

BPEL-Editor

Georg E. Paulusberger
July 2004

1. Project overview and targets

Based on WSDL 1.1 and BPEL 1.1, a graphical editor for modelling Web Services based business processes was developed. This first version of the BPEL-Editor supports the modelling of, for example, a Web Services based loan approval or ticket ordering service employing up to approximately 5 Web Service partners. The editor uses two types of WSDL files as input. The first one is the WSDL description of the Web Service partners, and the second one is the WSDL file of the BPEL process itself. They are both represented as annotated rectangles. The output is the XML-BPEL process file. Calls of or information passing between the BPEL process and its web service partners are represented by directed edges. On a formal level the implementation is based on the mathematical theory of directed graphs, implemented by the underlying JGraph-Framework (see chapter “IDE, Tools, APIs”).

From a modelling or representational point of view on the one hand and from the user’s standpoint on the other, the BPEL-Editor tries to accomplish the following tasks:

1. Present a high-level graphical model of the underlying business process
2. Free the modeller of the mundane task of writing BPEL-XML elements and attributes to define a business process with all its variables, partners, partner links, roles a.s.f.

2. WSDL und BPEL in a nutshell

We will now give a concise description of the Web Services Concept and the two Web Services core specifications WSDL 1.1 and BPEL4WS 1.1 including their interrelationship.

2.1. The Web Services Concept

2.1.1.XML (eXtensible Markup Language)

XML forms the very basis of Web Services. It is the common language for defining remote procedure calls (based on SOAP = Simple Object Access Protocol), defining the interface of a Web Service and Web Service directories. Finally it is used to describe a BPEL-modelled business process.

Any communication between Web Services is XML-coded. Thus XML enables and supports a language- and platform-independent programming model facilitating the integration of applications both within an enterprise and between enterprises.

2.1.2.SOAP (Simple Object Access Protocol)

SOAP is an XML-based messaging system. According to [GOTTSC02] any message transported over a network usually falls into one of the following two categories:

1. “Messages that are composed primarily of a document that is to be processed remotely.
2. Messages that contain commands and parameters that are used to directly invoke a remote procedure (i.e., remote procedure calls)”

SOAP supports both of them. Essentially SOAP provides XML-based structures (elements and attributes) for the following Web Service operations:

1. **Publish**, i.e. either a Web Service provider enters its Service Description into a Web Service directory or it makes it directly known to the user.
2. **Find**, i.e. either a Service Requestor queries a Web Service directory for a particular Web Service description or retrieves it directly from the Web Service provider.
3. **Bind**, i.e. the abstract definitions of Web Service operations and messages are connected with (bound to) a concrete message format and the protocol details. This happens at runtime when the Service Requestor invokes or initiates an interaction with the Service Provider. It then uses the binding details in the Service Description to locate, contact and invoke the Service.
4. **Invoke**, i.e. a client (Service Requestor) calls a Web Service to use its functionality.

2.1.3.Definition

[KREG01] gives the following definition of a Web Service:

„A *Web service* is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its *service description*. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and locations. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. ... Web Services fulfil a specific task or a set of tasks. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction.”

Central to this IBM-understanding of Web Services is the interface nature of a Web Service. In concrete terms this is expressed by the WSDL-based definition of a Web Service. Any WSDL Web

Service definition clearly separates between the (abstract) definition and the implementation of its operations offered to a Service Requestor.

2.1.4. Model and Structure of Web Services

[KREG01] gives the following overview of the Web Service architecture and the flow of interactions:

„The Web Services architecture is based upon the interactions between three roles: service provider, service registry and service requestor. The interactions involve the publish, find and bind operations. Together, these roles and operations act upon the Web Services artefacts: the Web service software module and its description.”

2.1.5. Artefacts (Service and Service Description)

According to [KREG01] the two central Web Services artefacts are Service and Service Description. These are defined as follows:

- **„Service.** Where a Web service is an interface described by a service description, its implementation is the service. A service is a software module deployed on network-accessible platforms provided by the service provider.
- **Service Description.** The service description contains the details of the interface and implementation of the service. This includes its data types, operations, binding information and network location. ... The service description might be published to a service requestor or to a service registry.”

2.1.6. Operations

Web Service operations are XML-/SOAP-based and are described [here](#).

2.1.7.Roles

Again according to [KREG01] the [above](#) already mentioned roles and functions are defined as follows:

- „**Service provider.** From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that hosts access to the service.”
- “**Service requestor.** From a business perspective, this is the business that requires certain functions to be satisfied. From an architectural perspective, this is the application that is looking for and invoking or initiating an interaction with a service. The service requestor role can be played by a browser driven by a person or a program without a user interface, for example another Web service.”
- **Service registry.** This is a searchable registry of service descriptions where service providers publish their service descriptions. Service requestors find services and obtain binding information (in the service descriptions) for services during development for static binding or during execution for dynamic binding.”

Note: Both the Service Provider and the Service Requestor are logical roles. A Service can either switch roles or show characteristics of both of them during an interaction.

2.1.8. Typical flow of Web Service interaction

In [KREG01] the following typical scenario is given:

„... a service provider hosts a network-accessible software module (an implementation of a Web service). The service provider defines a service description for the Web service and publishes it to a service requestor or service registry. The service requestor uses a find operation to retrieve the service description locally or from the service registry and uses the service description to bind with the service provider and invoke or interact with the Web service implementation.”

The graphic below shows these operations, the components which provide them and their interaction:

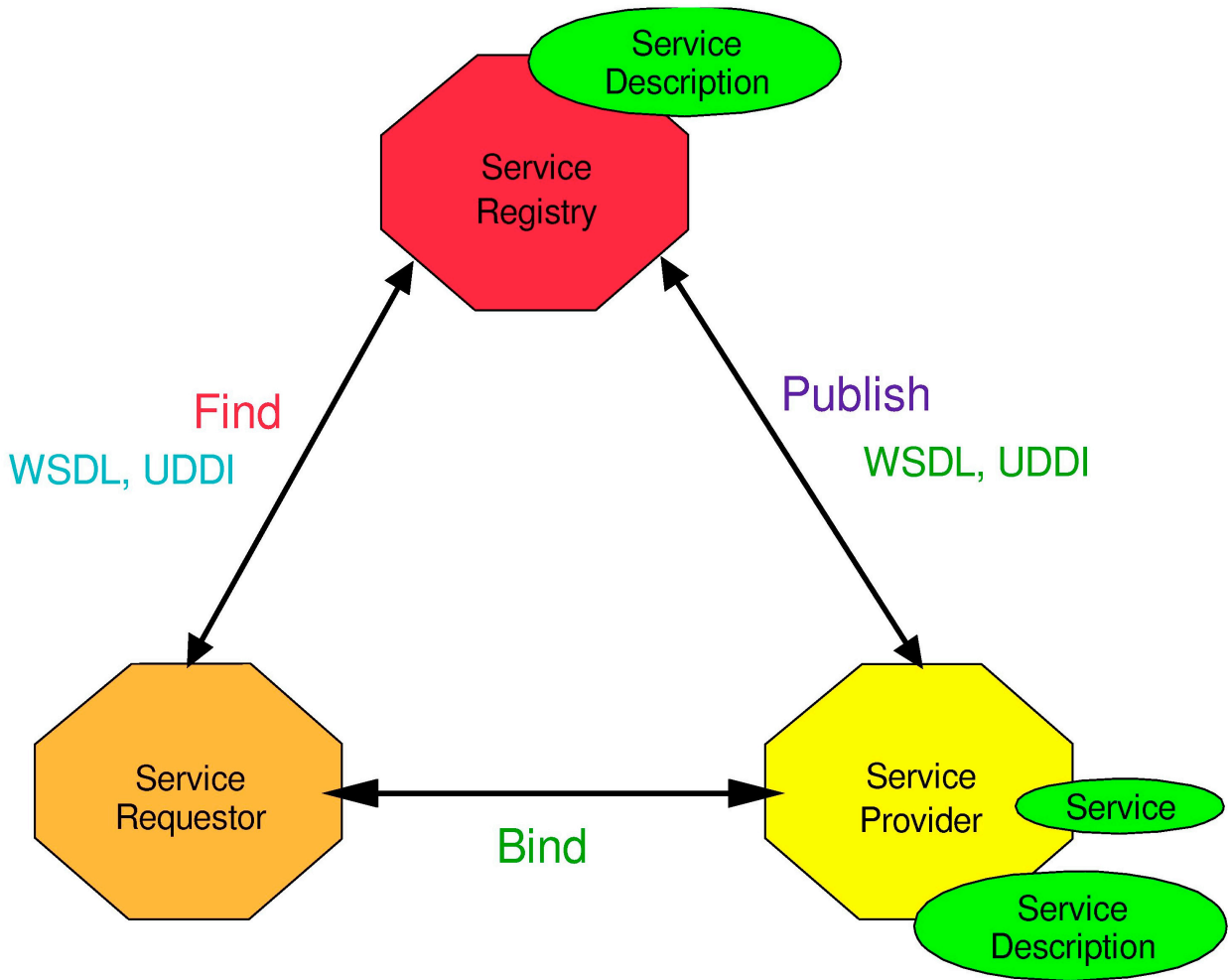
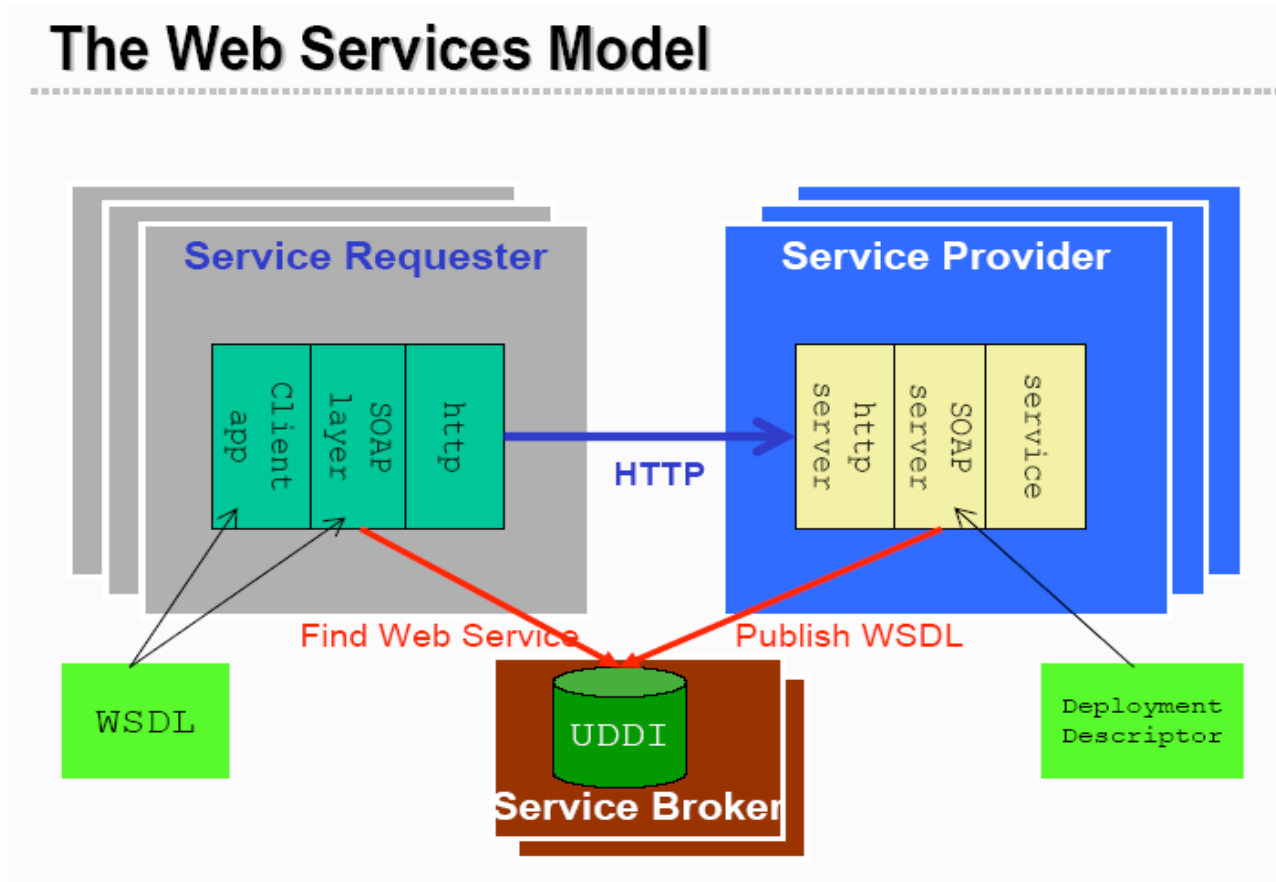


Figure 1. Web Services roles, operations and artifacts

A more detailed representation of the Web Service model is given in the following graphic. It also shows part of the protocol stack involved including HTTP being used as a transport protocol for SOAP.



2.1.9. The WSDL protocol stack

The Conceptual Web Services Stack

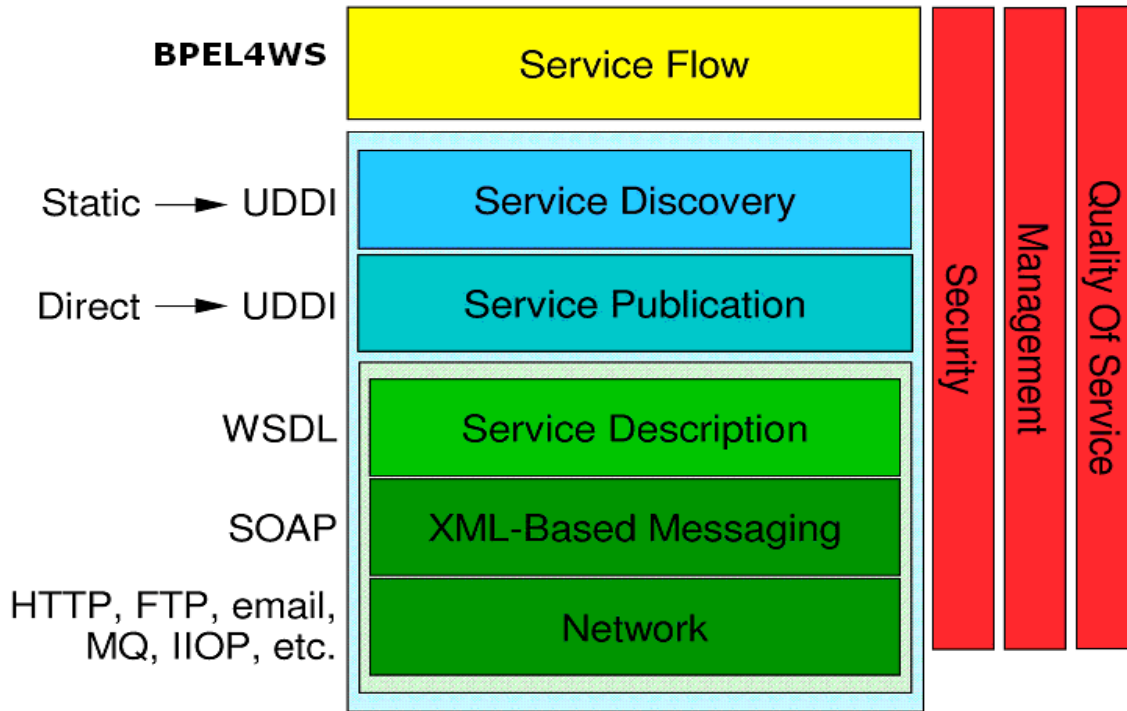


Figure 2. Web Services conceptual stack

2.2. Web Services Description Language (WSDL 1.1)

2.2.1. Some essential properties

WSDL is an XML application for describing network services as a set of communication endpoints capable of exchanging messages. The abstract definition of endpoints and messages (exchanged via these endpoints) is separated from their concrete network deployment and data format bindings.

Messages in a WSDL document are abstract descriptions of the data being exchanged. Port types are abstract collections of operations. Concrete protocol and data format specifications for a particular port type are used and constitute a reusable binding between abstract port types and abstract messages on the one hand and concrete network protocols and data formats on the other.

