DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS C
REPORT C-2003-74

Using BPEL4WS for Supply-Chain Integration - Experiences and Evaluation

Juha-Pekka Haataja, University of Helsinki (ed.)

Renne Tergujeff, VTT Information Technology

UNIVERSITY OF HELSINKI
FINLAND

Contact information

Postal address:
Department of Computer Science
P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
Finland

Email address: postmaster@cs.Helsinki.FI (Internet)

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 1911

Telefax: +358 9 191 44441

Helsinki 2003

DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS C
REPORT C-2003-74

Using BPEL4WS for Supply-Chain Integration - Experiences and Evaluation

Juha-Pekka Haataja, University of Helsinki (ed.)
Renne Tergujeff, VTT Information Technology

UNIVERSITY OF HELSINKI
FINLAND

Using BPEL4WS for Supply-Chain Integration - Experiences and Evaluation

Juha-Pekka Haataja, University of Helsinki (ed.)
Renne Tergujeff, VTT Information Technology

Department of Computer Science
P.O. Box 26, FIN-00014 University of Helsinki, Finland
juha.haataja@cs.helsinki.fi

VTT Information Technology
P.O. Box 1201, FIN-02044 VTT, Finland
renne.tergujeff@vtt.fi

Technical report, Series of Publications C, Report C-2003-74
Helsinki, December 2003, iii + 23 pages

Abstract

The major future challenges of e-business are in the field of business to business integration (B2B). Important aspects of B2B are business process automation, supply-chain integration and support for automated logistics management. XML based Web services are aiming at becoming the de facto B2B integration solution.

In the Web-Pilarcos project a case study was performed, modelling and implementing a realistic supply-chain integration scenario and testing the usability of BPEL4WS specification. The case study was implemented using Apache Tomcat, Apache Axis and IBM alphaWorks BPWS4J platforms.

BPEL4WS was found to be well suited for composing Web services into executable business processes, but somewhat lacking in its ability to publish abstract processes. The specification has only recently been presented to a standards organisation and will undoubtedly mature. The evaluated BPWS4J platform proved to be usable, but to contain some serious deficiencies as well. Commercial BPEL4WS platforms are arriving, but no open source support is yet available.

In order for Web services to become widely utilised the support for business process management as well as transactions, reliability, and security must be further addressed.

General Terms:
Design, Documentation, Experimentation

Additional Key Words and Phrases:
BPEL4WS, BPWS4J, Business Process Integration, Logistics, SOAP, Supply-Chain Integration, Web-Pilarcos, Web Services, WSDL

Contents

Chapter 1

Introduction

In recent years a growing number of Web services technologies have been developed. This work has been done in order to upgrade existing Web infrastructure to support machine to machine (M2M) integration. One important subset of M2M is the integration of information systems of autonomous organisations ("B2B integration").

The Web services effort approaches M2M integration by producing a growing layered set of protocols and specifications which try to tackle the different interoperability problems. The fundamental building block in the Web services specifications is a stateless messaging protocol, called SOAP [24]. SOAP is capable of transporting XML based messages on top of de facto Web protocols (like HTTP and SMTP). Another very important building block is a service interface description protocol, called Web Services Description Language (WSDL) [31]. WSDL is capable of describing the supported messages and interfaces ("portTypes") of a Web service as well as their binding to underlying Web protocols. In addition to these core specifications there are many others, building on their foundation. One of the newer and a widely noticed specification is BPEL4WS, or the Business Process Execution Language for Web Services [3]. BPEL4WS can be used for describing processes that combine several WSDL described Web services with some inner logic. The composed functionality can be published as a WSDL described Web Service itself. Figure 1.1 presents one approach for describing Web services stack and the positioning of SOAP, WSDL, and BPEL4WS in relation to it.

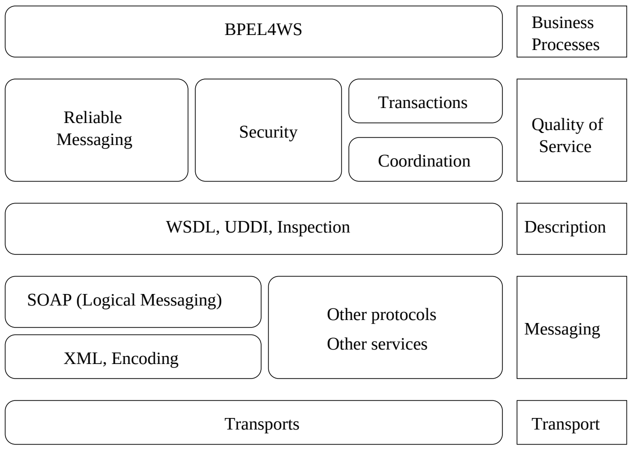

Figure 1.1: Web services stack [22].

The Web-Pilarcos project [28] studies the usability of Web services in B2B integration. In the first phase of the project the usability of core Web service specifications is evaluated by implementing a set of case studies. In the first case study the basic Web service standards (SOAP and WSDL) were briefly experimented with. In the second case study a more advanced approach was taken and, in addition to using SOAP and WSDL, a process oriented integration scenario was designed and implemented using the BPEL4WS language.

The implemented case evolved around a supply chain integration scenario which involved three domains, each representing an autonomous organisation (Buyer domain, Seller domain, and Logistics operator domain). In each domain there were several primitive, asynchronously communicating, Web services (implemented in Java) as well as a BPEL4WS process which defined the business process of the organisation. The BPEL4WS processes at each domain then communicated together by sending and receiving SOAP messages.

The main goals of the second case study were to get practical experience of (1) BPEL4WS language, (2) selected BPEL4WS platform, (3) implementing Java based Web services, and (4) modelling process oriented B2B architectures. The experiences gained should provide input for the development of Web-Pilarcos architecture and set the stage for future case studies.

This report describes the architecture and experiences of this second Web-Pilarcos case study. The report is divided into introduction chapter, three main chapters, and conclusions chapter. Of the three main chapters the first one introduces the designed process and platform architectures, the second one concentrates on evaluating the features of the BPEL4WS specification, and the third chapter summarises the experiences from the used platforms. The conclusions chapter then provides some discussion and concludes the report.

Chapter 2

Goals and Architecture

As already mentioned in the introductory chapter the main goals of this case study can be divided into four categories,

- getting practical experience of BPEL4WS language and its usability,
- getting practical experience of a selected BPEL4WS platform and its usability,
- getting practical experience of implementing Java based (asynchronously communicating) Web services (where Web service means a WSDL described and SOAP accessible service),
- getting experience and understanding on modelling interoperable B2B architectures using a process oriented approach.

The study was performed by defining and implementing a supply-chain integration scenario. The scenario was chosen so that it adequately closely described a supposed real life situation and also allowed for a thorough evaluation of the language features. In this chapter we provide a closer view of the scenario and the related process and platform architectures. It should be noted, however, that the case is demonstrative in nature and an actual real world scenario might or might not resemble the one described here.

2.1 Process Architecture

One of the key ideas behind process oriented integration is that business functions in separate domains are not connected by a complex instance level integration network. Instead the business functions are connected via a business process/workflow.

A business process combines a set of business functions into a new service by defining a workflow and an access interface to the workflow. When this is done in each domain we are left with a situation where only the business processes in each domain must be able to interoperate together and not the larger set of individual business functions. This not only simplifies the integration problem but also decouples the development of internal business functions from the development of the organisations business processes.

It should be noted here that using process oriented integration architecture does not necessarily mandate using business process definition/execution languages but these languages are often a good choice because they are designed for the purpose.

