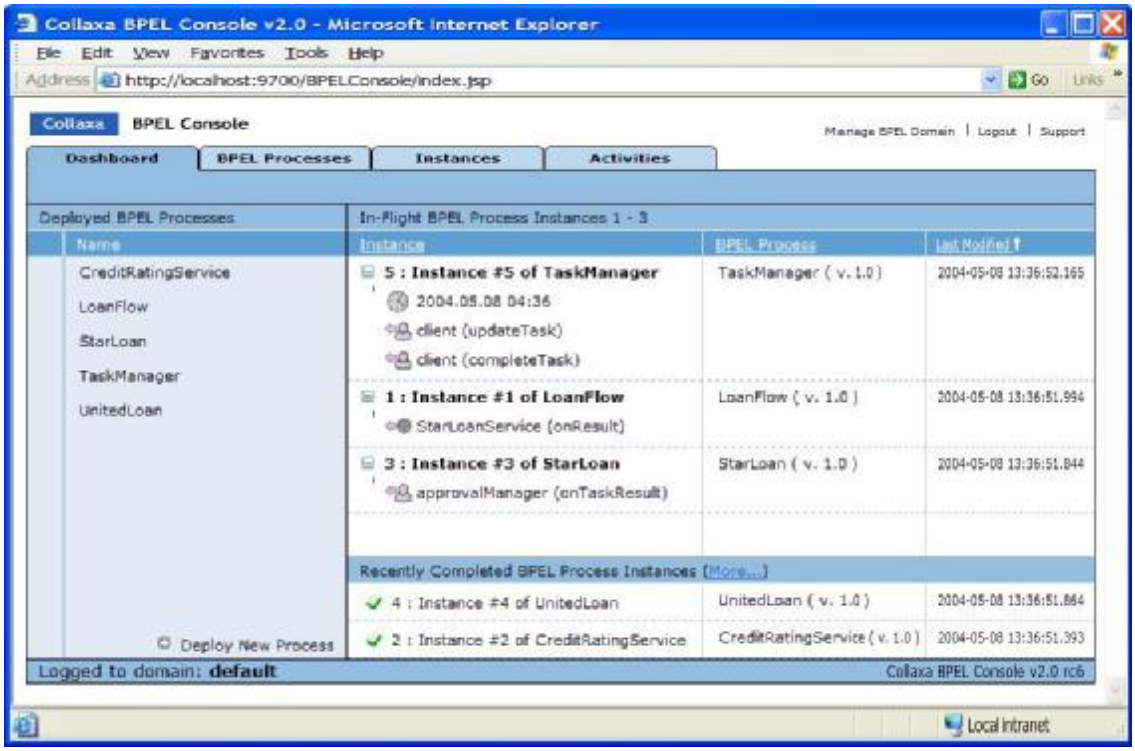


Figure 8 BPEL Console: overview of process instances

Collaxa provides the possibility to monitor and manage BPEL execution in a BPEL Console, a web-based user interface. After deployment, start the server and login to the start page under URL http://localhost:9700/BPELConsole.

An overview of the console is all around the deployed processes, listing on the left side. On the right side are instances of these processes, with those who are still pending on above and the completed instances at bottom (See Figure 8).

By clicking the name of a process, we can come to the site for initiating a new instance (See Figure 9). Here is the start point of execution of this BPEL. Data needed for initiation can be either filled in a HTML form or specified in XML source. Once the data is submitted, namely a message is sent to the BPEL's Web service, the client invokes the BPEL and the flow start to run.

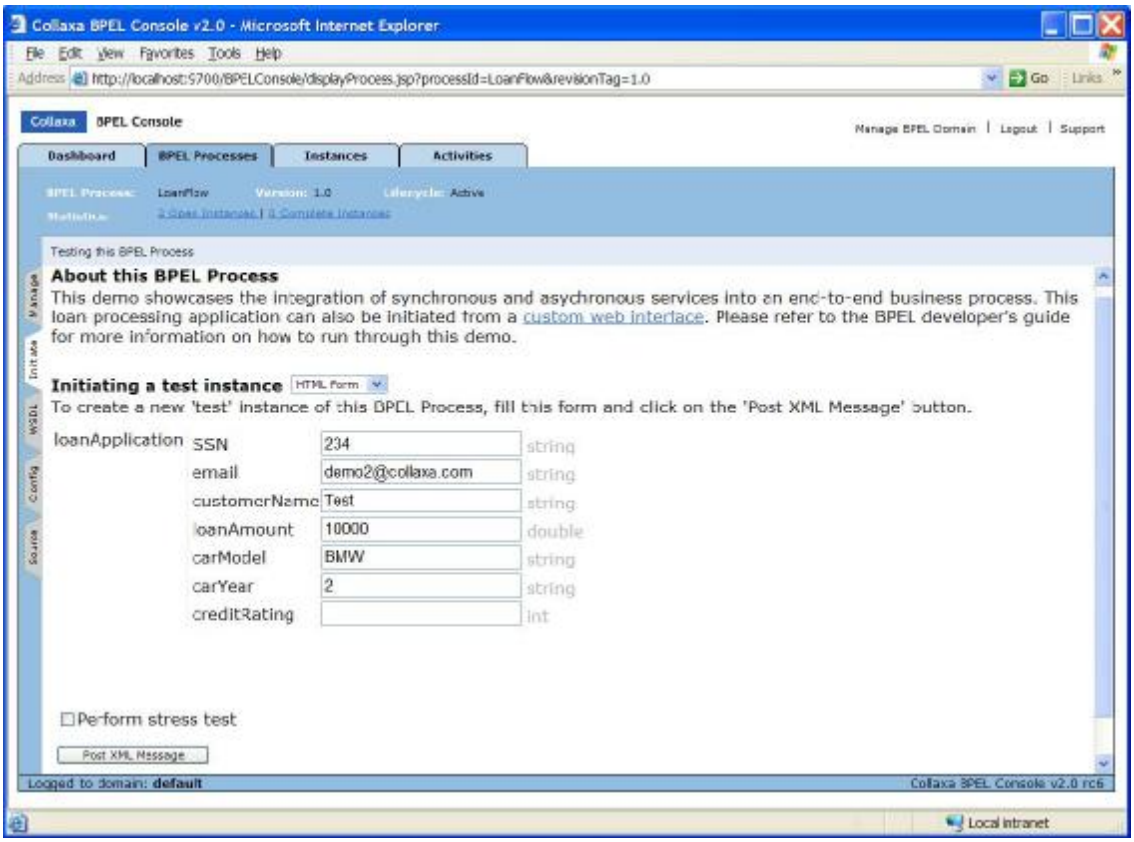


Figure 9 BPEL Console: initiate a new instance

To monitor the execution of a process, Collaxa Console provides three kinds of representation: the visual flow, a graphic view of the process activities; the audit trail, the text audit of the instance; and the debugger for inspecting current state of an instance when waiting for an asynchronous call back (Figure 10).

Selecting the Visual Flow link, we can see a graphic representation of the current state of the flow (Figure 11). The graphic model is the same as designed in Collaxa Designer. It is to some sense similar to the activity diagram in UML, with arrows connecting each activity represented by a node. To be noticed here is that, only the BPEL specific and the Collaxa specific activities are used to construct the model.