2.2.2. Elements of a WSDL document

2.2.2.1. Service interface elements

- Types:= Container for data type definitions using some type system (e.g. XSD)
- Message:= Abstract typed definition of the data being communicated
- Operation:= Abstract description of an action supported by the service
(Child element of Port Type)
- Port Type:= Abstract set of operations supported by one or more endpoints (= ports)
- Binding:= Concrete protocol and data format specification

2.2.2.2. Service implementation elements

- Port:= Association of a network address with a reusable binding
- Service:= Collection of related endpoints (= ports)

2.3. Business Process Execution Language 4 Web Services (BPEL4WS 1.1)

2.3.1. Introduction

During the last couple of years Web Services have become the state-of-the-art-tool for EAI (= Enterprise Application Integration) on different platforms. If Web Services are looked at as individual or isolated Internet-based services they are “easy” to use by programs. However, if two or more Web Services are to interact to create a “higher level” business process they must be coordinated. For that purpose their interfaces must complement each other and set into relationship to each other so they can process a business transaction. Therefore to fully use the potential of Web Services a language for the formal description of the higher level process is needed. By now BPEL4WS has been accepted as an industry-wide standard for that purpose.

2.3.2. Relationship between BPEL4WS and WSDL

In addition to XML-based specifications like XML schema 1.0 (= XSD) and XPATH 1.0 WSDL had the strongest influence on BPEL4WS. The BPEL4WS process model is situated on top of the [service model defined by WSDL 1.1](#). The most essential properties of the relationship between BPEL4WS and WSDL are described in [BPEL], chapter “Relationship with WSDL”, as follows:

„At the core of the BPEL4WS process model is the notion of peer-to-peer interaction between services described in WSDL; both the process and its partners are modelled as WSDL services. A business process defines how to coordinate the interactions between a process instance and its partners. In this sense, a BPEL4WS process definition provides and/or uses one or more WSDL services, and provides the description of the behaviour and interactions of a process instance relative to its partners and resources through Web Service interfaces. That is, BPEL4WS defines the message exchange protocols followed by the business process of a specific role in the interaction.

The definition of a BPEL4WS business process also follows the WSDL model of separation between the abstract message contents used by the business process and deployment information (messages and portType versus binding and address information). In particular, a BPEL4WS process represents all partners and interactions with these partners in terms of abstract WSDL interfaces (portTypes and operations); no references are made to the actual services used by a process instance.”

2.3.3. BPEL4WS development (historical origins)

Several companies started to define such a formal language mentioned in the introduction. In the end it was IBM's WSFL (= Web Services Flow Language) and Microsoft's XLANG (= eXtensible LANGuage) that were relevant for the development and definition of BPEL4WS. BPEL4WS is a synthesis of both workflow languages. On the one hand it was WSFL's support for graph-oriented processes and on the other it was XLANG's support for programming constructs that entered the synthesis. This means that the WSFL workflow modelling properties were united with XLANG's selection and control constructs like 'switch/case' and a while-loop known from higher level programming languages, thus forming BPEL4WS.

2.3.4. BPEL4WS process modelling paradigm

The central and fundamental modelling paradigm of BPEL4WS is the Orchestration paradigm. It determines all of BPEL4WS properties. In [[PELTZ](#)] this paradigm is characterized as follows:

“Orchestration describes how web services can interact with each other at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or organizations, and result in a longlived, transactional, multi-step process model. ... Orchestration refers to an executable business process that may interact with both internal and external web services. For orchestration, the process is always controlled from the perspective of one of the business parties.”



2.3.5. BPEL4WS functionality

BPEL4WS can be used to compose a new service from a given set of services. The BPEL4WS 1.1 specification provides for the following two usage patterns of BPEL4WS:

Firstly there are BPEL4WS-defined business processes that can be executed on a BPEL host, so-called “executable processes”. Secondly there are non-executable processes, so-called “abstract processes”. They are only meant to describe the flow of a business process between partners. Based on such an abstract process the participating Web Service partners can define (program) their own implementation in BPEL4WS and get it executed on their respective BPEL-host.

BPEL4WS-defined business protocols formalize the complex process interactions and thus make them verifiable. Using the BPEL4WS elements in the following table such a workflow can be defined.

Overview of BPEL4WS – Workflow Constructs	
BPEL – Construct	Semantics
<receive>	A business process provides services to its partners through receive activities and corresponding reply activities. Until an appropriate message is received the process is blocked.
<reply>	A reply activity is used to send a response to a request previously accepted through a receive activity. By combining <receive>/<reply> synchronous request-/response operations can be modelled.
<invoke>	Used by a client calling the Web Service interface of a partner; both request-/response calls and one-way-calls are supported.
<assign>	The assign activity can be used to copy data from one variable to another, as well as to construct and insert new data using expressions.
<throw>	Starts an exception handling procedure within a process.
<terminate>	Unconditionally terminates a workflow.
<wait>	Allows a business process to specify a delay for a certain period of time or until a certain deadline is reached.
<empty>	Void operation
<sequence>	A sequence activity contains one or more activities that are performed sequentially, in the order in which they are listed within the <sequence> element, this, in lexical order. The <sequence> element is a so-called „structured activity“.
<switch>	Based on a condition exactly one operation gets selected and executed.
<while>	Loop-construct. Semantics comparable to while-loop in higher level programming languages like Java.
<pick>	The pick activity awaits the occurrence of one of a set of events and then performs the activity associated with the event that occurred.
<flow>	Defines statements which are to be processed in parallel. Dependencies between these statements can be modelled with so-called „Links“.
<scope>	Defines an area within a workflow with its own local variables, its own exception handling and transaction management.
<compensate>	Defines a compensation handler for an already completely processed inner scope.

2.3.6. Exception handling and transaction management

In particular we would like to mention the <scope> element. It allows the definition of sub-scopes within a workflow that have their own exception handling. Moreover it is the transaction management that is situated on this level. BPEL4WS provides for the definition of business transactions with compensation code being attached to them, for example the cancellation of a booking.

2.3.7. Data model

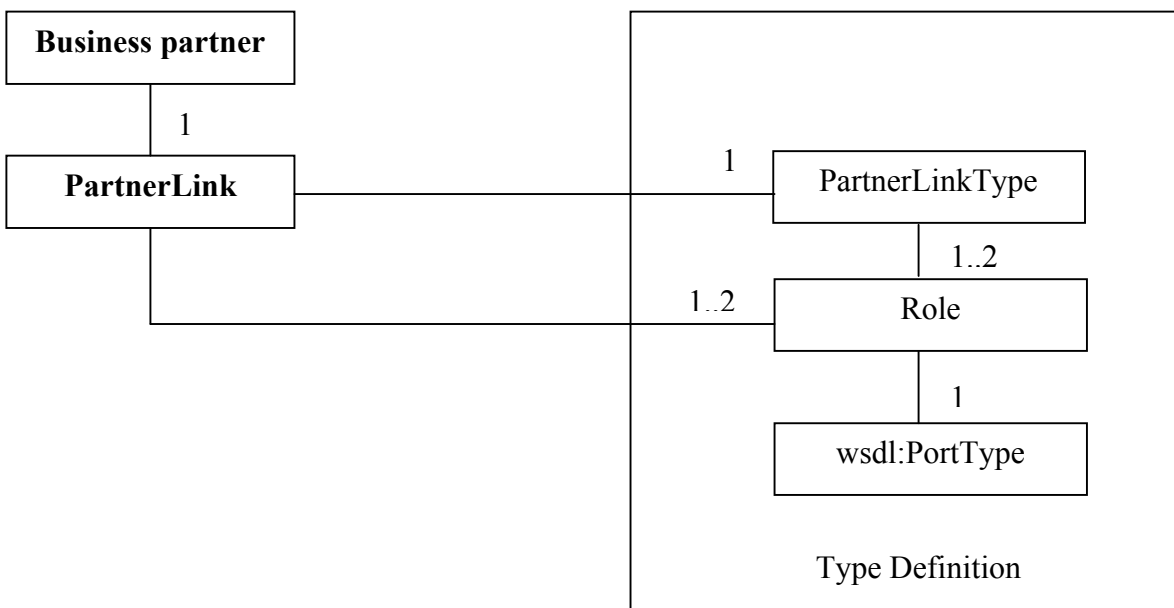
BPEL4WS uses XML schema, or XSD (= XML Schema Definition) for short, as its data model. XSD provides for both the definition of variables and the description of messages which are to be exchanged between Web Services. Moreover BPEL4WS relies on XPATH 1.0. This allows the selection of individual elements and/or attributes from XML messages. For example this is needed for the correlation of messages belonging to the same BPEL-process. In addition to that XPATH supports the definition of conditions to be used in constructs like <while> and <switch>.