In the next subsections we explain the global process architecture from static and dynamic views. Static view shows the business functions, business processes, and their relationships as well as the human-to-machine and human-to-human interfaces that would exist in a real world scenario. Dynamic view is divided into abstract (business protocols) and executable (business processes) subviews which together cover the business workflow within and between the interacting business domains.

2.1.1 Static view of process architecture

The static view of the process architecture is shown in Figure 2.1. The figure shows three business domains: a Buyer domain, a Seller domain, and a Logistics operator domain. The business motivation of the Buyer domain is to purchase items from the Seller. The business motivation of the Seller domain is to sell items to the Buyer. The business motivation of the Logistics operator domain is to sell transport services (e.g. to receive a payment for transporting items from Seller domain to Buyer domain).

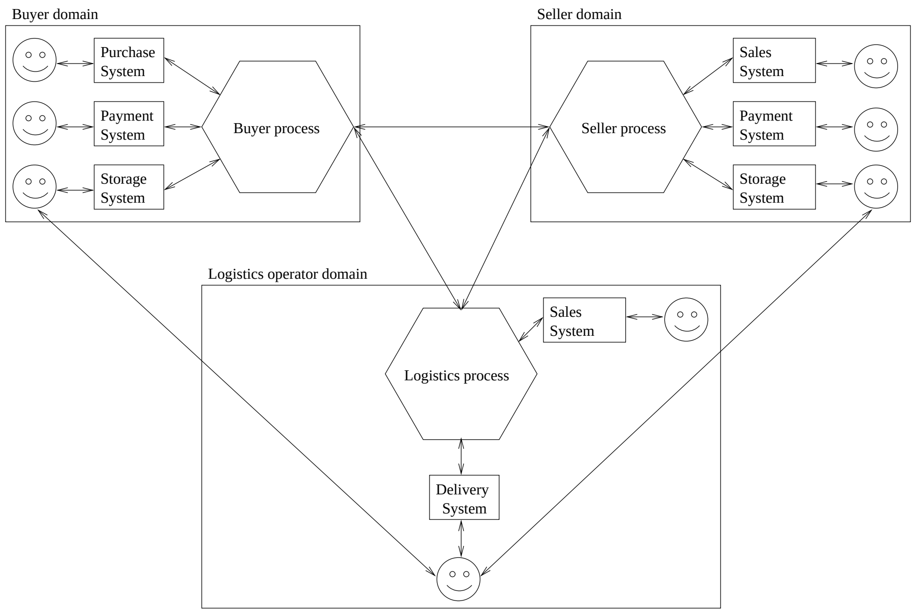

Figure 2.1: Static view of process architecture.

Buyer domain contains three internal business functions: a purchase system for initiating purchases, a payment system for initiating bill payments, and a storage (or inventory) system for managing the warehouses of the company. In addition to the business functions the Buyer domain contains the Buyer process which specifies the workflow used when managing purchases and related transports.

Seller domain contains three internal business functions. From those functions the payment system and storage system are identical to those in Buyer domain. However, instead of having a purchase system in the Seller domain there exists a sales system which is used for handling purchase orders coming from the buyer. There exists no specific purchase system in the Seller domain although the Seller makes purchases from the Logistics operator domain (it buys transport for the items it has sold). The Seller to Logistics operator purchase procedure is statically embedded in the Seller process. In a more dynamic scenario the Seller process might use an additional internal purchase system (possibly with human intervention) for initiating the transport purchase.

Logistics operator domain contains two internal business functions: a sales system identical to the one in Seller domain and a delivery system responsible for scheduling the transports (vehicles, drivers, pick up times, ...). It is also good to notice that there exists a human-to-human interface between the Logistics operator delivery system and the two other domains. This represents the fact that someone has to actually pick up the sold items from the Seller domain warehouse and physically deliver them to the Buyer domain warehouse. This process of pick up and delivery may involve human to human communication and signing of legally compulsory documents and receipts.

In the implemented scenario any actual human-to-human communication did not take place. Instead the human-to-human connections were simulated by additional Web service interfaces between the delivery system and the storage systems. This enabled for the delivery system to notify item pick ups and deliveries to the storage systems as if they were mediated by persons.

2.1.2 Dynamic view of process architecture

The dynamic view of the process architecture can be visualised in two separate ways. The first way is an external viewpoint where the used business protocols and business protocol compositions (or business roles) are shown. Figure 2.2 shows this kind of view of the dynamic relationship between business domains. The figure shows all the three business roles involved: Buyer, Seller, and Logistics operator.

It can be seen that between Buyer and Seller roles there exist two business protocols: purchase protocol and payment notification protocol. Purchase protocol consists of two activities: sending a purchase order and receiving an invoice. The payment notification protocol consists of one activity which is sending a notification of the committed payment transaction. Between Seller and Logistics operator there exist the same basic protocols. Between Logistics operator and Buyer there exists one business protocol called pick up notification protocol. This protocol defines how to notify the Buyer about the logistics/transport situation (e.g. when the delivery is picked up from the Seller).

The business protocols are composed to an integrated workflow (i.e. a business process) in each domain. For example in Buyer domain Figure 2.2 shows that the "legal" business process supported by the Buyer must first execute the purchase protocol after which the payment notification protocol and pick up notification protocol must be executed (but execution of the latter two can occur in any order).

Another approach to visualise the dynamic view is to show the inter- and intraoperation of the business domains at workflow level. This can be done by spelling out in a workflow diagram the actual activities performed by the business processes. This workflow diagram (Figure 2.3) shows all the interacting processes and business functions and the SOAP interactions between them. Each SOAP interaction is marked with a horizontal arrow and can be either an asynchronous request or a request-reply interaction where the request carries the actual business data and the reply is a notification message with no payload. The latter approach was used in the implementation because of platform limitations. In addition to the SOAP interactions, the figure also shows the delivery of the physical merchandise (marked with a larger arrow). In real world the delivery would happen between persons but in the implemented case it is simulated with a SOAP interaction as previously explained.

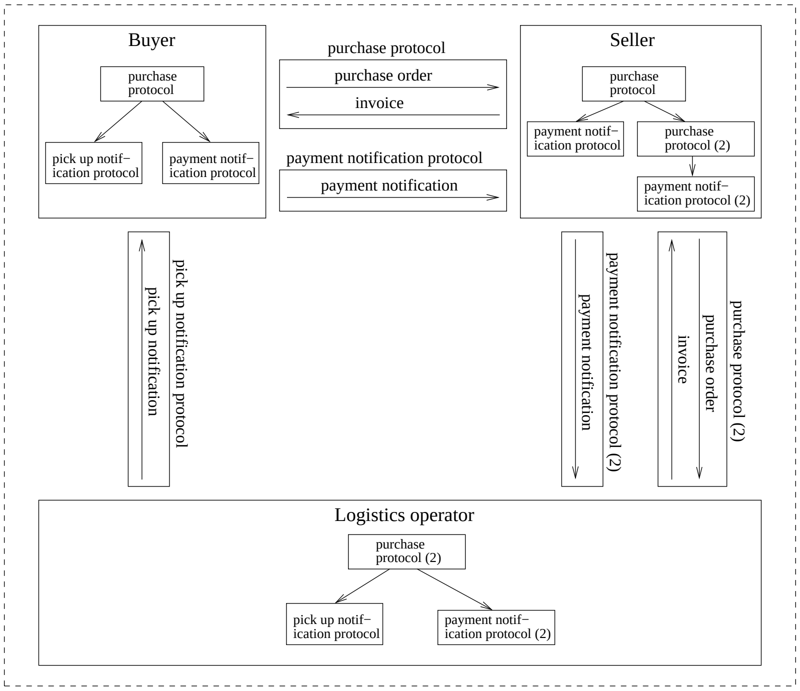
Buyer
purchase protocol
pick up notification protocol
payment notification protocol
purchase protocol
purchase order
invoice
payment notification protocol
payment notification
Seller
purchase protocol
payment notification protocol
purchase protocol (2)
payment notification protocol (2)
pick up notification protocol
pick up notification
payment notification protocol (2)
payment notification
purchase protocol (2)
purchase order
invoice
Logistics operator
purchase protocol (2)
pick up notification protocol
payment notification protocol (2)

Figure 2.2: Dynamic view of process architecture: business roles and protocols.