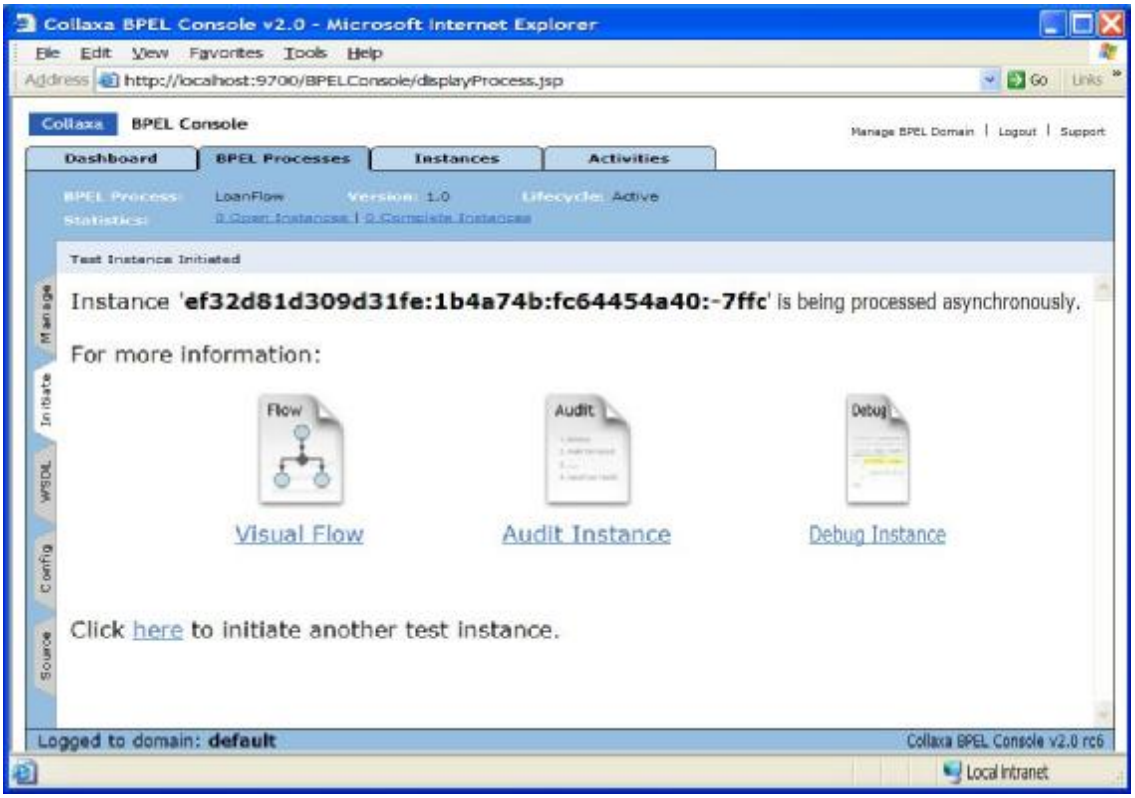


Figure 10 BPEL Console: three views of an instance

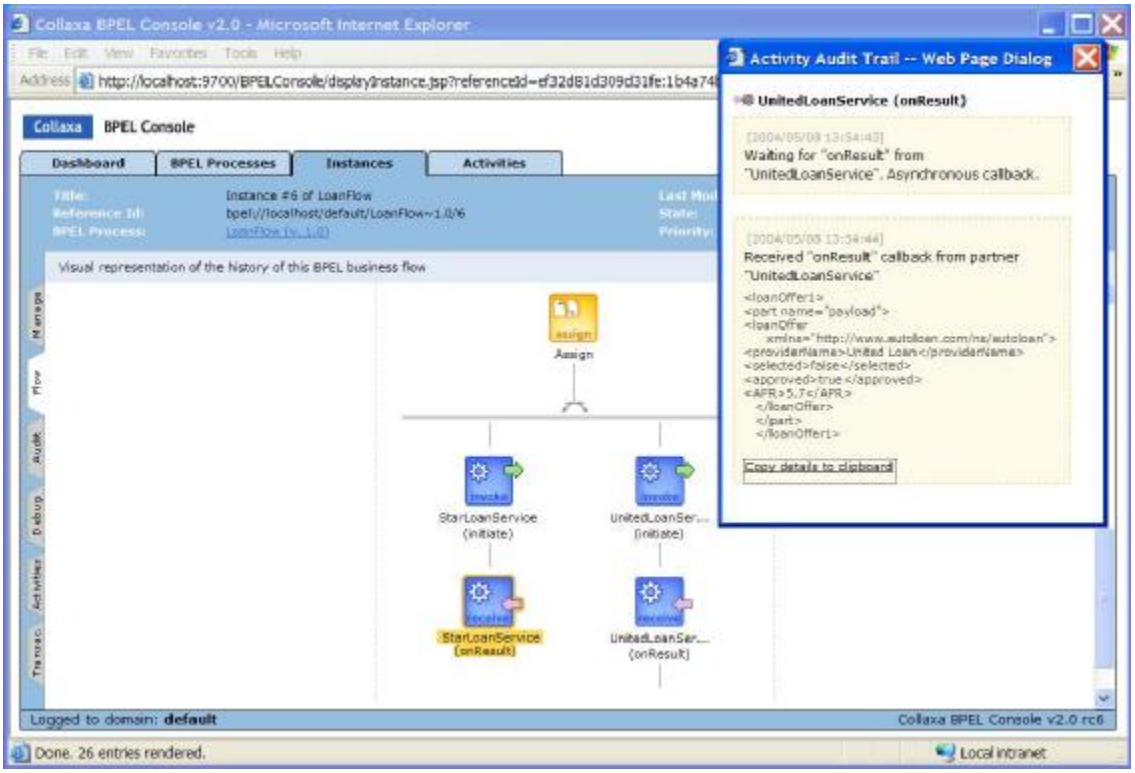


Figure 11 BPEL Console: visual view of the flow instance

By clicking each activity node, a popup window is opened, showing the message

exchange in this activity. Figure 11 gives an example of the value of an asynchronous callback. The <receive> activity of StarLoanService is marked as yellow, indicating that the flow is dehydrated because no response got from StarLoan service provider yet. The process will wait until it gets the return value or timeout.

Clicking the "Audit" tab along the left hand side of the window, a text audit is available to be inspected, with message exchanges occurred in each activity (Figure 12). The audit trail is also organized in a XML-like hierarchy, with sub-activities under their super-activities. In this example, the dehydrated activity is marked as pending. The link "more" or "less" shows or hides the detailed information of the XML variables.

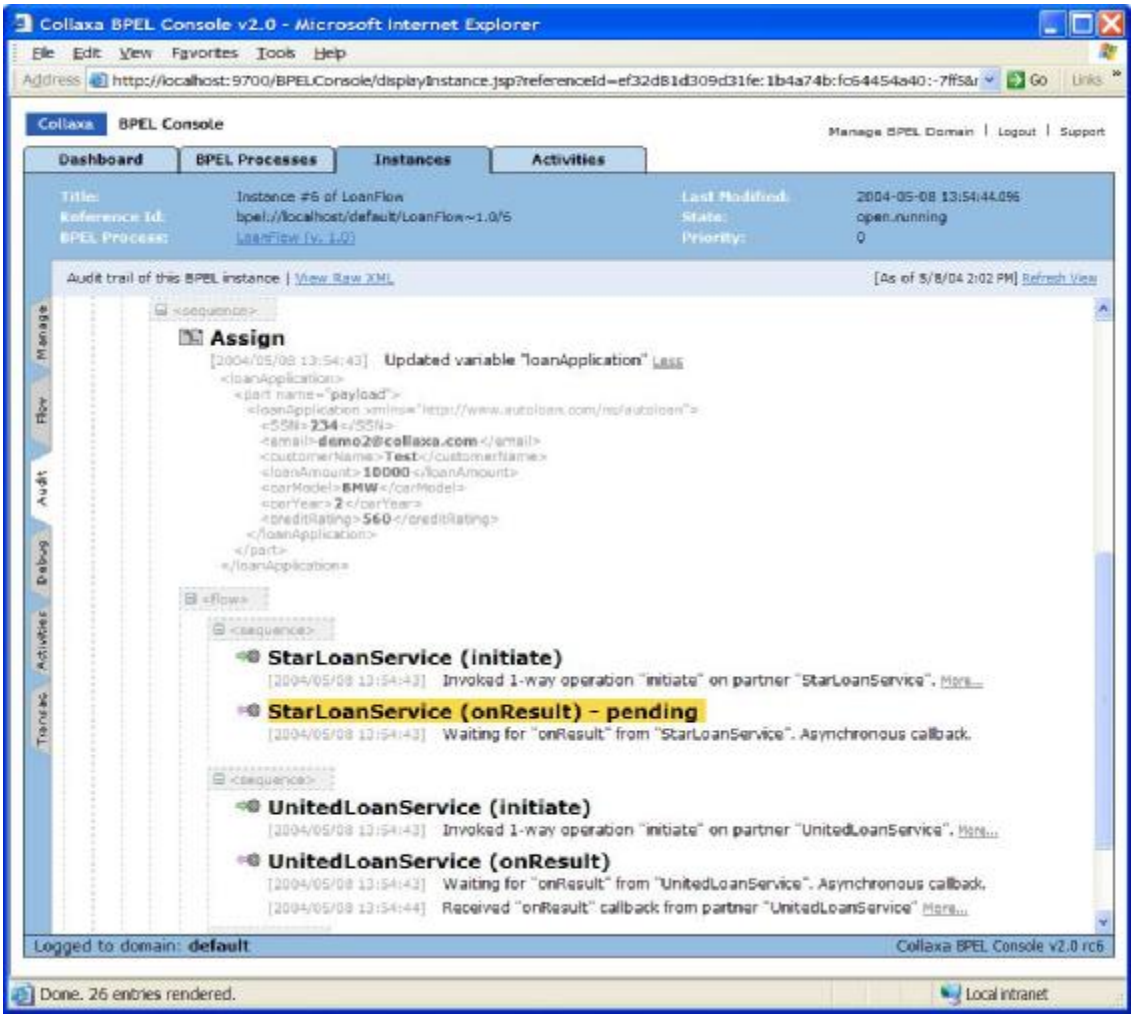


Figure 12 BPEL Console: audit trail of the flow instance

Another representation of the flow is shown in the debugger, opened from the tab "Debug" on the left side. It is similar to a program IDE's debug perspective. The asynchronous <receive> activity is highlighted like a breakpoint in a program. By clicking the link of each variable, a popup window shows its current value (Figure 13).