2.3.8. Service interface description, roles and partners

The service interfaces of a BPEL-process are described using WSDL. When defining these service interfaces BPEL4WS follows the WSDL architecture by making a strict distinction between the definition of an abstract service and the binding of this service to a concrete URL. Within the BPEL-definition the URL is part of the deployment of the service. Thus processes are defined in a sufficiently abstract way that they can be reused in different deployments.

Within the BPEL-specification the concept of roles and partners is central. It helps to satisfy the requirements of inter-enterprise applications as follows:

Based on the WSDL `<portType>` element BPEL4WS defines a `<partnerLinkType>` element. As stated above WSDL `<portType>` is a set of (abstract) operations. `<partnerLinkType>` elements define classes or types of relationships between business partners. These types are then used in BPEL-processes. For example this could be a customer-supplier relationship within an ordering transaction. A `<partnerLinkType>` element allows the definition of up to two roles within a business relation. These roles get directly mapped to WSDL-`portTypes`. Thus interfaces get defined which the partners must provide in order to fulfil their roles. This is shown in the following graphic:



In order to use `PartnerLinkTypes` within a BPEL-process the BPEL-modeller defines one or more `PartnerLinks`. These `PartnerLinks` define the type of business relation within the given BPEL-process. `PartnerLinkTypes` can be reused in different processes. Further it is possible to define several `PartnerLinks` based on the same `PartnerLinkTypes`. For example there can be several suppliers within a common ordering transaction.

To summarize:

BPEL4WS is a very mature specification for the formal modelling of business processes and protocols based on Web Services. BPEL gets broad industry-wide support from renowned IT companies like IBM, BEA and Microsoft. It can be expected that Web Services will prevail over alternative concepts for enterprise application integration. There are already BPEL execution engines available (e.g. BPWS4J (= BPEL Web Services for Java) from IBM Alphaworks) which are capable of processing executable BPEL- models.

3. Implementation features and the GUI interaction model

3.1. Supported BPEL usages pattern

Of the two BPEL usage patterns defined in the BPEL specification version 1.1 (which are the abstract business protocol description and the executable business process description) the abstract pattern is supported by the BPEL-Editor. However, during the development process the following became evident: in order to model some meaningful and fairly realistic business processes a “critical mass” of BPEL language elements had to be implemented. Therefore, several BPEL elements (extensions) only defined for executable processes were also implemented (please refer to the complete list of implemented BPEL elements in chapter “Implemented BPEL elements”).

3.2. Modelling prerequisites and output produced

The only output the BPEL-Editor produces is the BPEL-process (XML-) file itself. All necessary WSDL-files of the involved Web Services partners and the BPEL process itself must be produced using some other WSDL authoring tool (e.g. the CapeClear WSDL-Editor, IBM’s WSTK kit, etc.).

3.3. GUI interaction model

In order to integrate two or more Web Service partners (including the BPEL process itself) into a BPEL business process basically the following steps must be taken:

- (1) Read in the WSDL description of the BPEL central coordinator through the standard file open dialogue.
- (2) Do the same as in step (1) for a Web Service partner
Note: For each output message in a <portType>-element¹ an outgoing edge will automatically be connected to the Web Service and/or central BPEL coordinator rectangle.
- (3) After dragging the outgoing edge of the BPEL central coordinator onto a Web Service partner or vice versa an <invoke>-dialogue or a <receive>-dialogue will popup, respectively.
- (4) Now values can be selected from lists, drop down boxes or entered in text fields and then the dialogue can be left by either clicking OK or Cancel.
- (5) If OK is clicked new BPEL process elements (like <partnerLink>, <invoke>, <assign>, etc.) will be created and added to the DOM-object representing the XML-BPEL process document.
- (6) In a final step, the BPEL process model, which is only memory resident up to now, can now be saved to a file (serialized).

Although the above description does not explicitly account for a selection-GUI element or while-GUI element being added to the graphical model, the GUI interaction model remains the same.

¹ To be precise: This will be done for each output message in an <operation>-element, where the <operation>-element is a child element of the <portType>-element.

3.4. Implemented BPEL elements

3.4.1. Relational elements

<partnerLinkType> and its child elements

<partnerLink> and its child elements

3.4.2. Basic activities

<invoke>, <receive> and <assign> with <copy> as child element

3.4.3. Structured activities

<sequence>, <switch> and <while>

4. Two Examples of BPEL-modelled Business processes

The following two business cases will be modelled using our BPEL-editor:

1. Case: Credit Selection Service: Both the BPEL <sequence>-element and the <switch>-element will be used. The BPEL-process will receive two answers (offers) from the participating Web Services partners and select the one with the lower interest rate.
2. Case: Separate and more abstract example applying the <while>-element.

4.1. Asynchronous Credit Selection Service

4.1.1.Participating Webservice Partner

1. Loan Client
2. Loan Broker
3. Credit-offering bank-1 (Star Loan)
4. Credit-offering bank-2 (United Loan)

In this scenario the Loan Broker takes the role of the central BPEL coordinator interacting with the Loan Client and both credit-offering banks.

4.1.2.Process flow

In the example, the assumption is made that the client is creditworthy. Therefore, the Loan Broker just transmits the loan application of the client to both banks without checking the client's creditworthiness.

1. The Loan Broker receives the loan application from the client
2. The Loan Broker contacts two credit-offering banks and chooses the one offering the lower interest rate.
3. The Loan Broker sends the offer with the lower interest rate to his client.

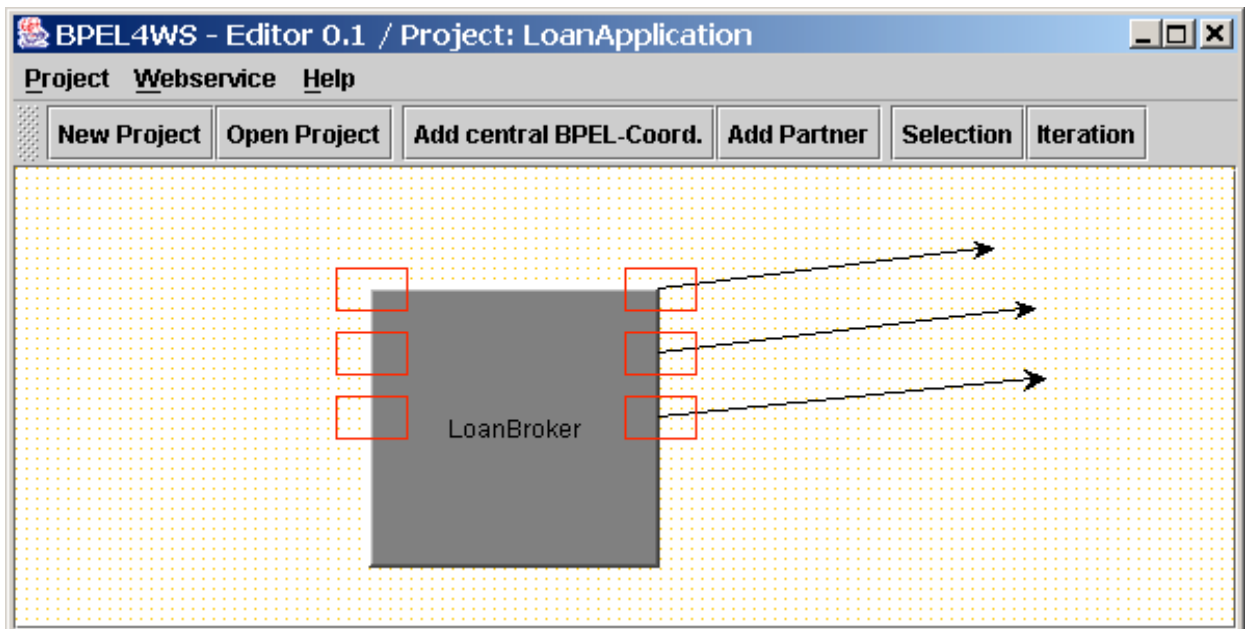
4.1.3.Preconditions for modelling with the BPEL-Editor

For each participating Web Service partner there has to be a WSDL file. The same holds for the Loan Client.