The workflow in Figure 2.3 shows a typical purchase scenario where the Buyer makes a purchase from the seller and the Seller makes a transport purchase from the Logistics operator. In the figure the scenario is sequential but in reality some concurrency is implemented in the control flow, within the limits placed by the protocol compositions shown in Figure 2.2.

In addition to the actual purchases the dynamic scenario includes the messaging between the internal business functions and business processes as well as some after purchase notifications between the domains (for example notification of transport pick up).

It should be noted here that the actual SOAP messages sent between the domains contain more parameters than only the basic business documents. They may contain, for example, ids identifying the contracts within which the interactions take place as well as some other ids. These additional ids are mostly used for management purposes. These technical details are, however, not essential to understand the modelled business scenario.

2.2 Platform Architecture

The platform architecture in the case study was distributed in three physical hosts (one host for each business domain in the process architecture). Each host was running a (Red Hat) Linux operating system.

Two of the hosts were located at the University of Helsinki, Department of Computer Science and one host was located at VTT. The hosts at the University were connected through the local intranet. Between University and VTT there were two firewalls at the University side (department firewall and university firewall) and a firewall at the VTT side. The University and VTT networks were connected by the public Internet. Figure 2.4 shows the platform distribution architecture. The traffic at VTT side was routed through a proxy server which is not visible in the figure.

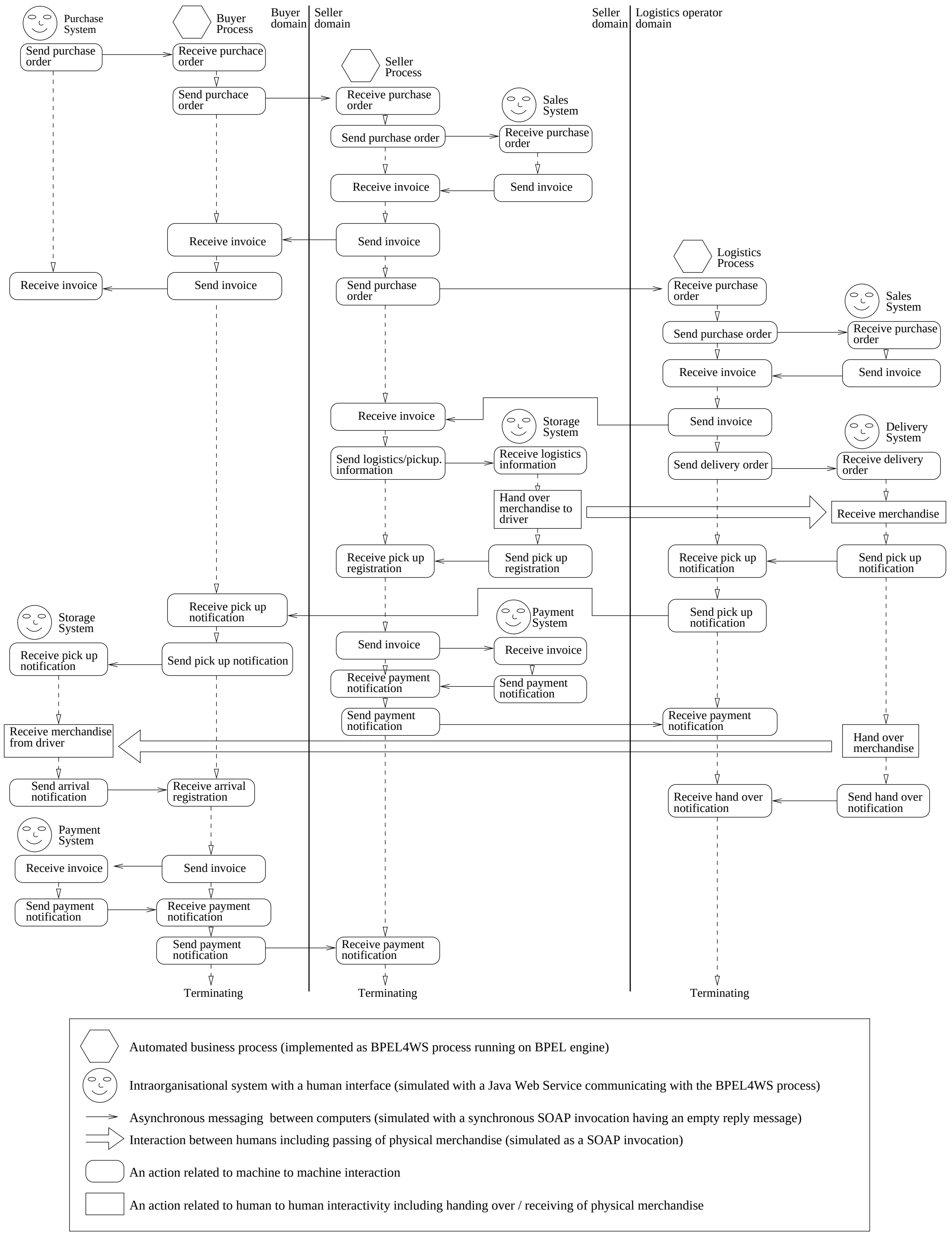
Purchase System
Buyer Process
Buyer domain
Seller domain
Seller domain
Logistics operator domain
Send purchase order
Receive purchace order
Seller Process
Send purchace order
Receive purchase order
Sales System
Send purchase order
Receive purchase order
Receive invoice
Send invoice
Receive invoice
Send invoice
Logistics Process
Receive invoice
Send invoice
Send purchase order
Receive purchase order
Sales System
Send purchase order
Receive purchase order
Receive invoice
Send invoice
Receive invoice
Storage System
Send invoice
Delivery System
Send logistics/pickup. information
Receive logistics information
Send delivery order
Receive delivery order
Hand over merchandise to driver
Receive merchandise
Receive pick up registration
Send pick up registration
Receive pick up notification
Send pick up notification
Receive pick up notification
Payment System
Send pick up notification
Storage System
Receive pick up notification
Send pick up notification
Send invoice
Receive invoice
Receive payment notification
Send payment notification
Send payment notification
Receive payment notification
Receive merchandise from driver
Hand over merchandise
Send arrival notification
Receive arrival registration
Receive hand over notification
Send hand over notification
Payment System
Receive invoice
Send invoice
Send payment notification
Receive payment notification
Send payment notification
Receive payment notification
Terminating
Terminating
Terminating
Automated business process (implemented as BPEL4WS process running on BPEL engine)
Intraorganisational system with a human interface (simulated with a Java Web Service communicating with the BPEL4WS process)
Asynchronous messaging between computers (simulated with a synchronous SOAP invocation having an empty reply message)
Interaction between humans including passing of physical merchandise (simulated as a SOAP invocation)
An action related to machine to machine interaction
An action related to human to human interactivity including handing over / receiving of physical merchandise

Figure 2.3: Dynamic view of process architecture: business workflows

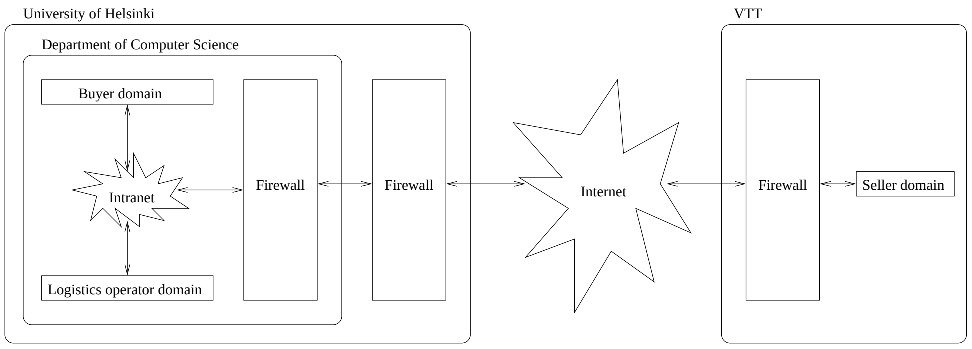

Figure 2.4: Domains and Distribution Architecture.