This is convenient for the developer to view the BPEL source and check the XML variables' state if the flow is pending or exceptions occur. But it is not a real sense debugger, since no break points can be set manually. That is to say, if some invalid value

is returned from the service provider or some uncaught exceptions, but still accepted by the flow, it won't be possible to stop the process and debug it step by step. This might be a new feature enabled in the future version of Collaxa Console.

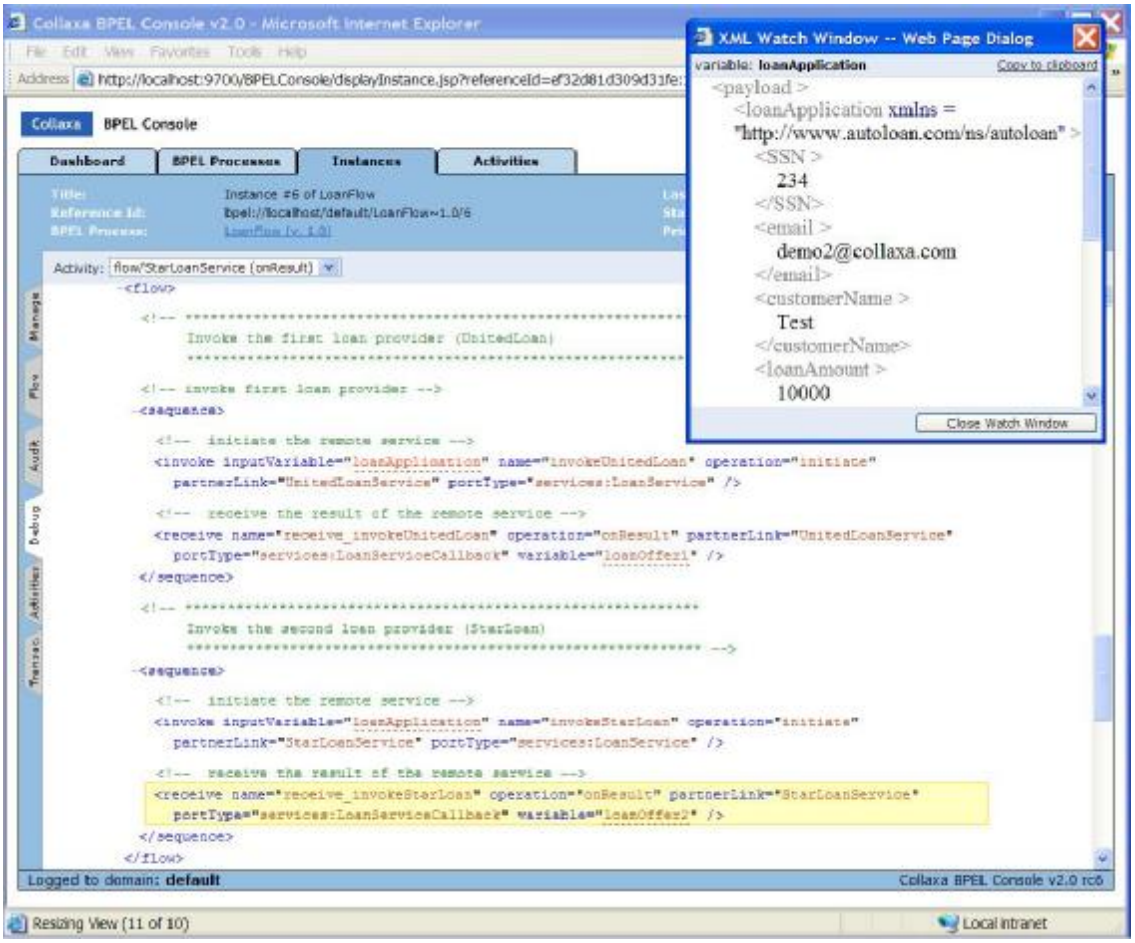


Figure 13 BPEL Console: debug the flow instance

Collaxa Console also gives other possibilities to manage the flows, instances and activities. All processes, instances and activities have their corresponding states. A process's state can be on or off, indicating its lifecycle is active or retired. New instances can only be initiated from active processes. There are four states for each instance: running, completed, cancelled or stale. Instances can be searched by their states, time of creation, process to which it belongs, or some other attributes. To improve the server's performance, closed instances can also be exported to an external datastore. To clean the testing/development environment, it is also possible to delete all instances on the server by clicking a single purge button.

3.4. Main Features of the Collaxa BPEL Server

Figure 14 shows the system architecture of the Collaxa BPEL Server. The server is the heart of the solution. It executes BPEL processes and correlates and coordinates asynchronous interactions into collaborative and transactional business flows. The

dehydration module passivates BPEL instances in a database while waiting for callbacks from remote partnerLinks and other asynchronous or long-running activities. The WS-T module coordinates compensating business transactions across loosely-coupled services. The version control module offers side-by-side versioning of the BPEL processes. It helps to simultaneously test and compare multiple versions of the flows. The delivery service manages the delivery of SOAP, JMS and email messages to remote destinations. It includes hooks for authentication, encryption and non-repudiation [Collaxa Developer's Guide, 2003].

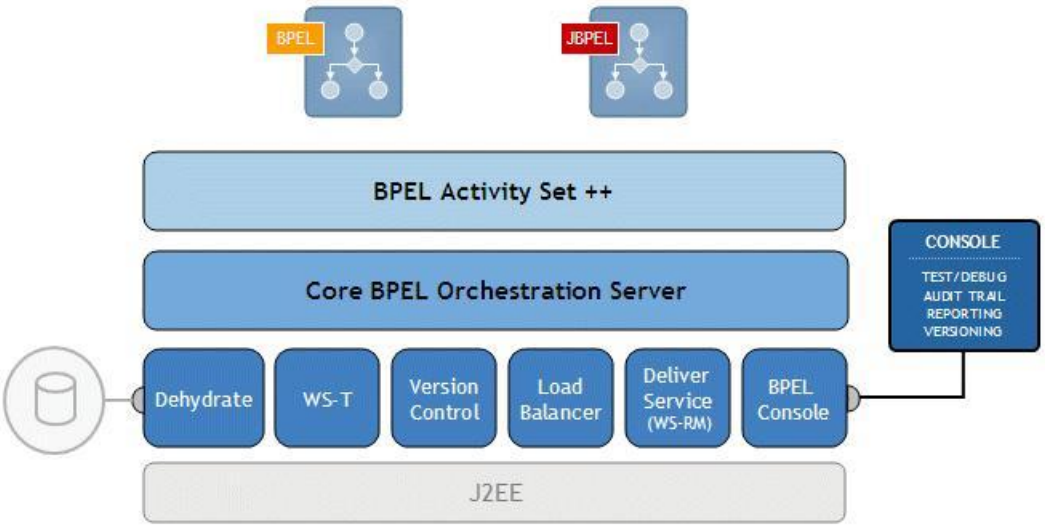


Figure 14 System Architecture [Collaxa Developer's Guide, 2003]

After looking at the system architecture, the following sections will focus on some main features supplied by the server. These are also the basic characteristics required for running Web Service Orchestration and Choreography.

3.4.1. Correlate asynchronous conversation

Task of Web Service Choreography is to assemble and coordinate published service. Since each of the services has its own presentation logic, internal business logic and sophisticated interfaces, the execution of each service may certainly take some time. Frequently, some particular transactions may need manual approval, which depends on the working time of the employees; or according to the server's performances, not all requests can be processed at once so that some will have to be put in a waiting list, and so on. For the long-running execution of processes in a loosely-coupled environment, asynchronous messaging must be supported by the server.

Whether a particular Web service is synchronous or asynchronous is decided by the service provider. For asynchronous services, the BPEL Server provides a listener for the callbacks, in order to take care of correlating the initiation of the service with the result being returned. In the BPEL standard, the <invoke> activity can directly call a synchronous service, while for an asynchronous service, two steps should be taken: one to <invoke> the service and a second to <receive> its result. The <invoke> call returns immediately. The second call to receive the result can be made later. Figure 15 gives an example of two parallel asynchronous invocations of two services.

In the case of such asynchronous communication, the BPEL Server will supply additional information to the provider to facilitate a response from the provider. The protocol used is that specified by the emerging standards supporting Web Services, such as WS-Transaction, WS-Coordination, WS-Addressing, JMS and JCA.