The following presentation will only roughly sketch the modelling steps using the BPEL-editor. For a more detailed discussion of the example please refer to the (German) [paper](#). **← Hier sollte ein Link auf die eigentliche Doku. erfolgen!**

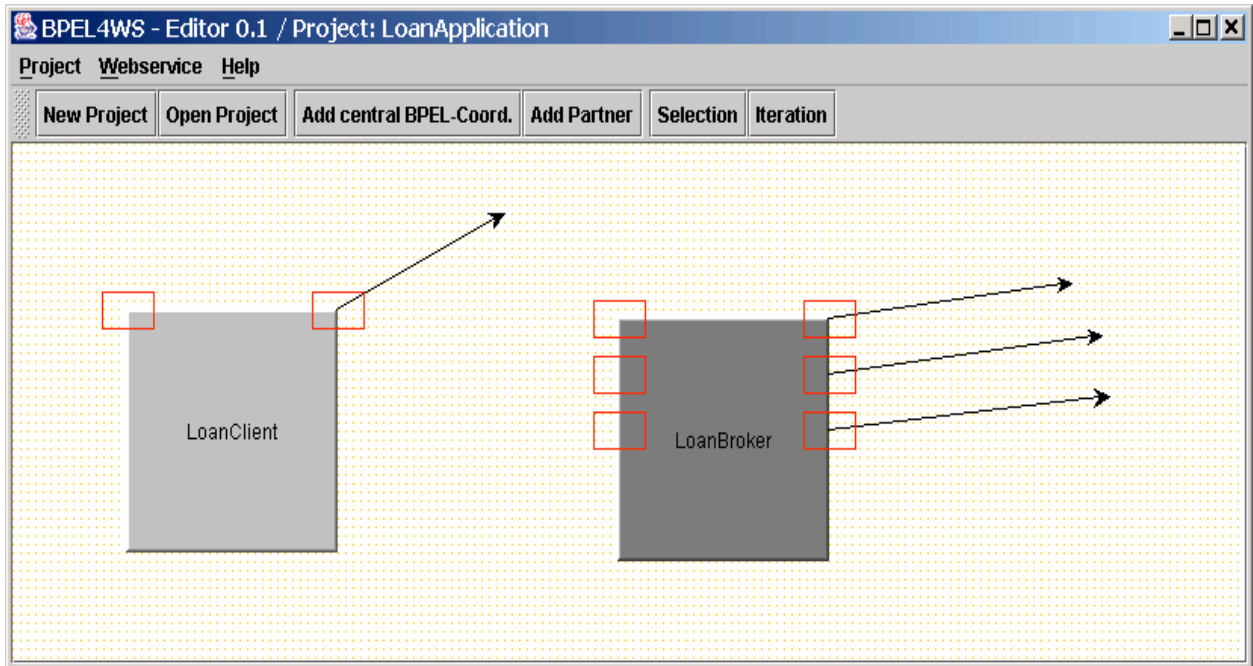
4.1.3.1. Modelling steps using the BPEL-editor

1. Reading in the WSDL file of the Central BPEL-coordinator



BPEL – Editor

2. Reading in of the Loan Client's WSDL-file



BPEL – Editor

3. Connecting the outgoing edge of the Loan Client with the input port of the Loan Broker

This models the call of one the operations provided by the Loan Broker in his WSDL interface description. It invokes the following dialogue for the creation of a <partnerLink>- and <receive>-element as well as an optional <assign>-element.

The screenshot shows a dialog box titled "Webservice-Partner calls BPEL-Process / <receive>". It is divided into three main sections: <partnerLink> - Element, <receive> - Element, and <assign> - Element.

<partnerLink> - Element

- Partner Link Name: LoanClient
- Partner Link Type: ClientLoanFlow, LoanBroker
- Roles: LoanBrokerService (checked), LoanBrokerRequester
- My Role: LoanBrokerService (checked), LoanBrokerRequester
- Partner Role: LoanBrokerService, LoanBrokerRequester (checked)

<receive> - Element

- Step no.: 1
- Receive name: receiveInputLoanClient
- Partner Link: LoanClient
- Porttype: LoanBroker_PT
- Operation: initiate (selected), pollResult
- Variable: input
- Input Message: initiateLoanFlowSoapRequest
- Create instance: Yes (selected), No

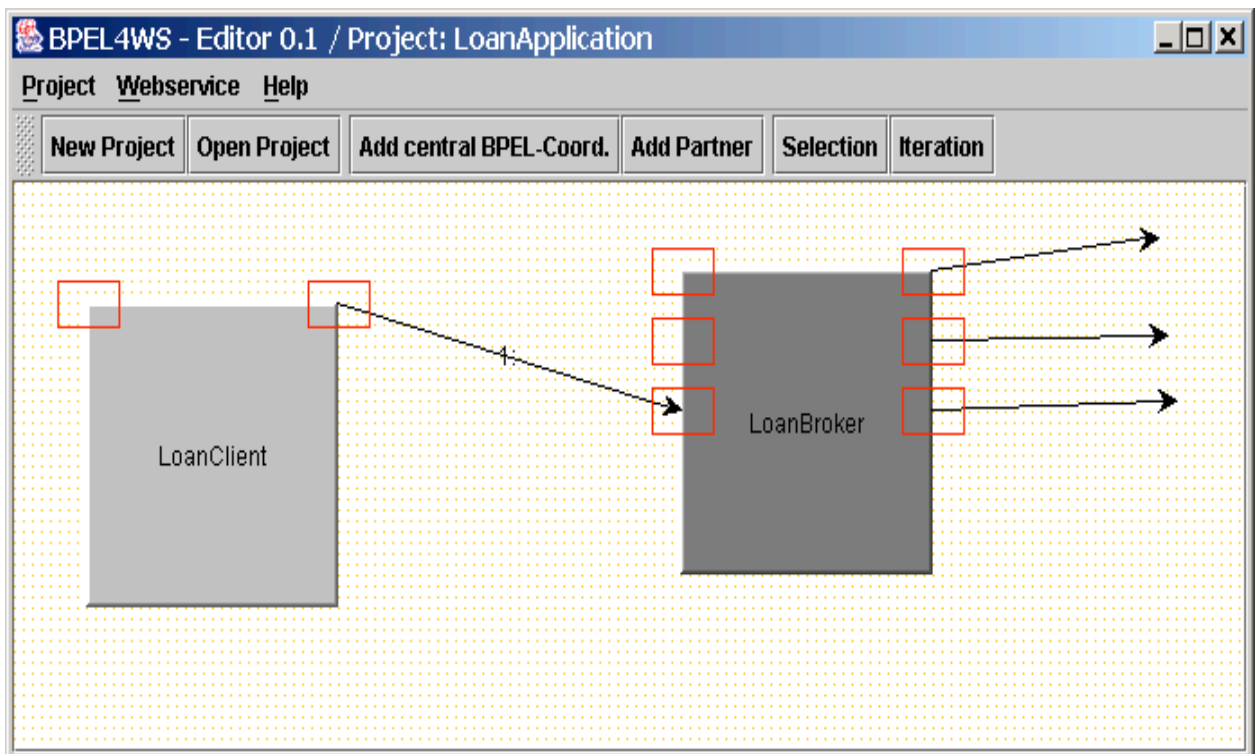
<assign> - Element

- Add <assign>-element
- Step no.: 2
- <copy>
 - <from>
 - expression: [empty]
 - variable: input
 - Input-Msg: initiateLoanFlowSoapRequest
 - <to>
 - variable: loanApplication
 - Input-Msg: initiateLoanServiceSoapRequest
- part: parameters
- query: //xmlLoanApp />
- part: parameters
- query: /initiateLoanService/xmlLoanApp />

Buttons: OK, Cancel

BPEL – Editor

After completing the dialogue (clicking the OK-button) a step number is assigned to the connecting edge.



BPEL – Editor

3. Connecting an outgoing edge of the Loan Broker with the credit-offering bank Star Loan
 This models the call of the first of the two credit-offering banks for getting an interest rate offer. In the ensuing dialogue a <partnerLink>- and an <invoke>-element as well as an optional <assign>-element are created.