Each of the hosts supported a similar platform configuration where Apache Tomcat [2] was used as the servlet engine and Apache Axis [1] was used as the Java based Web services platform (supporting SOAP and WSDL). In addition the IBM alphaWorks BPWS4J [11] engine was used to host and execute the BPEL4WS processes. Figure 2.5 shows the platform configuration in Buyer domain. The figure makes visible the platforms, business functions, and business processes located therein.

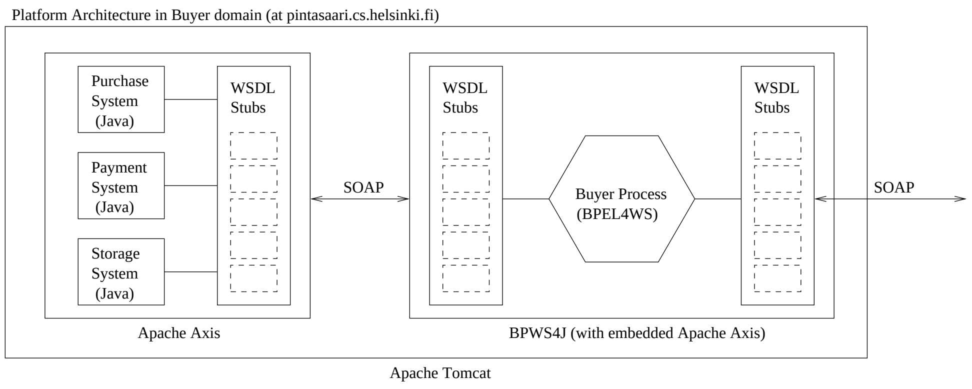

Figure 2.5: Platform Architecture in Buyer domain.

Chapter 3

Evaluation of BPEL4WS

In the Web services world where SOAP is the enabling technology in the messaging layer and WSDL is the one in the interface layer the role of the BPEL4WS specification is to build on these technologies and enable the definition and execution of automated business processes.

BPEL4WS 1.0 specification [3] was originally developed as a proprietary specification by a closed group of companies. Especially IBM was a major contributor. For this reason IBM has produced a lot of support material on BPEL4WS and its usage [6] as well as the first implementation of a BPEL4WS execution engine [11]. The BPEL4WS specification was later given to OASIS for standardisation and further development. In the OASIS the work is carried out in the WS-BPEL Technical Committee [19] which was founded in April 2003. BEA, IBM, Microsoft, SAP and Siebel submitted an updated version of the specification to WS-BPEL TC which then became BPEL4WS version 1.1 and was used as the basis for further development.

In this chapter we will first look at the initial goals of the BPEL4WS specification, then go through the features of a BPEL4WS process and finally evaluate the usability and limitations of the specification in modelling the kind of integration architectures described in previous chapter.

3.1 Goals of BPEL4WS

In order to understand the philosophy behind BPEL4WS we should first take a closer look at its initial design goals. A good reference material for this is the note to WS-BPEL Technical Committee [13] from the original authors of the BPEL4WS specification. In the note the authors list the ten most important design goals of the BPEL4WS specification. We will shortly summarise them here.

Goal 1 is to base BPEL4WS firmly in the Web services world. This is achieved via a tight binding with WSDL. A BPEL4WS process sees external entities through WSDL portTypes and external entities see BPEL4WS through a WSDL portType. Binding and deployment issues (the "grounding" of the portType) should be left out of the scope of the specification as far as possible.

Goal 2 is to use XML as the process definition language. BPEL4WS only defines the XML schemas but does not mandate the usage of any graphical notation or design methodology.

Goal 3 is to provide a rich set of Web services orchestration concepts which can be used by both external ("abstract") and internal ("executable") process descriptions. Both views of the process should use the same core concepts and use only a minimum amount of specialised extensions. The orchestration should be specified separately from each partners point of view. This eases up the specification of data dependent behaviour in business protocols. The experiences with EDI and RosettaNet further verify this approach.

Goal 4 is for BPEL4WS to support hierarchical and graph like control of workflows and allow them to be blended seamlessly. The hierarchical model is a legacy of the Microsoft's XLANG specification [41] and the graph like control is a legacy of the IBM's WSFL specification [32]. BPEL4WS tries to supersede both specifications and attract their developer bases.

Goal 5 is to support limited capabilities for data manipulation. The data manipulation functions should provide support for the definition of process related data and control flows. The more advanced manipulation of data should be kept to the actual business functions and not included in the business process.

Goal 6 is for BPEL4WS to support an instance correlation mechanism that allows instance identifiers to be defined at the application message level and allows them to change over time. The reasons to define the instance correlation at the application message level is to promote the idea of business process as a business artifact and provide a binding independent instance correlation scheme. It is also important to enable different participants in the process to define their own instance correlation ids in order to enable a loosely coupled relationships between different partners.

Goal 7 is to support implicit life cycle management of process instances. At the minimum the creation and termination functions should be supported but in future releases more advanced functions, like suspend and resume, may be added. The implicit life cycle management is needed in order to provide a unified usage model for both stateless and stateful Web services (i.e. the process implementer does not have to specify or know whether the targeted service is stateless or stateful). This contrasts to the explicit factory pattern often used in distributed object middleware.

Goal 8 is to support a long-running transaction model that is based on proved techniques. These techniques include scoping and subscoping of the process and scope specific compensation handlers. In a long-running transaction it is desirable to be able to support backward recovery in small units in order to avoid global rollbacks (which may even be impossible to perform).

Goal 9 is for BPEL4WS to use Web services as the model for process composition and assembly. The Web services grounding of BPEL4WS together with the implicit life cycle management enable a recursive and flexible aggregation model. This approach can be combined with WS-Policy [33] statements which further specify the dependencies of Web services.

Goal 10 is for BPEL4WS to be compatible with and build on existing Web Services standards and standard proposals. BPEL4WS should concentrate on defining Web services process modelling concepts and reuse existing mechanisms as much as possible. When no proper standard is available the extension mechanisms should be specified separately from the core BPEL4WS specification. Examples of using these principles is the definition of the BusinessActivity protocol of WS-Transactions specification [36] (which is becoming a separate specification of its own) and the reliance on parts of WS-Addressing [29] standards proposal.

Keeping these ten design goals in mind we will next go through and evaluate the features of BPEL4WS.

3.2 Features of BPEL4WS

BPEL4WS defines a rich set of features for modelling the abstract processes (or business protocols) and executable processes. The abstract and executable processes share the same core concepts and have only minor specialised extensions. All features of a BPEL4WS process are shown in Figure 3.1.