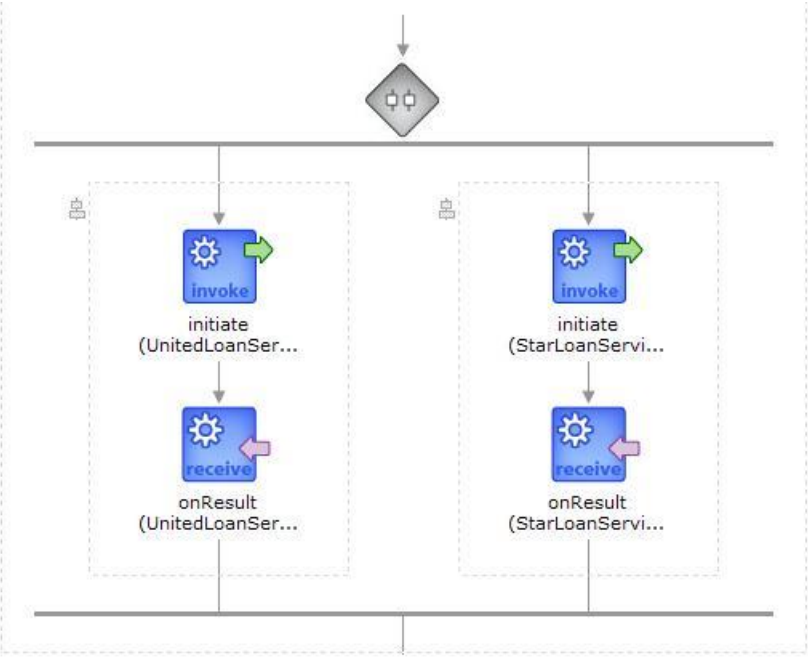


Figure 15 Asynchronous messaging and concurrent activities

3.4.2. Coordinate multi-step flows

The client couldn't be satisfied, if each service in the process could only be invoked one by one in sequence. For instance, the customer who is applying for a Loan may want to try with more providers and compare their rating. This requires branching and join pattern in the process.

Branching in BPEL is realized by nesting the parallel <sequence> activities in a <flow>. As shown in Figure 15, the requests are submitted to the two loan providers, UnitedLoanService and StarLoanService simultaneously. Afterwards, when both of them receive the return value from the service providers, or any or both of them run in timeout, the parallel branches are joined again. Here the comparison of results from two services are made by the <switch> <case> activities.

3.4.3. Dehydrate flow

As talked in section 3.4.1, the flow can not go on to the next step while it is waiting for the asynchronous callback. Thus, it must be avoid that arbitrary code is executed in the interim. The process must be blocked for a while until receipt of the return value. This is achieved by the feature of dehydration.
While the execution of a BPEL instance is waiting for the return value, the Collaxa BPEL Server will dehydrated the flow, so to say passivate it in a database, so that no resources or threads are being held by the waiting flow. In this way, the execution is completely

reliable. The server can be restated or even reboot of the machine will not affect the correct execution of the flow.

After receiving the callback message, the state of the flow is recovered, the process again released and ready to execute the rest part.

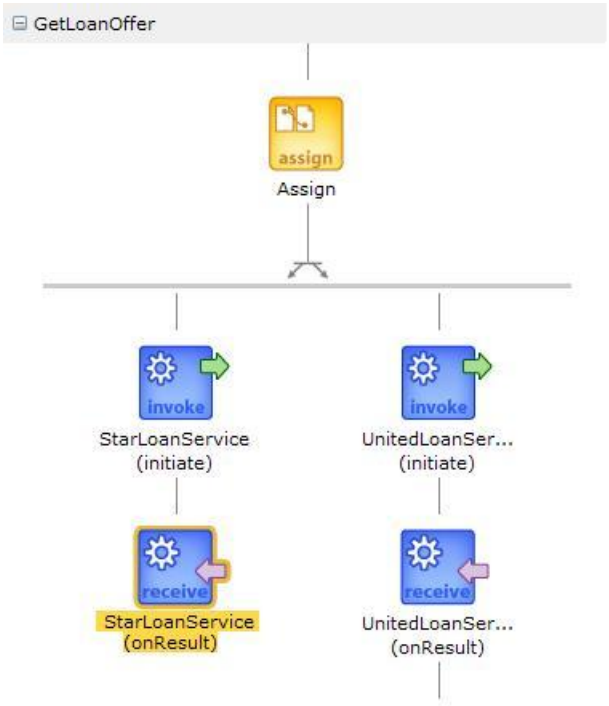


Figure 16 Activity dehydration

3.5. Features not supported by Collaxa

As currently there are many approaches to model BPEL, at least the one using Unified Modelling Language (UML) activity diagram and the one used in Collaxa BPEL Designer [Yuan, 2004]. Since Collaxa has its own modelling notations, it loses the capability to adapt the design phase of a process with existing design of programs using UML. Ideally for the developers if they could construct business processes using software based on the know-how they had, without learning new notations and extra rules. Sonic Orchestration Server, for example, provides this kind of features, so that the UML activity diagram edited in Rational Rose can be directly imported to a business process definition.

Besides, the validation of a process is performed on deployment. Although there might be errors when validating, the process is loaded on the server and could be executed. This is actually an error prone feature. It should be originally a task of the Designer to validate and refine the process, and the server is only responsible for run-time exceptions.

Security is also a concern for web services orchestration and web services in general. It is an important issue because we are now exposing interfaces on a somewhat less-secure

protocol. There are a number of standards being discussed for web services security including Digital Signatures, Encryption and WS-Security. These standards hope to fill specific web services security requirements for the authentication and authorization of users, and for securing the XML message itself. However, the Collaxa products presented in this paper neither claim to offer direct support for security, nor provide any mechanisms through third party software. For example, how do the roles defined for each partner relate the existing authentication and authorization? When should the request and response messages be encrypted and decrypted? At this point, the vendor should consider a more secure solution in the next release.

4. Business Process Modeling and Deployment

4.1. Modeling a process in BPEL Designer

A case study will be introduced in this chapter to illustrate the concept of Web Service Choreography. As we are not going to develop real world business transactions and there are not many free services offered, the following business process is taken as an example of running BPEL using Collaxa software.

Assume there is a business process handling books. The customer may want to query book information and compare the prices from two different service providers. The complete book information is taken from Amazon Web Service (AWS)[1], whereas the BN Quote Service (BNQS)[2] also provides prices of books. Comparison of prices is realized using XPath query in the BPEL flow.

We name the project as AmazonFlow. In Collaxa BPEL Designer, a project consists of at least 5 files [Collaxa Tutorial, 2003]:

AmazonFlow.bpel	The BPEL source for the process. The New Project wizard creates an empty flow, with just the minimum activities and definitions for the selected flow type. For a synchronous BPEL process, the only activities will be a `<receive>` to initiate the flow from a synchronous client request and a `<reply>` to return.
AmazonFlow.wsdl	The WSDL (client) interface for this process. Defines the input and output messages for this flow, the client interface and operations supported, and the BPEL partnerLinkType(s), so that the flow can be incorporated into other processes. The New Project wizard generates a document-literal style WSDL that takes a string input message and returns a string response message. In this case study we should manually modify the input variable to construct request messages for other services.
bpel.xml	The deployment descriptor for the process. Defines the locations of the WSDL files for services called by this flow, along with other project-specific parameters.
build.xml	Apache Ant script for compiling and deploying this process.
.project	Eclipse .project format file.

[1] http://soap.amazon.com/schemas3/AmazonWebServices.wsdl

[2] http://www.xmethods.net/sd/BNQuoteService.wsdl

Besides these, some additional WSDL files are needed to create partner links between the process and the service partners.

4.1.1. The service partners

Amazon Web Services (AWS) provides direct access to Amazon's technology platform[3]. The WSDL of AWS exposes many operations for different searching mechanisms and enabling purchases of products. Using AWS, we can access catalogue data, create and populate an Amazon shopping cart, and even initiate the checkout process.

To invoke the operations of AWS, it is first required to register and obtain a developer's token from AWS. This token is then attached to each request call of the service. In this example, the "AsinSearchRequest" is chosen as search function by giving a book's ISBN and retrieving complete product information including price supplied by Amazon.

The BN Quote Service is a simple Web Service based on the data of "Barnes & Noble" book store[4]. It provides only one operation "getPrice" to get a book's price by giving ISBN.

These two services are added to the flow as partners. For most synchronous or older web services, the service WSDL will not actually include a partnerLinkType since this was added by the BPEL standard (leveraging the WSDL 1.1 extensibility features). In this case, the partnerLinkType has to be created by the developer and the WSDL of the service should be referenced in the local WSDL file.

To get clear how to write a WSDL identifying partners for Amazon Web Service, let's look at the following example:

```
<definitions name="AmazonSearch"
    xmlns:tns="http://soap.amazon.com"
    targetNamespace="http://soap.amazon.com"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
    <import
location="http://soap.amazon.com/schemas3/AmazonWebServices.wsdl"/>
    <plnk:partnerLinkType name="AmazonSearchService">
        <plnk:role name="AmazonSearchServiceProvider">
            <plnk:portType name="tns:AmazonSearchPort"/>
        </plnk:role>
    </plnk:partnerLinkType>
</definitions>
```

Example 1 AmazonSearchServiceRef.wsdl

As we can see, a partner is added to the WSDL file with the role `AmazonSearchServiceProvider` and the original WSDL for Amazon Web Service is imported, so that the BPEL can refer to the operations defined for AWS. A similar WSDL should be defined for BN Quote Service as well. These two files are included in the

[3] http://www.amazon.com

[4] http://www.barnesandnoble.com

BPEL project.