BPEL-Process calls Webservice-Partner / <invoke>
✕

<partnerLink> - Element

Partner Link Name:

Partner Link Type	Roles	My Role	Partner Role
LoanBroker	LoanServiceService	<input type="checkbox"/>	<input checked="" type="checkbox"/>
LoanService	LoanServiceRequester	<input checked="" type="checkbox"/>	<input type="checkbox"/>

<invoke> - Element

Step no.:

Invoke name:

Porttype: ▼

Partner Link:

Operation: ▼

Input Variable:

Input-Message: ▼

Output Variable:

Output-Message: ▼

<assign> - Element

Add <assign>-element Step no.:

<copy>

<from>	variable: <input type="text"/>	part: <input type="text"/>	
	Input-Msg: <input style="border: none; background-color: #eee; padding: 2px;" type="text" value="initiateLoanBrokerSoapRequest"/> ▼	query: <input style="border: none; background-color: #eee; padding: 2px;" type="text"/> />	
<to>	variable: <input type="text"/>	part: <input type="text"/>	
	Input-Msg: <input style="border: none; background-color: #eee; padding: 2px;" type="text" value="initiateLoanBrokerSoapRequest"/> ▼	query: <input style="border: none; background-color: #eee; padding: 2px;" type="text"/> />	

</copy>

BPEL – Editor

4. Connecting an outgoing edge of the Loan Broker with credit-offering bank United Loan
 This models the call of the second of the two credit-offering banks for getting an interest rate offer. In the ensuing dialogue a `<partnerLink>`- and an `<invoke>`-element as well as an optional `<assign>`-element are created. It is basically the same step as the previous one.

BPEL-Process calls Webservice-Partner / <invoke>

<partnerLink> - Element

Partner Link Name:

Partner Link Type	Roles	My Role	Partner Role
LoanBroker	LoanServiceService	<input type="checkbox"/>	<input checked="" type="checkbox"/>
LoanService	LoanServiceRequester	<input checked="" type="checkbox"/>	<input type="checkbox"/>

<invoke> - Element

Step no.:

Invoke name:

Porttype:

Partner Link:

Operation:

Input Variable:

Input-Message:

Output Variable:

Output-Message:

<assign> - Element

Add <assign>-element Step no.:

<copy>

<from>	variable: <input type="text"/>	part: <input type="text"/>	
	Input-Msg: <input type="text" value="initiateLoanBrokerSoapRequest"/>	query: <input type="text"/>	/>
<to>	variable: <input type="text"/>	part: <input type="text"/>	
	Input-Msg: <input type="text" value="initiateLoanBrokerSoapRequest"/>	query: <input type="text"/>	/>

</copy>

5. Inserting a Selection-GUI-Element and connecting a StarLoan outgoing edge and a UnitedLoan outgoing edge with one of the two Selection-Input-Ports

After both outgoing edges are connected to the selection-GUI-control the following dialogue will be shown. This models an asynchronous communication between the Loan Broker and both credit-offering banks. It also creates a BPEL selection construct (<switch>-/<case>-elements).

Selection – Dialog:

Selection Dialogue

Input - Port-1
 Step no.: 3
 <invoke> - Element 1
 Partner-Link: StarLoan
 Porttype: LoanService Operation: initiate
 Input Var: loanApplication Output Var:
 <receive> - Element-1
 Name: receive_invokeStarLoan Pt-Link: StarLoan
 Porttype: LoanBrokerCallback_PT Operation: onResult
 Variable: loanOffer1 Input-Msg: initiateLoanServiceSoap...

Input - Port-2
 Step no.: 4
 <invoke> - Element 2
 Partner-Link: UnitedLoan
 Porttype: LoanService Operation: initiate
 Input Var: loanApplication Output Var:
 <receive> - Element-2
 Name: receive_invokeUnitedLoan Pt-Link: UnitedLoan
 Porttype: LoanBrokerCallback_PT Operation: onResult
 Variable: loanOffer2 Input-Msg: initiateLoanServiceSoap...

Condition
 BPEL-function-1: 'getVariableData' - parameters
 Variablename 1: loanOffer1 Operator: >
 Partname-1: parameters
 Location path-1: result/APR
 BPEL-function-2: 'getVariableData' - parameters
 Variablename 2: loanOffer2
 Partname-2: parameters
 Location path-2: result/APR

THEN - activity
 Assign
 <copy>
 <from> variable: loanOffer1 part: parameters
 query: result />
 <to> variable: selectedLoanOffer part: parameters
 Input-Msg: onLoanServiceResultSo...
 query: /onLoanFlowResult/result />
 </copy>

ELSE - activity
 Assign
 <copy>
 <from> variable: loanOffer2 part: parameters
 query: result />
 <to> variable: selectedLoanOffer part: parameters
 Input-Msg: onLoanServiceResultSo...
 query: onLoanFlowResult/result />
 </copy>

OK Cancel

BPEL – Editor

7) Finally connecting an outgoing edge of the Loan Broker with the Loan Client

This models the handing over of the best of the two interest offers to the Loan Client. In the ensuing dialogue an `<invoke>` element is created.

The screenshot shows the BPEL-Process editor interface for configuring an `<invoke>` element. The window title is "BPEL-Process calls Webservice-Partner / <invoke>".

<partnerLink> - Element

Partner Link Name:

Partner Link Type	Roles	My Role	Partner Role
LoanBroker	LoanBrokerService	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ClientLoanFlow	LoanBrokerRequester	<input type="checkbox"/>	<input checked="" type="checkbox"/>

<invoke> - Element

Step no.:

Invoke name:

Porttype:

Input Variable:

Output Variable:

Partner Link:

Operation:

Input-Message:

Output-Message:

<assign> - Element

Add <assign>-element

Step no.:

<copy>

<from>	variable: <input type="text"/>	part: <input type="text"/>
	Input-Msg: <input type="text" value="initiateLoanBrokerSoapRequest"/>	query: <input type="text" value="/>"/>
<to>	variable: <input type="text"/>	part: <input type="text"/>
	Input-Msg: <input type="text" value="initiateLoanBrokerSoapRequest"/>	query: <input type="text" value="/>"/>

</copy>

OK Cancel

4.1.4. Synchroner While – Service

This example demonstrates an application of the BPEL <while> element without considering a concrete business background

4.1.4.1. Participating Web Service Partners

1. Client (Requestor)
2. Remote Increment Service
3. BPEL-While Service

The BPEL-While Service acts as the BPEL central coordinator and interacts with both the client and the Remote Increment Service.

4.1.4.2. Process flow

Goal of the process:

A synchronous iterator will be called n-times until an initial value is received, i.e. until a preset limit is exceeded.

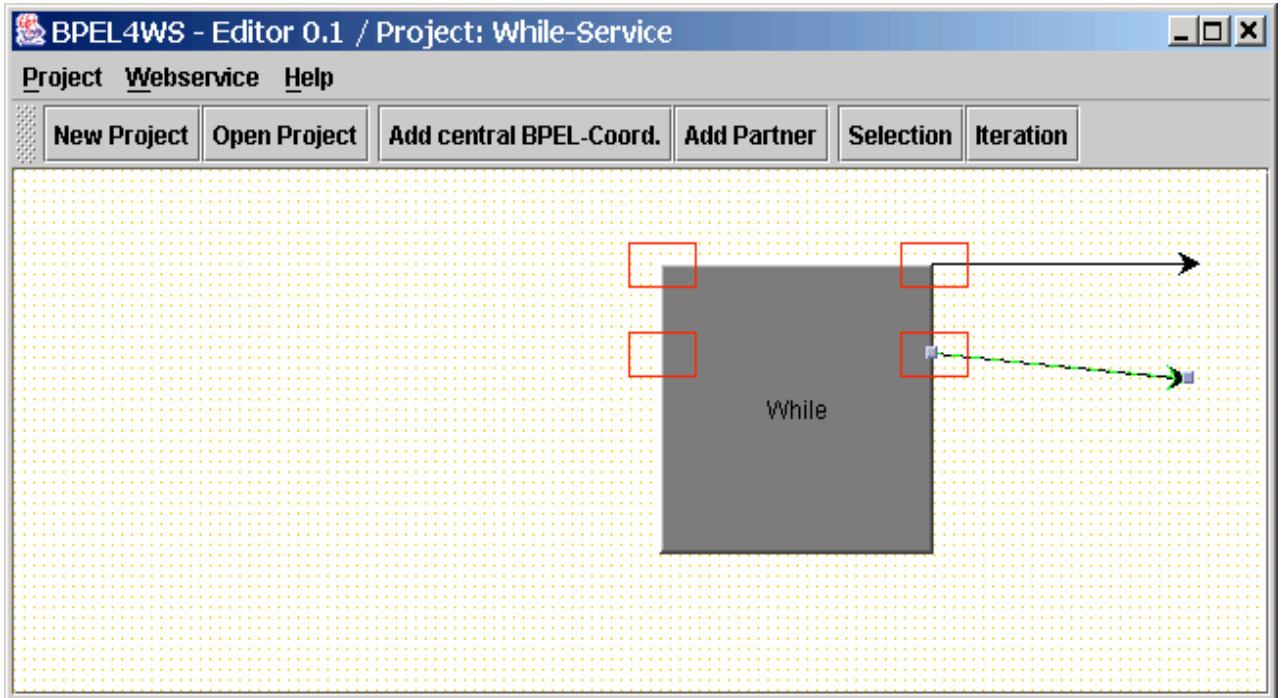
1. <receive>: The BPEL-coordinator receives the input from the requestor. This starts the process. The limit is passed in the <receive>-element variable „input“.
2. <assign>: Now the BPEL-coordinator copies the initial value of 0 into the variable „request“ for comparison.
3. <while>: While-Iteration
 - a) As long as the limit „input“ is greater or equal to „request“ the following steps are executed:
 - b) Call of the Remote Increment Service, passing of the variable „request“ and the output variable „response“ for the synchronous reply.
 - c) Assigning (<assign>/<copy>) the answer in „response“ to the variable „request“
 - d) Go to step a).
4. <assign>/<copy>: Assigning of the final value (above the preset limit) in „response“ to the variable „output“.
5. <invoke>: Returning the final „output“-value to the requestor.