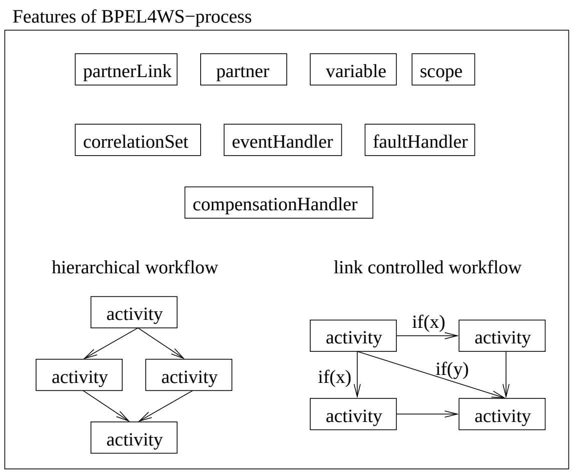

Figure 3.1: Features of BPEL process.

The basic features include defining multiple business partners and partner relationships. Variables can be defined and used for handling and storing business data. There are also constructs for event based modelling, as well as for fault handling. Successfully executed but later cancelled transactions can be modelled with compensation handlers. Finally, the instance correlation rules for process can be defined using the correlation constructs.

For workflow modelling BPEL4WS offers a rich set of primitive activities. These activities can be structured into separate namespaces (scopes). The control flow of the primitive activities can be defined using links and structured activities. The latter allow sequential, concurrent and repeated execution of activities, as well as selecting the path of execution depending on the outcome of a condition. Thus, BPEL4WS successfully allows controlling the behaviour of a business process based on interactions with its partners.

As far as security is concerned, BPEL4WS specification does not offer its own solutions. The specification only recommends using WS-Security [34] and other XML security specifications and encourages to address security considerations in process architectures. The particular execution platforms and programming environments will have a great effect on the security solutions that can be built.

Next we will take a closer look at each feature included in BPEL4WS specification.

partnerLink represents a conversational relationship between an external party and the BPEL4WS process. A partner link is created by defining myRole (the role the process plays), partnerRole (the role the partner plays), and partnerLinkType. Partner link type is created by defining the WSDL portTypes provided by each role. As WSDL separates the abstract interface (portType) from the actual port this distinction exists in BPEL4WS too. Port defines the details of the actual communication endpoints (e.g. URLs) and other deployment-specific information (e.g. public keys).

Process is statically dependent on portTypes but port related information can be configured at deployment time or at runtime. The EndpointReference construct taken from the WS-Addressing [29] can be used for runtime configuration. EndpointReference is thus useful for implementing a dynamic binding scheme (like a callback mechanism).

partner enables grouping of partner links based on expected business relationships. It is possible for example to define that partner links X and Y must be supported by one partner and not by two separate partners. This enables the reuse of partner links in defining complex business relationships.

variable is used for storing state information in the process. State information can be, for example, messages received and sent. Variables are used mainly for holding parts of received business messages. They enable stateful conversations and storing of process execution history.

correlationSet and correlation constructs are used for defining instance management rules for the process. Correlation approach to instance identification enables partner specific instance correlation ids which may change over time. Usage of correlation ids is an alternative approach to using object references. Correlation ids allow for a loosely coupled approach to instance identification as well as allow for instance management rules to be defined at the business data level (they become infrastructure independent). The price to pay compared to usage of object references is the increased complexity and the extra programming effort required.

eventHandler is used for modelling event-driven business processes or adding event-driven functionality to conventional business processes. Events can be messages from an external partner or they can be internal "alarms", for example an expired deadline. One possible use of event handlers is to define activities that do not occur in a basic business scenario but occur occasionally and may affect the outcome of the business process. One example of this kind of activity is an occasional status query or an "out of stock" notification from a warehouse system. It should be noted that events are not faults but instead they are considered to be part of the normal processing in a scope. Events may thus occur concurrently with other execution in the scope.

faultHandler is used for defining exceptional workflow paths. Fault handlers define what actions to perform in order to recover from a fault. Faults are specified in WSDL. Fault handlers are equivalent to exception handlers in many programming languages (like Java). This means that they are fundamentally designed for communication based on (remote) procedure calls. In a tightly coupled RPC based communication the remote faults happen during the processing of the procedure call. In a loosely coupled situation where communication is asynchronous in nature it is also possible that faults happen between sending the asynchronous request and receiving the asynchronous response. How the asynchronous situation should be modelled with BPEL4WS fault handling mechanism is not trivial as BPEL4WS does not offer a modelling pattern to support this directly.

compensationHandler is used for "reversing" or "undoing" successfully executed scopes in an application defined way. An example is cancelling a previously made flight reservation because of a fault later in the process. At the moment compensation handler is only a "wrapper" for compensating actions. The BPEL4WS specification states that "it is recognised that in many scenarios the compensation handler needs to receive data about the current state of the world and return data regarding the results of the compensation".

scope is a construct which allows the process to be structured into separate namespaces (sort of subprocesses) that tie together related pieces of the process. These scopes may again be structured into subscopes. A scope can define its own variables, fault handlers, event handlers, correlation sets and a compensation handler.

activity is a basic building block of business workflow. Basic activities which can be used in workflows are invoke, receive, reply, assign, empty, wait, throw, and compensate. In executable processes also the terminate activity is available. The activities are typically used for sending and receiving messages (invoke, receive, reply) or handling variables and related data flow (assign).

structured activity is a control structure which can contain basic activities or other structured activities. Structured activities are used for hierarchical structuring of the workflow and are inherited from XLANG. Structured activities include flow, sequence, switch, while, and pick. In a typical usage scenario sequences are used for specifying sequential execution and flows are used for specifying concurrent execution of activities.

links are inherited from WSFL and can be used to implement link-based control of workflow. Links can be mixed with structured activities and they resemble the "if (condition) then goto (place in the workflow)" style of control. Links are useful in cases where workflow's execution path is conditionally dependent on the contents of the business data.

In the next section we will further evaluate the usability and future development directions of BPEL4WS.

3.3 Usability of BPEL4WS

The main usage patterns of BPEL4WS are development of executable business processes and publishing business processes to external business partners as abstract processes (i.e. business protocols). Next we will evaluate the usability of BPEL4WS from both viewpoints.

3.3.1 Modelling Abstract Business Processes (Business Protocols)

The BPEL4WS approach to publishing abstract processes is quite straightforward. Let's assume that there exists a business process that is modelled to communicate with several interorganisational partners. When a business protocol related to a partner is wanted to be published, its relevant parts are extracted from the executable process in to an abstract BPEL4WS process. The new process contains only those parts of the executable process which define the choreography with the selected partner. This new BPEL4WS process is called abstract process (or business protocol) and can be published in some repository or given directly to an already known business partner. This business partner can then take the BPEL4WS based business protocol, produce a "mirror" image of it (by hand or by tool) and integrate it to one of its existing or new BPEL4WS processes.

Depending on the complexity of the situation the integration effort might be very smooth or it might require some hands-on adjustments at the integrator site. After the integration effort is complete, the business processes in both sides can automatically cooperate. This scenario is illustrated in Figure 3.2.

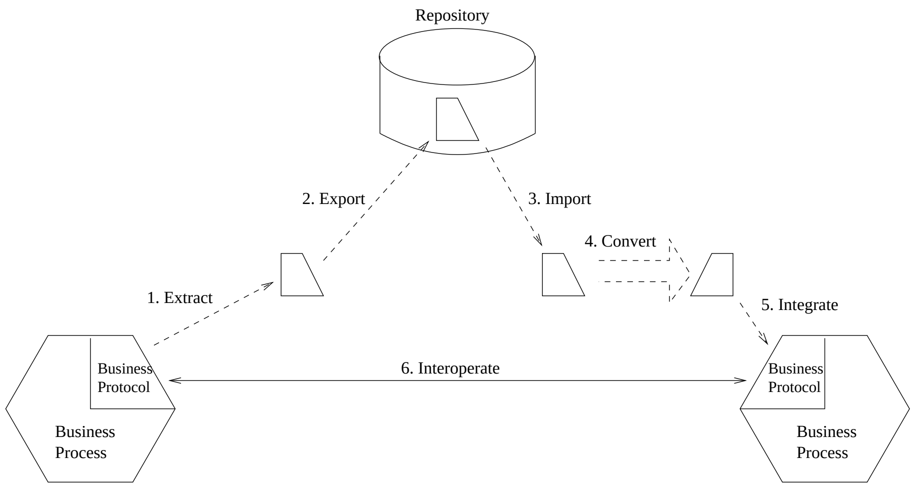

Figure 3.2: Business process integration using BPEL4WS abstract processes.