4.1.2. The deployment descriptor

The deployment descriptor for the process defines the locations of the WSDL files for services called by this flow, along with other project-specific parameters. Here is an example for AmazonFlow.

The "id" property defined in the bpel.xml deployment descriptor is used as the name of the deployed process in the BPEL Console and will also be used to name the deployment package jar which bpelc creates.

```
<bpel-process id="AmazonFlow" src="AmazonFlow.bpel" wsdlLocation="AmazonFlow.wsdl">
```

Example 2 Define process id in bpel.xml

The two WSDL locations are defined respectively under id “AmazonSearchService” and “BNQuoteService”. These are the services the process going to invoke. As explained in the last section, we need extra WSDL files to define partner links, therefore the “wsdlLocation” is not directly pointing to the original WSDL of the services, but the reference files.

```
<properties id=″AmazonSearchService">
  <property name="wsdlLocation">AmazonSearchServiceRef.wsdl</property>
  <property name="send-type-attribute">true</property>
</properties>
<properties id="BNQuoteService">
  <property name="wsdlLocation">BNQuoteServiceWrapper.wsdl</property>
</properties>
```

Example 3 Define WSDL locations in bpel.xml

The property “console-testForm” is an optional property used to customize the BPEL console test form. Here for the default input, an AsinSearchRequest is constructed according to AWS’s WSDL. When the AsinSearchRequest operation is invoked, all of the data are required as input request.

```
<properties id="console-testForm">
    <property name="defaultInput">
        <![CDATA[
        <ns1:AsinSearchRequest xsi:type="ns1:AsinRequest"
            xmlns:ns1="http://soap.amazon.com"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <ns1:asin xsi:type="xsd:string">0201633612</ns1:asin>
          <ns1:tag xsi:type="xsd:string">webservices-20</ns1:tag>
          <ns1:type xsi:type="xsd:string">heavy</ns1:type>
          <ns1:devtag xsi:type="xsd:string">D8907FASD398</ns1:devtag>
          <ns1:offer xsi:type="xsd:string">1</ns1:offer>
          <ns1:offerpage xsi:type="xsd:string">1</ns1:offerpage>
          <ns1:local xsi:type="xsd:string">en</ns1:local>
          <ns1:mode xsi:type="xsd:string">book</ns1:mode>
        </ns1:AsinSearchRequest>
       ]]>
    </property>
</properties>
```

Example 4 Define default input request in bepl.xml

It's necessary to explain some of the request data in details:

- asin: The ASIN of a project, here ISBN of the book to be searched.
- tag: Amazon.com enables Web site owners to link to their store and earn referral fees for any sales that are generated through their links. This is done through an associates program. Each associate get an Associates Id, which should be attached to the request in tag element. If no Associates Id, the tag is assigned with value "webservices-20" for invoking AWS in US.
- type: The output type, can be heavy or lite.
- devtag: The developer token, can be obtained when register as developer.
- offer: The offer parameter is used to request and limit the types of offerings returned as part of an ASIN search. 1 represents "ThirdPartyNew".
- offerpage: The offerpage parameter is used to request a particular page of offering information. The default value is 1; 25 offerings are returned per page.
- locale: International locale specifier.
- mode: Product line, can be book or dvd, etc.

4.1.3. The message data type

When creating a BPEL application, the first step is to use XML Schema to define the messages which will be exchanged with the process. The message data types are defined in AmazonFlow.wsdl, which describes the interface of the process.

```
<complexType name="AsinRequest">
  <all>
    <element name="asin" type="string" />
    <element name="tag" type="string" />
    <element name="type" type="string" />
    <element name="devtag" type="string" />
    <element name="offer" type="string" minOccurs="0" />
    <element name="offerpage" type="string" minOccurs="0" />
    <element name="locale" type="string" minOccurs="0" />
    <element name="mode" type="string" minOccurs="0" />
  </all>
</complexType>
<element name="AsinSearchRequest" type="tns:AsinRequest"/>
<element name="onAmazonFlowResult">
   <complexType>
     <sequence>
       <element name="result" type="string"/>
     </sequence>
   </complexType>
 </element>
```

Example 5 Define data types in AmazonFlow.wsdl

As shown in the example above, there are two types of elements defined for messages. One is "AsinSearchRequest", having the same structure as the default input message defined in the deployment descriptor bpel.xml (Example 4). The other element "onAmazonFlowResult" contains only one sub-element "result" for the return value of

the flow.

According to the data type, input and output messages are defined as following:

```
<message name="initiateAmazonFlowSoapRequest">
  <part name="parameters" element="tns:AsinSearchRequest"/>
</message>
<message name="onAmazonFlowResultSoapRequest">
  <part name="parameters" element="tns:onAmazonFlowResult"/>
</message>
```

Example 6 Define messages in AmazonFlow.wsdl

4.1.4. The operations to invoke the process

The AmazonFlow BPEL process will be exposed as a coarse-grained, asynchronous operation that will be initiated through a 1-way operation called `initiate` and will callback the requester through a 1-way callback operation called `onResult`. The input to the flow is a message `initiateAmazonFlowSoapRequest` and the result is message `onAmazonFlowResultSoapRequest`.

```
<portType name="AmazonFlow">
  <operation name="initiate">
    <input message="tns:initiateAmazonFlowSoapRequest"/>
  </operation>
</portType>
<portType name="AmazonFlowCallback">
  <operation name="onResult">
    <input message="tns:onAmazonFlowResultSoapRequest"/>
  </operation>
</portType>
```

Example 7 Define operations in AmazonFlow.wsdl

The client is added in the WSDL file as another partner link, so that the client can act as requestor to invoke both operations defined above and execute the process. The partner link binds the service and requestor portType into an asynchronous conversation.

4.1.5. The global variables

Figure 17 is an overview of the AmazonFlow in Collaxa BPEL Designer. Three service partners are connected to the flow. On the left side is the client, the service requestor initiating the process and receive return value from the asynchronous call back. On the right side are two services, whose operations invoked by the AmazonFlow. If the WSDL location is correctly defined, the Designer automatically loads all operations supplied by these services. As we can see from the figure, BNQuoteService exposes only one operation “getPrice” and AmazonSearchService provides powerful search functions and some other transaction methods.

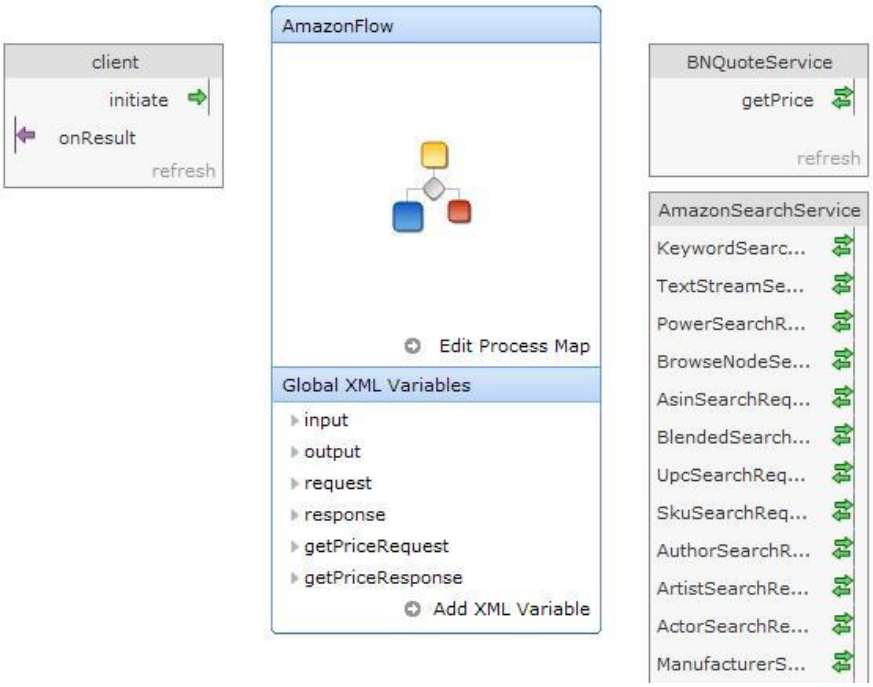


Figure 17 Overview of AmazonFlow

To exchange messages between partners, XML variables are used to store temporary values. By default, the BPEL Designer declares two variables `input` as request to initiate the process and `output` to keep the return value. To contain request and response messages for the two services, two pairs of variables are declared:

```
<variables>
   <variable name="request" messageType="ans:AsinSearchRequest"/>
   <variable name="response" messageType="ans:AsinSearchResponse"/>
   <variable name="getPriceRequest" messageType="BNQuoteService:getPriceRequest"/>
   <variable name="getPriceResponse" messageType="BNQuoteService:getPriceResponse"/>
</variables>
```

Example 8 Declare variables in AmazonFlow.bpel

Since these variables are input and output parameters of the service operations, their types are defined in the corresponding service WSDL files.

