BPEL Extensions for Compliant Services

Project acronym: COMPAS
Project name: Compliance-driven Models, Languages, and Architectures for Services
Call and Contract: FP7-ICT-2007-1
Grant agreement no.: 215175
Project Duration: 01.02.2008 – 28.02.2011 (36 months)
Co-ordinator: TUV Technische Universitaet Wien (AT)
Partners: CWI Stichting Centrum voor Wiskunde en Informatica (NL)
UCBL Université Claude Bernard Lyon 1 (FR)
USTUTT Universitaet Stuttgart (DE)
TILBURG UNIVERSITY Stichting Katholieke Universiteit Brabant (NL)
UNITN Universita degli Studi di Trento (IT)
TARC-PL Telcordia Poland (PL)
THALES Thales Services SAS (FR)
PWC Pricewaterhousecoopers Accountants N.V. (NL)

This project is supported by funding from the Information Society Technologies Programme under the 7th Research Framework Programme of the European Union.
Project no. 215175

COMPAS

Compliance-driven Models, Languages, and Architectures for Services

Specific Targeted Research Project
Information Society Technologies

Start date of project: 2008-02-01   Duration: 36 months

D4.2 BPEL Extensions for Compliant Services

Revision 1.0

Due date of deliverable: 2009-12-31
Actual submission date: 2009-12-22

Organisation name of lead partner for this deliverable:
USTUTT Universitaet Stuttgart, DE

Contributing partner(s):
TUV Technische Universitaet Wien, AT
CWI Stichting Centrum voor Wiskunde en Informatica, NL
THALES Thales Services SAS, FR
TILBURG UNIVERSITY Stichting Katholieke Universiteit Brabant, NL
UCBL Université Claude Bernard Lyon 1, FR

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>PU</th>
<th>PP</th>
<th>RE</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted to other programme participants (including the Commission Services)</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted to a group specified by the consortium (including the Commission Services)</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Confidential, only for members of the consortium (including the Commission Services)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Project funded by the European Commission within the Seventh Framework Programme
History chart

<table>
<thead>
<tr>
<th>Issue</th>
<th>Date</th>
<th>Changed page(s)</th>
<th>Cause of change</th>
<th>Implemented by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2009-06-22</td>
<td>All sections</td>
<td>New document</td>
<td>USTUTT</td>
</tr>
<tr>
<td>0.2</td>
<td>2009-08-15</td>
<td>All sections</td>
<td>Added outline</td>
<td>USTUTT</td>
</tr>
<tr>
<td>0.3</td>
<td>2009-09-15</td>
<td>All sections</td>
<td>Restructuring</td>
<td>USTUTT</td>
</tr>
<tr>
<td>0.4</td>
<td>2009-10-30</td>
<td>All sections</td>
<td>Refined outline</td>
<td>USTUTT</td>
</tr>
<tr>
<td>0.5</td>
<td>2009-11-30</td>
<td>All sections</td>
<td>First complete version</td>
<td>USTUTT, TUV</td>
</tr>
<tr>
<td>0.6</td>
<td>2009-12-10</td>
<td>All sections</td>
<td>Refinements</td>
<td>USTUTT, TUV, CWI, TILBURG UNIVERSITY</td>
</tr>
<tr>
<td>0.7</td>
<td>2009-12-15</td>
<td>All sections</td>
<td>Review comments have been addressed</td>
<td>USTUTT, THALES, UCBL</td>
</tr>
<tr>
<td>0.8</td>
<td>2009-12-17</td>
<td>All sections</td>
<td>Review comments have been addressed</td>
<td>USTUTT, TUV</td>
</tr>
<tr>
<td>0.9</td>
<td>2009-12-18</td>
<td>All sections</td>
<td>Review comments have been addressed</td>
<td>USTUTT, CWI</td>
</tr>
<tr>
<td>1.0</td>
<td>2009-12-22</td>
<td></td>
<td>Approve &amp; release</td>
<td>TUV</td>
</tr>
</tbody>
</table>

Authorisation

<table>
<thead>
<tr>
<th>No.</th>
<th>Action</th>
<th>Company/Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prepared</td>
<td>USTUTT</td>
<td>2009-12-21</td>
</tr>
<tr>
<td>2</td>
<td>Approved</td>
<td>TUV</td>
<td>2009-12-22</td>
</tr>
<tr>
<td>3</td>
<td>Released</td>
<td>TUV</td>
<td>2009-12-22</td>
</tr>
</tbody>
</table>

Disclaimer: The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

All rights reserved.