Summarising all this we arrive to several conclusions about BPEL4WS usability in publishing business processes. As a procedure it is easy to publish the internal business process as a group of abstract subsets of it. However at runtime level the original process stays monolithic (the published part is just a "view" of a subset of the monolithic process). This means that it is difficult to change either the public or the internal parts of the process independently of each other. They are tightly coupled. It is also difficult to support more than one version of the published business protocol as this would require supporting more than one version of the executable business process.

This kind of approach works well if the relationship between business partners is quite stable and cooperation is close. If, however the relationship is more dynamic and completely autonomic or if there are relationships to a large group of separate business domains at the same time then management problems easily arise.

In the series of articles at IBM developerWorks [14, 15, 16, 17] the BPEL4WS usability was evaluated by converting the RosettaNet [23] based real world business protocols to BPEL4WS. In this evaluation BPEL4WS was seen as a generally good tool. BPEL4WS advantages compared to conventional RosettaNet modelling were support for several partner business protocols, flexibility and powerful control over business process structure, and standard mechanisms for tying the public process to back-end systems (via WSDL portTypes). What was considered as a downside was BPEL4WS lack of QoS parameters (like number of retries, timeouts, ...). Moving from a third party view in RosettaNet protocols to separate views for each partner in BPEL4WS was not trivial in all occasions. The lack of standard security and reliability guarantees provided by the infrastructure made it difficult to replicate the exact behaviour of the RosettaNet protocols using BPEL4WS.

In addition to the shortcomings in the actual modelling capabilities, one problem is the programming language kind of nature of BPEL4WS. Although it is expressive it does mandate modelling at quite a low abstraction level. This means that the business protocol models have to be very detailed. The high detail level combined with the lack of formal semantics makes automatic verification of BPEL4WS based business protocols difficult.

As a summary it can be said that BPEL4WS offers a straightforward and simplistic approach to integration of autonomic business processes but does not cope well with the development of agile and dynamic business networks. In order to support more adaptable integration model the internal and external views of business processes should be better separated and the conversations and process execution more dynamically managed. For example the CS-WS initiative [10, 12] is aiming at this direction.

3.3.2 Modelling Executable Business Processes

The other contribution of BPEL4WS is the capability to model executable business processes. BPEL4WS offers a great deal of flexibility and expression power for this purpose. As a diffusion of two existing description languages BPEL4WS supports several modelling approaches and thus might attract a large base of existing developers.

Even though the relatively low level of abstraction in BPEL4WS does not fit perfectly to modelling of business protocols between autonomous parties, it fits very well to modelling of the executable internal business processes. The tight coupling with WSDL and reliance on WSDL portTypes as the interface between the process and the rest of the world makes integrating the business process with organisations' IT systems easy and straightforward. Of course mandating one interface description language (WSDL) has its drawbacks and systems which do not naturally support a WSDL interface must be wrapped with one (or adaptors must be used between the process and the back-end).

The lack of standards support for business transactions management (BTM) has raised some questions in the OASIS WS-BPEL TC and elsewhere [9]. For example Choreology Ltd. has produced a submission [8] which suggests that BPEL4WS should include programming constructs for creating, terminating, joining, and propagating as well as manipulating business transaction contexts. This kind of BTM-enabled BPEL4WS process could run on top of several different business transaction protocols. In addition, the submission suggests further developing the compensation handling mechanisms and adding support for cancel handlers and confirm handlers.

Finally, there exists no standard definition of a stateful Web Service and since business processes are stateful by their very nature, the designers of BPEL4WS have had to come up with their own definitions and interpretations. In the future this situation may change and for example the OASIS ASAP TC [18] is developing extensions to SOAP in order to "provide a generic means for asynchronous (and long running) services that can be easily incorporated in BPEL4WS as well as other protocols". How the developments in this effort as well as in other efforts where stateful Web services are being considered affect the development of BPEL4WS remains to be seen.

As a summary, the major shortcomings of BPEL4WS in the area of process modelling are the lack of fault modelling support for loosely coupled and asynchronous processes and the lack of direct support for programming transactional processes. The fundamentally monolithic nature of BPEL4WS process also makes it hard to build an adaptive execution environment.

A more generalised and thorough evaluation of BPEL4WS features can be found in [37] where BPEL4WS features are evaluated in reference to the well known van der Aalst workflow pattern framework [27].

Chapter 4

Evaluation of Platforms

In this chapter we will go through the platform experiences gained during the implementation of the case study. As already explained in chapter 2 the implemented platform architecture utilises three Java based platforms running on Red Hat Linux operating systems. Apache Tomcat [2] was used as the servlet engine. Apache Axis [1] (Apache eXtensible Interaction System) was used as the SOAP [24] and WSDL [31] platform and IBM alphaWorks BPWS4J [11] (IBM Business Process Execution Language for Web Services Java) was used as the runtime engine for BPEL4WS [3] processes.

In the next sections we will first summarise the experiences from the Apache Tomcat and Apache Axis platforms, then look deeper into the problems and solutions of BPWS4J platform and finally discuss the future of the BPEL4WS platforms in general.

4.1 Apache Tomcat and Apache Axis

The community developed Apache platforms are usually good in quality and Tomcat and Axis make no exception to the rule. Apache Tomcat has earned its reputation as an industrial strength servlet engine and Apache Axis, which is a replacement/follower for Apache SOAP, is the de facto Java based Web services platform.

Since our main interest lies in the area of Web services the more interesting of these two platforms is Axis. The Axis server can be plugged into servlet engines like Tomcat. Axis provides a full blown SOAP implementation as well as support for generation of Java stubs from WSDL descriptions and vice versa (WSDL2Java and Java2WSDL). In addition the Axis platform implements a TCP/IP monitoring tool. The monitoring tool is flexible and easily configurable and it was found to be of great help in learning about and debugging SOAP based applications. The Axis platform can be used as such or it can be embedded into larger platforms (like BPEL4WS). Both Tomcat and Axis are Open Source platforms.

Tomcat and Axis were found to be reliable and suitable to many kinds of software development scenarios. The performance and scalability of the platforms were not evaluated but at least with the QoS requirements of our scenario they performed well.

Properly securing Web services running on Tomcat and Axis is not an easy challenge. Tomcat does offer a range of security support, mainly concerned with authentication, access control, integrity and confidentiality (for an overview see e.g. [25]). Axis doesn't currently implement any security protocols directly. It does support XML Signatures via a sister project, and transport level SSL encryption can be set up independently of Axis. Client authentication can be implemented using either HTTP basic authentication or certificates. A new project called WSS4J [35] aims to add WS-Security support for Axis.

4.2 IBM alphaWorks BPWS4J

The main interest in the platform evaluation work was focused on BPWS4J from IBM alphaWorks. AlphaWorks produces prototype implementations of emerging technologies which may, or may not, end up integrated into the IBM's production platforms.