4.1.6. Assign variables and invoke services

Once the partner and associated variables have been appropriately declared and configured, we can add the BPEL activities responsible for actually calling the web service.

As we want to query the book's price simultaneously, the two invocations are executed in parallel. In BPEL this is realized by the `<flow>` activity. The scope is similar to a piece of code block in normal programming language, like C or Java.

```
<scope name="getPrices">
    <flow>
        <sequence>… …</sequence>  <!-- invoke getPrice -->
    </flow>
    <flow>
```

```
        <sequence>… …</sequence> <!-- invoke AsinSearchRequest -->
    </flow>
</scope>
```

Example 9 Parallel execution in AmazonFlow.bpel

Figure 18 is the graphic representation of the code in Example 9. Each of the branches is represented by a `<flow>` activity. The sequence inside a flow consists of an `<assign>` and an `<invoke>` activity respectively.

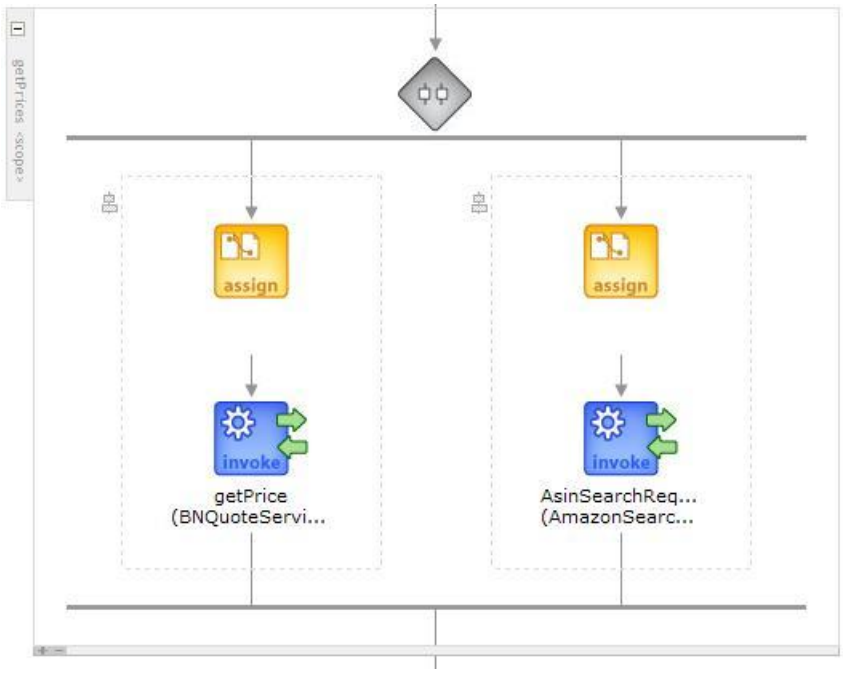


Figure 18 Invoke the services

The assign activities are used to copy part of a message to another message, namely to exchange values between variables. Here for example, in variable `input`, the default input data is sent by the client as request. To invoke BNQuoteService, only the ISBN is required. Therefore the ISBN value needs to be extracted from input request and copied to the variable `getPriceRequest`.

The Collaxa BPEL Designer provides graphic editor to modify the flow model. New copy rules can be added to the `<assign>` activity using "Copy Customizer" (Figure 19). In BPEL, an `<assign>` statement can have many copy rules, each using variable data, XPath queries, XPath expressions, and/or literals to do simple data manipulation and transformation. Each copy rule for an `<assign>` activity has a **From** part, which specifies the source data, and a **To** part, which specifies a variable or element part as the destination for the data.

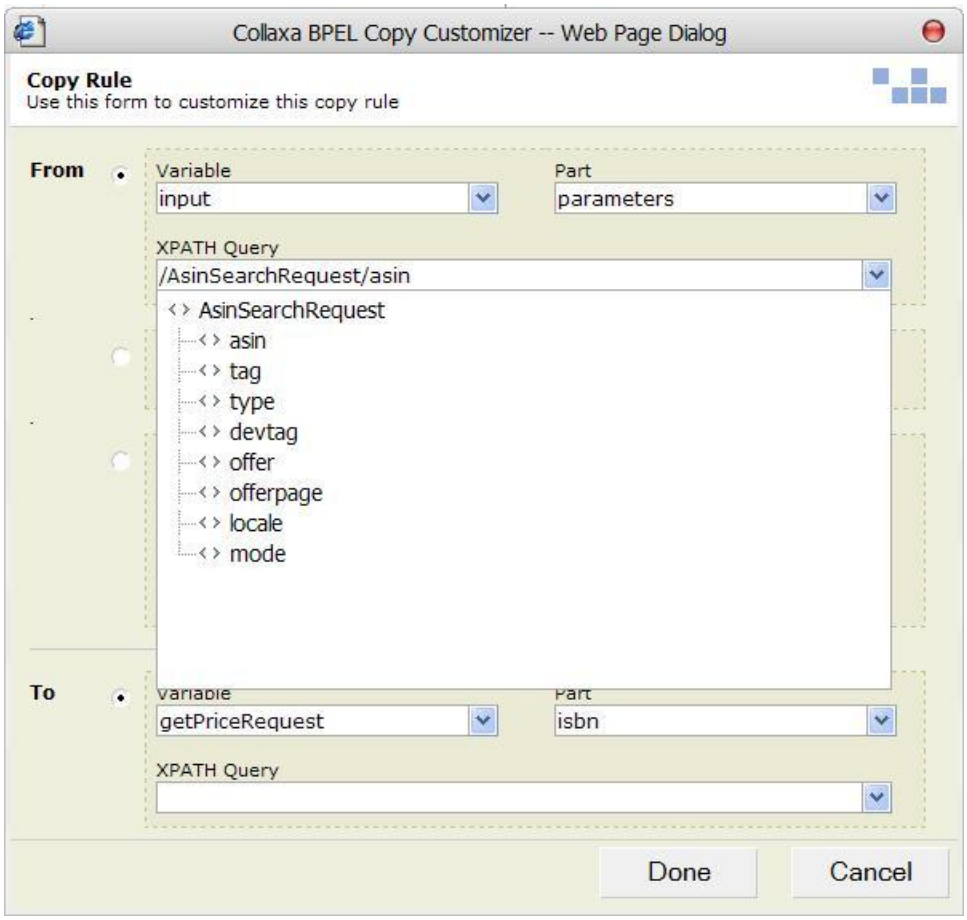


Figure 19 Collaxa BPEL Copy Customizer

Figure 19 is the screen shot of the Copy Customizer, showing the copy rule when assign ISBN to the input request of BNQuoteService. The source variable is "input" and destination is "getPriceRequest". What makes it convenient for developers is that the Customizer can load the structure of the variable in XPath Query field according to its definition in Schema. The developer can just pick up the desired element and the query is automatically filled in.

The attributes of invoke activity can also be edited in graphic model. As partner links and variables are properly configured, they are loaded in the drop down lists. By selecting the corresponding values, the following code is generated automatically.

```
          <invoke name="invokdBNQSearch" partnerLink="BNQuoteService"
portType="BNQuoteService:BNQuotePortType" operation="getPrice"
inputVariable="getPriceRequest" outputVariable="getPriceResponse"/>
          <invoke name="invoke" partnerLink="AmazonSearchService"
portType="ans:AmazonSearchPort" operation="AsinSearchRequest"
inputVariable="request" outputVariable="response"/>
```

Example 10 Invoke services in AmazonFlow.bpel

4.1.7. Compare the prices

The Amazon Search operation provides the complete information about the book, including its price. The BNQuoteService returns the price available in their book store.

To output the lower price to the client, the <switch> - <case> - <otherwise> activities are implemented. If the condition in <case> activity is fulfilled, say, the price get from BNQuoteService is lower than that from Amazon, this price is then copied to the variable "output". Otherwise the price provided by Amazon is assigned.