4.1.4.3. Preconditions for modelling with the BPEL-Editor

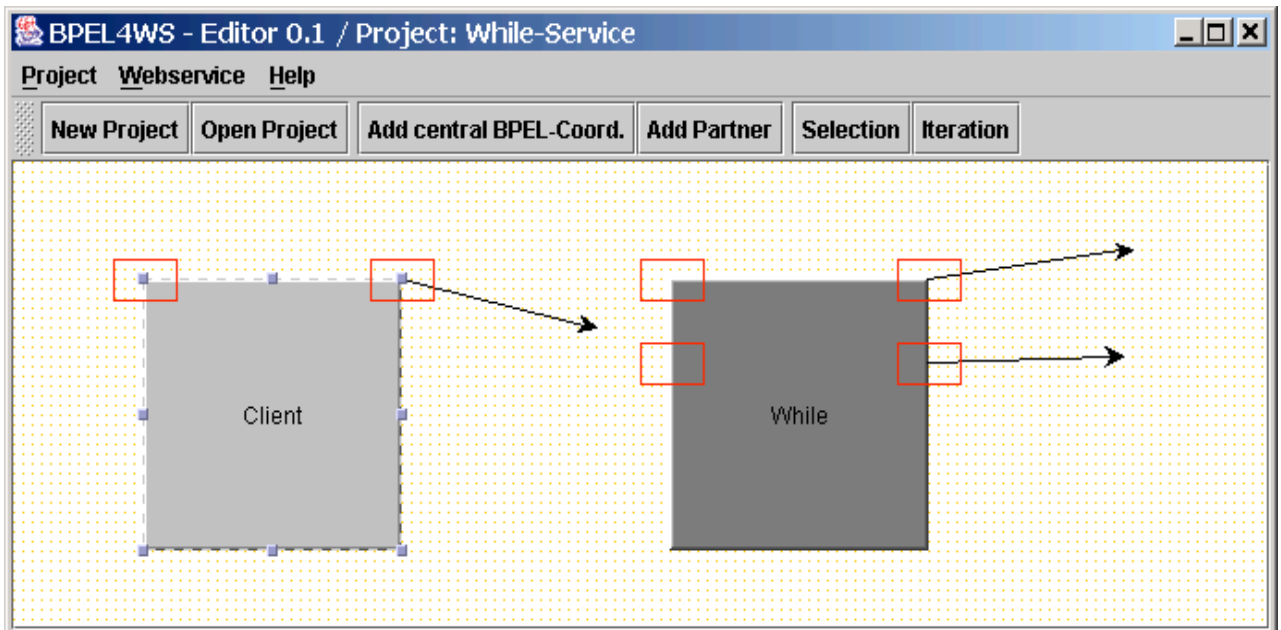
For each participating Web Service partner there has to be a WSDL file. The same holds for the requestor.

4.1.4.4. Modelling steps using the BPEL-editor

1. Reading in of the central BPEL-coordinator's WSDL-file (= While-Service)



2. Reading in of the requestor's WSDL-file (= client)



BPEL – Editor

3. Connecting the outgoing edge of the requestor with the input port of the While-Service

This models the call of one the operations provided by the While-Service in its WSDL interface description. It invokes the following dialogue for the creation of a <partnerLink>- and <receive>-element as well as an optional <assign>-element.

The screenshot shows a dialog box titled "Webservice-Partner calls BPEL-Process / <receive>". It is divided into three main sections: "<partnerLink> - Element", "<receive> - Element", and "<assign> - Element".

<partnerLink> - Element

Partner Link Name: Client

Partner Link Type	Roles	My Role	Partner Role
ClientService	WhileService	<input checked="" type="checkbox"/>	<input type="checkbox"/>
While	WhileRequester	<input type="checkbox"/>	<input checked="" type="checkbox"/>

<receive> - Element

Step no.: 1

Receive name: receiveInput

Partner Link: Client

Porttype: While

Operation: initiate

Variable: input

Input Message: initiateWhileSoapRequest

Create instance: Yes No

<assign> - Element

Add <assign>-element

Step no.: 2

<copy>

<from>	expression:	variable:	Input-Msg:	part:	query:
	0		initiateLoanFlowSoapRequest		/>
<to>		request	processIncrementServiceSoapRequest	parameters	/processIncrementService/value />

</copy>

OK Cancel

BPEL – Editor

4. Insertion of the Iteration-GUI-element to connect it with the BPEL-coord. and the Web Service to be called

After inserting the iteration-GUI-element into the model an outgoing edge of the While-Service must be connected with the input port of the iteration-GUI-element. Now the to-be-called Web Service must be inserted into the model and the outgoing edge of the iteration-GUI-element must be connected with the input port of the Web Service. After that the following dialogue will be shown:

The screenshot shows a dialog box titled "BPEL-Process calls Webservice-Partner / <invoke>". It is divided into three main sections:

- <partnerLink> - Element:**
 - Partner Link Name: IncrementService
 - Table of Partner Link Types:

Partner Link Type	Roles	My Role	Partner Role
While		<input type="checkbox"/>	<input checked="" type="checkbox"/>
IncrementService	IncrementServiceService IncrementServiceRequester	<input checked="" type="checkbox"/>	<input type="checkbox"/>
- WHILE - Condition:**
 - BPEL-function-1: 'getVariableData' - parameters
 - Variable name 1: input
 - Part name-1: parameters
 - Location path-1: /initiateWhile/value
 - Operator: >=
 - BPEL-function-2: 'getVariableData' - parameters
 - Variable name 2: request
 - Part name-2: parameters
 - Location path-2: /processIncrementService/value
- <invoke> - Element:**
 - Step no.: 3
 - Invoke name: invokeIncrService
 - Port type: IncrementService
 - Input Variable: request
 - Output Variable: response
 - Partner Link: IncrementService
 - Operation: process
 - Input-Message: processIncrementServiceSoapRequest
 - Output-Message: processIncrementServiceSoapResponse
- <assign> - Element:**
 - Add <assign>-element
 - <copy>
 - <from>
 - variable: response
 - Input-Msg: processIncrementServiceSoapRequest
 - <to>
 - variable: request
 - Input-Msg: processIncrementServiceSoapRequest
 - </copy>
 - part: parameters
 - query: /result />
 - part: parameters
 - query: /processIncrementService/value />

Buttons: OK, Cancel

BPEL – Editor

5. Connecting an outgoing edge of the While-BPEL-Koordinators with the requestor:

This models the return of the value requested from the Increment-Service to the requestor (client) and terminates the process.