BPWS4J is a BPEL4WS process engine which contains an embedded Axis environment. Thus it is capable of sending and receiving SOAP messages as well as generating and parsing WSDL. In addition to the execution environment BPWS4J includes a primitive validation utility for BPEL4WS processes and a plug-in for the Eclipse development tool, which provides a graphical user interface (GUI) for creating BPEL4WS processes.

It was known that BPWS4J is an experimental platform (a "research platform"). However, the used BPWS4J version was v2.0 so one might expect it to have some maturity. It was quickly found out that BPWS4J is not mature. First of all it does not fully implement the BPEL4WS specification. There is no support for true asynchronous messaging (only for RPC communication), the dynamic partner binding is not fully implemented, all data types are not supported, and the list goes on. In addition to the features completely left out of the implementation the implemented features are sometimes only partially implemented and the platform contains bugs.

One of the bugs discovered was found to be especially annoying. BPWS4J engine seems to be unable to decode SOAP messages containing multi-reference encodings. This bug is annoying because multi-reference encoding is widely used and for example BPWS4J engine itself uses it by default (implicating that for example two BPWS4J engines cannot properly communicate).

The reason for this incapability in the BPWS4J implementation could not be verified. It is, however, a known problem in the BPWS4J developer community. There exist at least two (partial) workarounds for this problem. The used workaround was to implement a simple SOAP gateway between communicating BPWS4J engines. The gateway is responsible for removing multi-reference encoding from the SOAP messages. It was later discovered that the bug could also be circumvented by reinstalling the embedded Axis engine in BPWS4J with a version which is reconfigured (and recompiled) not to ever use multi-reference encoding. This second workaround was never implemented since the already existing solution worked and was also useful in debugging efforts. No matter which workaround is selected its usage will only be partially successful, because without multi-reference support some more complex XML documents cannot be properly encoded.

In addition to missing functionality and bugs some performance and concurrency problems were encountered. For some reason the BPWS4J performance seems to be very indeterministic and sometimes even executing a simple workflow could take lots of time. Whether this is due to problems with the BPWS4J engine itself or with the Axis/Tomcat integration could not be verified. There were also problems with the concurrent execution of multiple process instances which sometimes resulted in problems in workflow execution.

BPWS4J does not offer any direct security support. The embedded Axis design also makes it difficult to use some of the security solutions otherwise supported by Axis. From this can easily be concluded that BPWS4J is not fit for applications with security considerations.

Of the missing functionality the lacking support for other than tightly coupled RPC communication is the biggest drawback. Many business to business scenarios are asynchronous in nature. The asynchrony problem cannot be completely solved but asynchronous communication can be emulated with RPC calls which have empty reply messages. This was the approach taken in the case implementation (where communication was asynchronous).

As a result of the experimentation with the BPWS4J engine it can be concluded that the engine seems like a non-finished research platform. With the workarounds in place it can be used to implement simple studies and scenarios in order to gain some initial experience but not much more.

An updated version of the BPWS4J platform is expected, and it is hoped to solve some of the biggest problems faced in the current release. The next logical step for IBM would however be to include BPEL4WS support in its production suite. In the next section we will further discuss the plans of several potential BPEL4WS vendors.

4.3 Future of BPEL4WS Platforms

In addition to BPWS4J [11] there exists a couple of other platforms with BPEL4WS runtime support. These platforms are BPEL Orchestration Server from Collaxa [5] and ChoreoServer from OpenStorm [20]. Both platforms claim to be fully BPEL4WS compatible and of industrial strength.

At the moment, however, there exists no real BPEL4WS support from the leading vendors in the application server market. IBM (WebSphere), Microsoft (BizTalk), and BEA (WebLogic) have all announced that BPEL4WS support will be included in "future" releases of the platforms. Some of them already have some editor support but not much more. Since no real support has surfaced from these vendors one must ask the question why? BPEL4WS still has momentum and is increasingly referenced in the Web and in research conference proceedings.

The most probable reason for the lack of BPEL4WS support is that the vendors with existing execution infrastructure already in place do not at the moment believe in BPEL4WS as a runtime concept. They view BPEL4WS as a standard way for exporting business processes, delivering them to trading partners and then importing them again at trading partners domains. Whether the runtime engine is a BPEL4WS engine is not considered important. For example Microsoft has announced that future BizTalk runtime capabilities are a superset of BPEL4WS runtime capabilities and thus BPEL4WS based processes can be imported into the BizTalk execution environment.

As a conclusion, BPEL4WS platform support exists and is still developing. However, the most successful BPEL4WS usage pattern seems to be exporting and importing public processes in a standard form. BPEL4WS has evolved to be more of an interoperability standard than a modelling or runtime standard. Whether the trend will change and whether all vendors will develop similar views of BPEL4WS's future remains to be seen. The possible arrival of Open Source BPEL4WS support would be good progress since it would change the current market dynamics and force the big vendors to make bolder decisions.

Chapter 5

Conclusions

During the summer and autumn of 2003 a case study was implemented in order to better understand process oriented B2B modelling and especially the usability of BPEL4WS language. The implemented architecture and the experiences gained are discussed throughout this report. In the end one must ask the question what can we conclude from all this?

First of all we can conclude that the main goals of the case study were achieved. Understanding and experience increased and technologies were successfully used. From those evaluations we can draw a few conclusions especially about BPEL4WS. As explained, BPEL4WS has two intended use cases (although other use cases are not forbidden). One use case is to use BPEL4WS as a way to compose Web services into executable business processes. For this purpose BPEL4WS suites quite well. Another use case is to use BPEL4WS as a way to publish executable processes as abstract processes ("business protocols"). For this purpose BPEL4WS can be used but it does not support dynamic evolution of business networks well.

On the other hand if we look at the market trends in the BPEL4WS platforms development we see much more confidence in BPEL4WS as a standard way of exporting and importing business protocols than as a way of modelling and executing business processes. There certainly exists some controversy and it makes one wonder what might be the reasons behind it.

The reason why BPEL4WS is gaining momentum as an interoperability standard seems obvious. This use case is valid and despite the lack of a sophisticated approach and some features, BPEL4WS still is the best-of-breed in this area. BPEL4WS can be used in static B2B integration scenarios quite successfully, it is compatible with other Web services technologies, and it is a standard which most vendors can agree on. Since most current B2B integration scenarios are quite static in nature BPEL4WS capabilities are more than enough. BPEL4WS is still developing and other supporting specifications may arrive so the current lack of features is not seen as a problem.

The reason why BPEL4WS is not gaining as much momentum as an execution standard seems to be the opposite. The use case is not as valid. All major platform vendors already offer high quality enterprise integration platforms as well as support for workflows. BPEL4WS offers a new approach to the problem but not necessarily a better one. Some startup companies believe in BPEL4WS in this area but they have started from "an empty table" and don't have the legacy of an existing family of platforms slowing down the efforts.

The advantages of the BPEL4WS approach as a runtime standard would be its Web services background and its status as a quite commonly agreed specification. This would make possible standard execution engine and tool support. The big vendors, however, do not necessarily see the standards status in this area such an important issue (at least at the moment). They seem to conclude that the integration of public BPEL4WS processes with the internal enterprise system can be done without directly using BPEL4WS at runtime. If the capabilities of the existing runtime platforms are greater than those required/offered by BPEL4WS, the public BPEL4WS processes can be converted to run on the vendor's environment without directly supporting BPEL4WS at runtime (this is claimed to be true at least in future platform releases).

The idea of a standard publishing and execution language is not new and for example Workflow Management Coalition (WfMC) [38] has been working on the issue since 1993. WfMC has for example developed a reference model [39] and later its own BPEL4WS "like" language called XPDL [40]. Other efforts like BPML [4] from Business Process Management Initiative (BPMI), ebXML Business Process Specification Schema (BPSS) [7] and W3C's WSCI [30] are aiming at same or similar goals. None of the standardisation efforts have been able to create a commonly adopted and truly interoperable workflow standard so far.