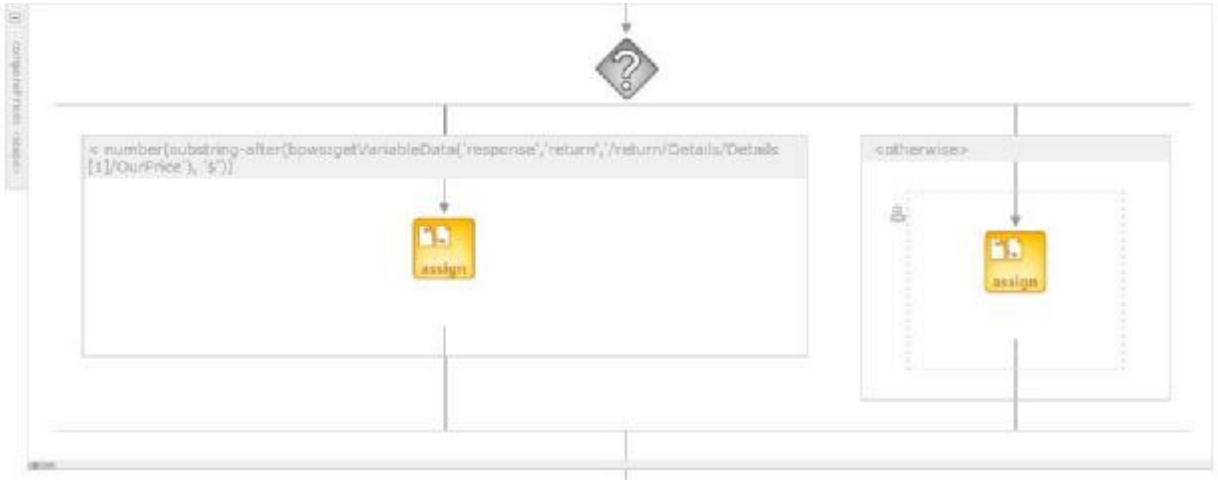


Figure 20 Compare prices scope

Notice that in the schema of these two services, prices are defined in different data types. Amazon's price is of string type and Barnes&Noble's price is float number. To investigate more examples from Amazon, it can be determined that the Amazon's prices are attached with currency sign. Since currently Amazon supplies international Web Services in US, UK, Germany and Japan, the currency varies with the different Services being invoked.

Here to simplify the implementation, it is supposed that we only call the service in US and the currency is "$". The condition attribute in <case> activity is :

```
<case condition="bpws:getVariableData('getPriceResponse','return') &lt;
number(substring-after(bpws:getVariableData('response','return','/retur
n/Details/Details[1]/OurPrice'), '$'))">
```

Example 11 Case activity in AmazonFlow.bpel

Here XPath is used for the expression for case statement. BPEL defines a number of XPath function calls, e.g `getVariableData()`. These are all defined within the BPEL namespace of http://schemas.xmlsoap.org/ws/2002/07/business-process.

To get the return value from BNQuoteService, the first parameter `getPriceResponse` is variable name, and the second `return` is the part where price value is stored. However, to retrieve price from the return value of AmazonSearchService is more complicated. Except the variable and part name as parameters, an additional XPath expression is required to refer to the element `OurPrice`. Because the data returned from Amazon is the whole information about the book, while what we need is only the price supplied by Amazon.

The XPath function `substring-after()`removes the "$" sign in front of the number, so that a string represented number is returned. To convert the string to float, another XPath

function `number()` is invoked.

Finally, as this statement is in an XML context, the "<" sign should be substituted by "`<`".

An extension to this implementation could be, invoking another Web Service for currency conversion before comparing prices. This will better support internationalization of the application and carry out the Web Service Choreography between multiple participants in run time.

4.2. Deploy and execute the process on the BPEL Server

As introduced in section 3.2, after design of the process, the BPEL project can be compiled and deployed on the Collaxa BPEL Server using command "cxant". After successful deployment, the AmazonFlow will appear in the Collaxa console. If the default input value (Example 4) is submitted to initiate the flow, the process is supposed to get the price of the book `Design Patterns` with the ISBN 0201633612.

When clicking the flow audit of the BPEL Console, we can see that the process is completely executed, because the invoked operations are both synchronous and the response is simultaneously returned.

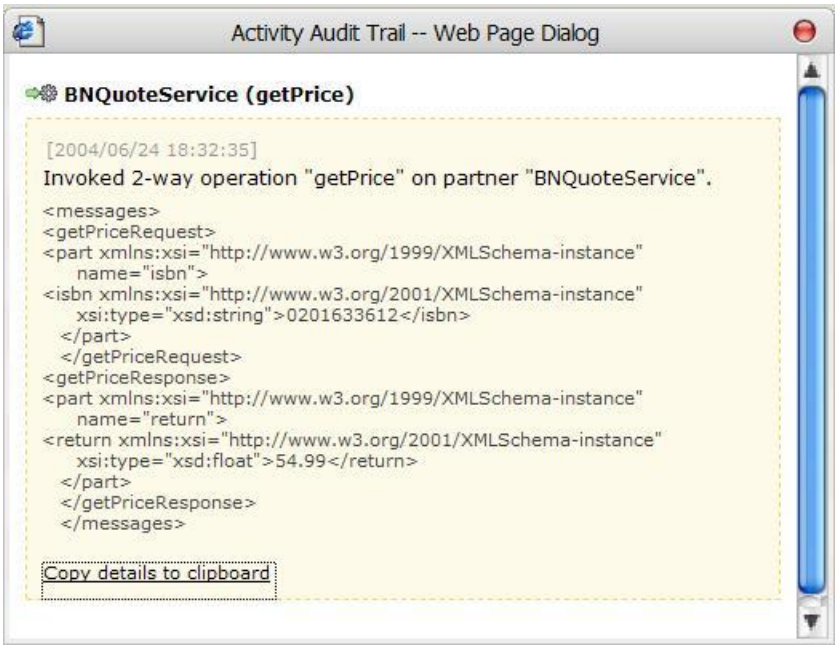


Figure 21 getPrice Request & Response

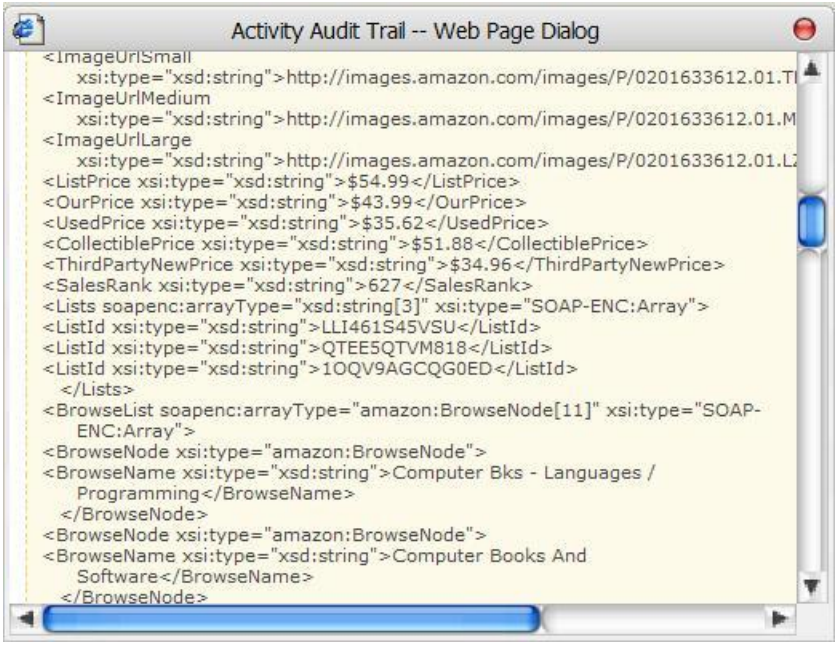


Figure 22 AsinSearch Request & Response

Figure 21 and Figure 22 are the audit trails from the two services respectively. When executing the operation **getPrice**, we can see ISBN is sent as request and the return value is the price 54.99 as float number. Whereas the response of operation **AsinSearchRequest** contains the entire information of the book, even with three prices: *ListPrice*, *OurPrice* and *UsedPrice*. *ListPrice* $54.99 is the same value as got from BNQuoteService, and *OurPrice* $43.99 is the price provided by Amazon with discount. Therefore, the result of comparison should be $43.99, as shown in Figure 23.

Figure 23 Result of price comparison

To search for another book, we can initiate a new instance of the flow and change the default ISBN by another one. In Collaxa Console, all the instances are numbered and listed under the **Instances** tab so that they can be reviewed and managed later.

Although not further investigated here, the scenario of the flow could be extended by building upon this basic process. For instance, if the price provided by `Amazon` is lower

than that of `Barnes&Noble`, the client can decide to get details of the customer reviews of that book and add the book to Amazon's shopping cart, or invoke some other service rather than Amazon.

This illustrates the idea of Web Service Choreography and Service Oriented Architecture, which allows Web services to be integrated and sequenced programmatically, with the result of one Web service affecting which action or Web service is going to be performed next.

5. Conclusion

5.1. Summary

The ultimate goal of integrating applications is business acceleration, which enables the agility and efficiency to response accurate information. This certainly rewards the companies increased business opportunities and profits. What makes the integration especially complicated is that the enterprise applications very often cross physical and organizational boundaries, therefore making the traditional business process management difficult. Since the emerging technology of Web Service Choreography results solution of interaction between applications by message exchange in XML format, the companies can now move away from expensive legacy connectors and adaptor solutions to Web Services.