The screenshot shows the BPEL Editor interface with the following configuration:

<partnerLink> - Element

Partner Link Name: Client

Partner Link Type	Roles	My Role	Partner Role
While	WhileService	<input checked="" type="checkbox"/>	<input type="checkbox"/>
ClientService	WhileRequester	<input type="checkbox"/>	<input checked="" type="checkbox"/>

<invoke> - Element

Step no.: 5

Invoke name: replyOutput

Porttype: WhileCallback

Input Variable: output

Output Variable:

Partner Link: Client

Operation: onResult

Input-Message: onWhileResultSoapRequest

Output-Message: initiateWhileSoapRequest

<assign> - Element

Add <assign>-element

Step no.: 4

<copy>

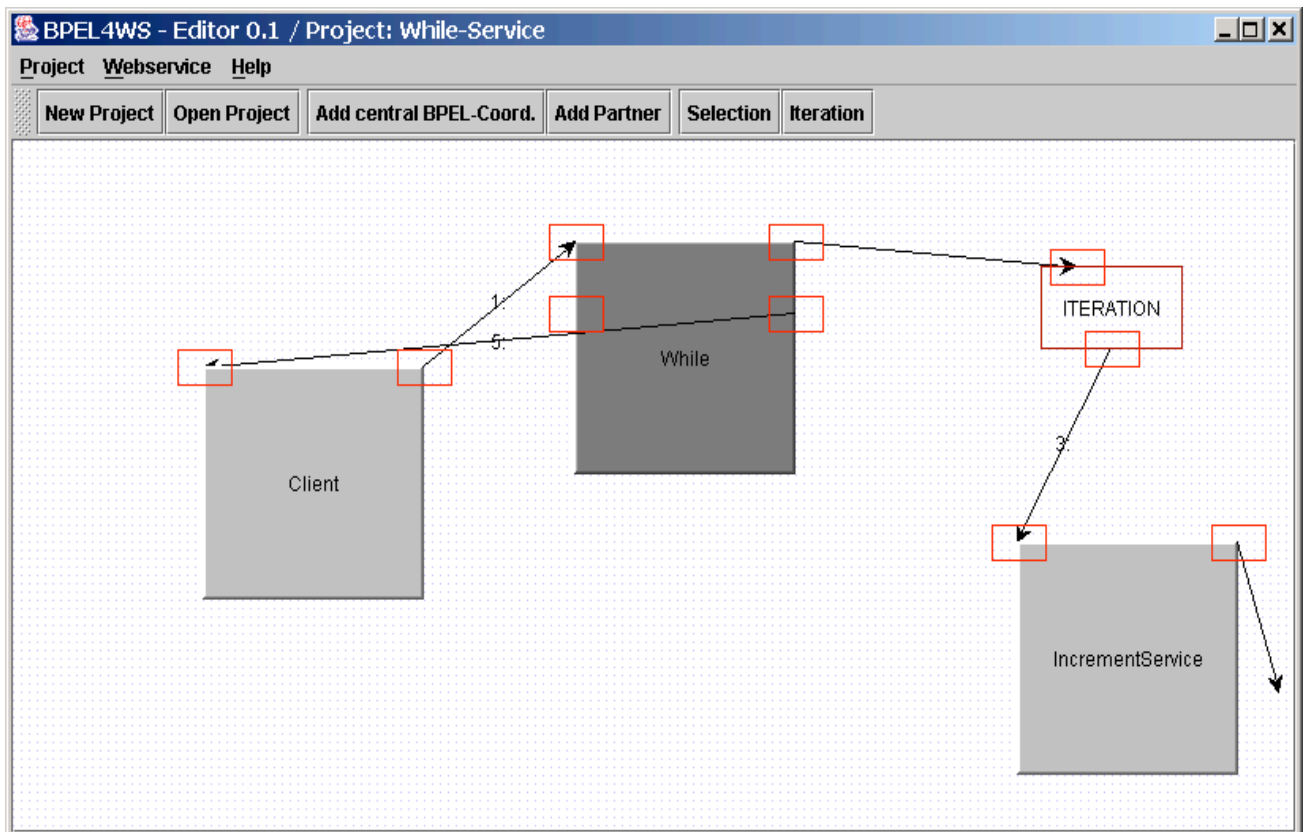
<p><from></p> <p>variable: response</p> <p>Input-Msg: processIncrementServiceSoapResponse</p>	<p>part: parameters</p> <p>query: /result /></p>
<p><to></p> <p>variable: output</p> <p>Input-Msg: onWhileResultSoapRequest</p>	<p>part: parameters</p> <p>query: /onWhileResult/result /></p>

</copy>

OK Cancel

BPEL – Editor

Finally this results in the following graphical representation of the While-Service-process:



While-BPEL-process – XML output file:

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="While-Service" suppressJoinFailure="yes">
  <partnerLinks>
    <partnerLink myRole="WhileService" name="Client"
      partnerLinkType="While" partnerRole="WhileRequester"/>
    <partnerLink myRole="IncrementServiceRequester"
      name="IncrementService" partnerLinkType="IncrementService"
      partnerRole="IncrementServiceService"/>
  </partnerLinks>
  <variables>
    <variable messageType="processIncrementServiceSoapRequest" name="request"/>
    <variable messageType="initiateWhileSoapRequest" name="input"/>
    <variable messageType="processIncrementServiceSoapRequest" name="response"/>
    <variable messageType="onWhileResultSoapRequest" name="output"/>
  </variables>
  <sequence>
    <sequence name="1">
      <receive createInstance="yes" name="receiveInput"
        operation="initiate" partnerLink="Client"
        portType="While" variable="input"/>
    </sequence>
    <sequence name="2">
      <assign>
        <copy>
          <from expression="0"/>
          <to part="parameters"
            query="/processIncrementService/value" variable="request"/>
        </copy>
      </assign>
    </sequence>
    <sequence name="3">
      <while condition="getVariableData('input','parameters','/initiateWhile/value') >=
        getVariableData('request','parameters','/processIncrementService/value')">
        <sequence>
          <invoke inputVariable="request"
            name="invokeIncrService" operation="process"
            outputVariable="response"
            partnerLink="IncrementService" portType="IncrementService"/>
          <assign>
            <copy>
              <from part="parameters" query="/result" variable="response"/>
              <to part="parameters"
                query="/processIncrementService/value" variable="request"/>
            </copy>
          </assign>
        </sequence>
      </while>
    </sequence>
    <sequence name="4">
```

```
<assign>
  <copy>
    <from part="parameters" query="/result" variable="response"/>
    <to part="parameters" query="/onWhileResult/result" variable="output"/>
  </copy>
</assign>
</sequence>
<sequence name="5">
  <invoke inputVariable="output" name="replyOutput"
    operation="onResult" partnerLink="Client" portType="WhileCallback"/>
</sequence>
</sequence>
</process>
```

5. Some proposals of new and additional features for a new implementation

- Web Services / UDDI – Explorer
 - This should allow the down- and upload from and to a UDDI registry
- Database for storing the BPEL process model instead of and/or in addition to storing it in an XML-file (serialisation)
- Comprehensive BPEL language support with fine grained mapping to corresponding graphical elements
 - For this to work a set of graphical elements and a mapping to the corresponding BPEL language elements (= their BPEL semantics) has to be defined
- More fine grained (“smaller”) dialogues providing more design flexibility for the experienced BPEL process modeller. At least on a conceptual level there is probably a strong interrelation between these “smaller” dialogues, the above mentioned BPEL language support and its mapping to graphical elements.
- Wizards for routine modelling steps

6. IDE, Tools, APIs

1. Eclipse 2.1.2 on Windows 2000 Prof. (<http://www.eclipse.org>)
2. Java JRE 1.4.1
3. APIs:
 - a) Apache Xerces 2.4.0 (<http://xml.apache.org>)
 - b) JGraph 2.1.1 (<http://jgraph.sourceforge.net>)
4. GUI : Sun Swing

7. Literatur, Links

[TIDWELL00]	Doug Tidwell, Web Services – The Web’s next revolution , IBM November 2000
[GOTTSCHE02]	Gottschalk et.al., Introduction to Web services architecture , IBM Systems Journal, Vol 41, No. 2, 2002
[OREILLY02]	O’Reilly Verlag, Top Ten FAQs for Web Services , http://webservices.xml.com/pub/a/ws/2002/02/12/webservicefaqs.html
[KREG01]	Heather Kreger, IBM Software Group, Web Services Conceptual Architecture (WSCA 1.0) , May 2001
[WSDL]	Web Service Definition Language 1.1, Specification , http://www.w3.org/TR/2001/NOTE-wsdl-20010315.html
[BPEL]	Business Process Execution Language 4 Web Services 1.1, Specification , http://www-106.ibm.com/developerworks/library/ws-bpel/
[PELTZ]	Chris Peltz, Hewlet Packard, January 2003, Web Service Orchestration – A review of emerging technologies, tools and standards