One reason behind the incoherence is the fact that the paths of the academia and industry have never really met. The result is that industry has developed a large group of non-interoperable and semantically lacking workflow languages and the research results of academia are not widely known or utilised. These issues are addressed for example by van Der Aalst [26].

How the market develops and whether new Open Source platforms (like the ones in [21]) will emerge to put pressure on the vendors remains an open question. The facts are that interest for BPEL4WS is still there but the consensus of its best uses has diverged some bit. Choosing to use BPEL4WS now is a bit more risky decision than it seemed to be six months ago. In any case, BPEL4WS still is an important frontier technology in Web services world and has not yet reached its full potential.

Bibliography

[1] Apache AXIS (v1.1). *Published on Web* (June 2003). `http://ws.apache.org/axis`.

[2] Apache Tomcat (v4.1). *Published on Web* (Jan. 2003). `http://jakarta.apache.org/tomcat`.

[3] Business Process Execution Language for Web Services (v.1.1). *Published on Web* (May 2003). `http://www-106.ibm.com/developerworks/library/ws-bpel`.

[4] Business Process Modeling Language. *Published on Web* (Mar. 2001). `http://www.bpmi.org/bpml-spec.esp`.

[5] Collaxa BPEL Server. *Published on Web* (Oct. 2003). `http://www.collaxa.com/`.

[6] Columns and Discussions of BPEL4WS. *Published on Web* (May 2003). `http://www-106.ibm.com/developerworks/library/ws-bpelcol.html`.

[7] ebXML. *Published on Web* (Oct. 2003). `http://www.ebxml.org`.

[8] FLETCHER, T., FURNISS, P., GREEN, A., AND HAUGEN, R. BPEL and Business Transaction Management: Choreology Submission to OASIS WS-BPEL Technical Committee. *Published on Web* (Sept. 2003). `http://www.oasis-open.org/committees/download.php/3263/BPEL.and.Busines%s.Transaction.Management.Choreology.Submission.html`.

[9] GREEN, A. Transacting Business with Web Services Part I. *Web Services Journal (Published on Web)* (Sept. 2003). `http://www.sys-con.com/webservices/article.cfm?id=647`.

[10] HANSON, E. J., NANDI, P., AND KUMARAN, S. Conversation Support for Business Process Integration. *6th International Enterprise Distributed Object Computing Conference (EDOC 2002)* (Sept. 2002). `http://www.research.ibm.com/convsupport/papers/edoc02.pdf`.

[11] IBM alphaWorks BPWS4J (v2.0). *Published on Web* (Apr. 2003). `http://www.alphaworks.ibm.com/tech/bpws4j`.

[12] KUMARAN, S., AND NANDI, P. Conversational Support for Web Services: The next stage of Web services abstraction. *Published on Web* (Sept. 2002). `http://www-106.ibm.com/developerworks/webservices/library/ws-conver`.

[13] LEYMANN, F., ROLLER, D., AND THATTE, S. Goals of the BPEL4WS Specification. *Published on Web* (Aug. 2003). `http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf`.

[14] MASUD, S. Use RosettaNet-based Web services, Part 1: How to instantly add years of e-business experience and expertise to your Web services. *Published on Web* (July 2003). `http://www-106.ibm.com/developerworks/webservices/library/ws-rose1`.

[15] MASUD, S. Use RosettaNet-based Web services, Part 2: Standing on the shoulders of a giant: Web services with RosettaNet. *Published on Web* (July 2003). `http://www-106.ibm.com/developerworks/webservices/library/ws-rose2`.

[16] MASUD, S. Use RosettaNet-based Web services, Part 3: Digital representations and e-business dialogues. *Published on Web* (Aug. 2003). `http://www-106.ibm.com/developerworks/webservices/library/ws-rose3`.

[17] MASUD, S. Use RosettaNet-based Web services, Part 4: Building reliable asynchronous processes with BPEL4WS. *Published on Web* (Sept. 2003). `http://www-106.ibm.com/developerworks/webservices/library/ws-rose4`.

[18] OASIS ASAP Technical Committee. *Published on Web* (Oct. 2003). `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=asap`.

[19] OASIS WS-BPEL Technical Committee. *Published on Web* (Oct. 2003). `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.

[20] OpenStorm ChoreoServer. *Published on Web* (Oct. 2003). `http://www.openstorm.com/`.

[21] Open source workflow and BPM-market. *Published on Web* (Oct. 2003). `http://jbpm.org/article.html`.

[22] POGLIANI, S. From content-oriented Web to transaction-oriented Web. *OMG Information Day on Integrating the Enterprise (Helsinki)* (Mar. 2003).

[23] RosettaNet Consortium. *Published on Web* (Oct. 2003). `http://www.rosettanet.org`.

[24] Simple Object Access Protocol (v.1.1). *Published on Web* (May 2000). `http://www.w3.org/TR/SOAP`.

[25] Tomcat Security Overview. *Published on Web* (Oct. 2003). `http://www.cafesoft.com/products/cams/tomcat-security.html`.

[26] VAN DER AALST, W. Don't go with the flow: Web services composition standards exposed. *Jan/Feb 2003 issue of IEEE Intelligent Systems* (Jan. 2003). `http://tmitwww.tm.tue.nl/research/patterns/download/ieeewebflow.pdf`.

[27] VAN DER AALST, W., TER HOFSTEDE, A., KIEPUSZEWSKI, B., AND BARROS, A. Workflow Patterns. *Distributed and Parallel Databases, 14(3), pages 5-51* (July 2003). `http://tmitwww.tm.tue.nl/research/patterns/download/wfs-pat-2002.pdf`.

[28] Web-Pilarcos Project. *Published on Web* (Oct. 2003). `http://www.cs.helsinki.fi/group/web-pil`.

[29] Web Services Addressing. *Published on Web* (Mar. 2003). http://www-106.ibm.com/developerworks/webservices/library/ws-add/.

[30] Web Services Choreography Interface (v.1.0). *Published on Web* (Aug. 2002). http://www.w3.org/TR/wsci/.

[31] Web Service Definition Language (v.1.1). *Published on Web* (Mar. 2001). http://www.w3.org/TR/wsdl.

[32] Web Services Flow Language (v.1.0). *Published on Web* (May 2001). http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[33] Web Services Policy Framework. *Published on Web* (May 2003). http://www-106.ibm.com/developerworks/library/ws-polfram/.

[34] Web Services Security. *Published on Web* (Apr. 2002). http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.

[35] Web Services Security for Java (WSS4J). *Published on Web* (May 2003). http://sourceforge.net/projects/wss4j.

[36] Web Services Transaction. *Published on Web* (May 2003). http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglob%spec/html/wstxspecindex.asp.

[37] WOHED, P., VAN DER AALST, W. M., DUMAS, M., AND TER HOFSTEDE, A. H. Pattern Based Analysis of BPEL4WS. *QUT Technical Report, FIT-TD-2002-06, Queensland University of Technology, Brisbane* (June 2002). http://tmitwww.tm.tue.nl/research/patterns/download/qut_bpel_rep.pdf.

[38] Workflow Management Coalition. *Published on Web* (2003). http://www.wfmc.org.

[39] Workflow Management Coalition - The Workflow Reference Model. *Published on Web* (Jan. 1995). http://www.wfmc.org/standards/docs/tc003v11.pdf.

[40] Workflow Process Definition Interface - XML Process Definition Language. *Published on Web* (Oct. 2002). http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf.

[41] XLANG: Web Services for Business Process Design. *Published on Web* (June 2001). http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

Helsinki 2003