This paper concludes with an outlook of emerging technology of Web Service Choreography and Orchestration, their technical requirements, tools supporting design and implementation of business processes and real world scenarios.

Web Service Choreography and Orchestration means to assemble and coordinate services, from both internal enterprise components and external trading partners, into a manageable business application. Implementing, executing and managing orchestration logic demands a set of infrastructure-level requirements.

- First, the support of coordination between loosely-coupled services and asynchronous messaging.
- Second, management of the process, error and events handling, transactions and compensations, etc.
- Third, monitoring the execution, to view the message exchange and report of the results.

To assist design of orchestration logic and simplify the implementation, a set of products are developed. Although all of the software provided by different vendors varies with some features and representation, the main functionalities are intended to fulfill these requirements above. Thus, they all have a graphical editor for modeling the process, a runtime engine to deploy and run the process and observer software to manage and monitor the process execution.

In this paper Collaxa product is taken as an example and are tested by developing and deploying a real world business scenario on it. Using Collaxa Designer, business analysts gather information required to model and design the business process the enterprise is planning to automate. The compiler can specify a version number so that different versions are kept for developers to fine tuning the process. During the implementation stage the runtime environment is set up and the project is deployed on the Collaxa Orchestration Server. After deployment, a test interface is automatically generated in

Collaxa Console. The process can be initiated and the observation is possible in three viewpoints: view visual flow (graphical audit trail), audit the instance (the text audit trail) or debug the instance (the source code).

The case study presented in this paper, is a simple business process integrating two Web Service partners to retrieve a book's information and compare prices supplied by these two vendors. It supports concurrent execution by branching and joining the process. It also introduces some XPath methods used in queries for assignment of BPEL variables. Last but not least, this case study gives a complete example of design and implementation phases of a simple BPEL project using Collaxa software.

5.2. Troubles encountered

At beginning of the project work, the Microsoft MapPoint Web Service[5] was considered to be illustrated in the scenario. It is a hosted, programmable XML Web Service for integrating maps, driving directions, distance calculations, proximity searches and other location intelligence into applications, business processes, and Web sites. Especially the Location Service based on the MapPoint Location Server is investigated, since it is very applicable in the mobile industry. This service contains methods that can be used to locate the geographic position of a user carrying a locatable device such as a cell phone or pager. It also provides methods for managing the personal preferences of users, such as culture, visibility settings, and contact lists.

Although the initial aim was to use MapPoint as case study, it was determined that the MapPoint Location Server has very high software and hardware requirements, and that the web service itself is not offered for free but through an evaluation account managed in a very sophisticated manner. Although being left aside because of these reasons, MapPoint Web Service is a well developed mature service that is worth of investigation.

Since Amazon Web Service is one of the earliest services that have been tested by number of developers, being basically free, it was finally chosen to be part of the case study. AWS supplies very powerful searches and enables transactions as well. The request and response can be delivered via either REST or SOAP protocol. In this paper, the examples are all based on SOAP messaging. While in AWS APIs, it is not clearly documented how the request invoking a transactional operation is constructed. Some sample SOAP files for executing the search methods are well defined, whereas those for handling shopping cart and payment are still missing.

5.3. Teaching ware for Collaxa products

A collateral product of this project work is a set of Flash demos created as teaching wares. Therefore, the achievement of the project can be reviewed not only by text report, but also by vivid animations, consisting of screen shots, sounds, captions and interactions. The traditional video format is nowadays substituted by small Flash files, which

[5] http://www.microsoft.com/mappoint/webservice/default.mspx

contributes mostly to the modern E-Learning area.

The tool used to create the Flash files is RoboDemo[6] developed by Macromedia. The user can choose automatic recording or manual recording mode, and the frame size can be set to fit a running application, or full screen, or customized size.

Figure 24 shows the RoboDemo movie project when editing the Flash demo for Collaxa Console. All frames, mouse motions, sounds and captions can be edited conveniently using a toolbar. Finally the movie can be exported to different format of flash files for Windows, Macintosh and Linux systems. It is also possible to export the frames in Microsoft Word and can be used to prepare presentation directly. In this project the demos are exported as .swf files.

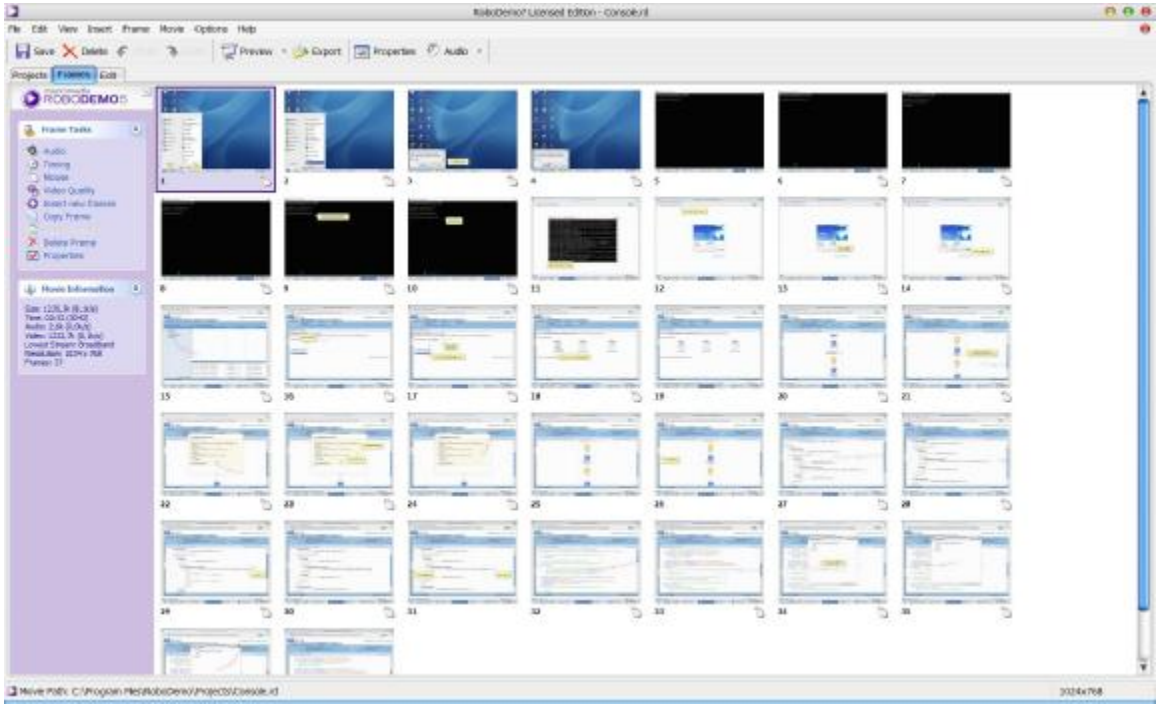


Figure 24 RoboDemo

5.4. Future work

Collaxa also provides many other features, which are not explained in details in this paper. From a developer's point of view, for example, it is an important strength that the <exec> activity enables Java/J2EE code to be invoked directly from a BPEL process. There is another important feature supplying user tasks for management of business processes. Collaxa provides a built-in user TaskService as a web service, which supports human intervention in a BPEL process using standard BPEL asynchronous activities: <invoke> and <receive>. These are to be further inspected and tested when implementing more sophisticated orchestration logic.

[6] http://www.macromedia.com/software/robodemo/

The other vendors also provide powerful software supporting Web Service Choreography and Orchestration. For example, the Sonic Orchestration Server, integrated with Sonic ESB (Enterprise Service Bus), seems to be a big competitor of Collaxa. It should be an interesting topic to compare and evaluate the features of these products along different dimensions.

References

Collaxa BPEL Getting Started Guide (2004)
http://www.collaxa.com/pdf/collaxa-d5.pdf

Collaxa BPEL Hello World Tutorial (2003)
http://www.collaxa.com/tutorials/BPEL-HelloWorldTutorial.pdf

Collaxa Developer's Guide (2003) *Programming BPEL Version 10.15.3*
http://www.collaxa.com/pdf/cx-bpel-developer-20.pdf

Greenwood M. (March, 2003) *Web Services and Process Modeling*
http://www.cs.man.ac.uk/ipg/CS637/WSandPM.pdf

McDonald C. (2003) *Orchestration, Choreography and Collaboration*
http://java.sun.com/developer/onlineTraining/webcasts/pdf/35plus/cmcdonald2.pdf

Peltz C. (July, 2003) *Web Service Orchestration and Choreography* Web Service Journal Volume 3 Issue 7

Rhody S. (July, 2003) *Dance Lessons* Web Service Journal Volume 3 Issue 7

Virdell M. (January, 2003) *Business processes and workflow in the Web services world*
http://www-106.ibm.com/developerworks/webservices/library/ws-work.html

Yuan K. (April, 20034 *Investigation of BPEL Modelling*
http://www.ti5.tu-harburg.de/publication/2004/Thesis/Yuan04/kai_yuan_projectWork_report.pdf