The document is proprietary of the COMPAS consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights.

This document reflects only the authors’ view. The European Community is not liable for any use that may be made of the information contained herein.
Contents

1. Introduction ................................................................................................................................. 6
   1.1. Purpose and scope................................................................................................................ 6
   1.2. Document overview............................................................................................................. 7
   1.3. Definitions and glossary...................................................................................................... 7
   1.4. Abbreviations and acronyms............................................................................................... 7
2. Integration into the overall COMPAS architecture ................................................................. 8
3. Extending BPEL ....................................................................................................................... 10
   3.1. BPEL extension background............................................................................................. 10
   3.2. BPEL extension formality ................................................................................................. 11
       3.2.1. Extension declaration ................................................................................................. 11
       3.2.2. Extension attributes .................................................................................................... 12
       3.2.3. Extension elements ..................................................................................................... 12
       3.2.4. Extension assign operations ....................................................................................... 13
       3.2.5. Extension activities .................................................................................................... 13
4. Process fragments for compliance ........................................................................................... 14
   4.1. Towards reusable implementations of compliance requirements ................................... 14
   4.2. Analysis of concepts for reusability and compliance ...................................................... 15
   4.3. Mapping of compliance concepts to process fragments .................................................. 16
   4.4. Concept for process fragments for compliance ............................................................... 18
5. BPEL extensions for compliant services ................................................................................. 20
   5.1. BPEL extension for process fragments ............................................................................ 20
       5.1.1. Process fragment meta model .................................................................................... 20
       5.1.2. Process fragment meta model for BPEL ................................................................. 27
       5.1.3. Process fragment XML schema ................................................................................. 28
       5.1.4. Process fragment XML example ............................................................................... 32
       5.1.5. Transformation of process fragments to standard BPEL code ................................ 35
   5.2. BPEL extension for traceability ........................................................................................ 37
6. Prototype implementation details ............................................................................................ 41
7. Conclusion ................................................................................................................................. 43
8. Reference documents ................................................................................................................ 44
   8.1. Internal documents ............................................................................................................ 44
   8.2. External documents ........................................................................................................... 44
A. Appendix A .............................................................................................................................. 47
   A.1. Related standards and technologies ................................................................................. 47
List of figures

Figure 1  Overall COMPAS architecture ................................................................. 9
Figure 2  BP Logic Fragment for approval ............................................................... 15
Figure 3  Refined BP Logic Fragment for approval ................................................. 19
Figure 4  Process fragment meta model as entity-relationship diagram .................. 21
Figure 5  WS-BPEL 2.0 meta model [Dub07] ......................................................... 22
Figure 6  Viewing a process fragment with standard BPEL constructs ................... 35

List of tables

Table 1  Entity sets of process fragment meta model .............................................. 24
Table 2  Relationship sets of process fragment meta model ................................... 26
Table 3  Mapping of technology independent meta model to BPEL ......................... 28
Table 4  Related standards and technologies ......................................................... 47

List of listings

Listing 1  BPEL extensions declaration example ................................................. 12
Listing 2  Extension attribute example ............................................................... 12
Listing 3  Extension element example ............................................................... 13
Listing 4  Extension activity example ............................................................... 13
Listing 5  XML schema for process fragment extensions ..................................... 31
Listing 6  XML schema for unique identifiers ..................................................... 32
Listing 7  Process fragment for approval in BPEL code ....................................... 34
Listing 8  XSL style sheet for process fragment transformation to standard BPEL code... 37
Listing 9  Batch code for process fragment transformation via drag & drop .......... 37
Listing 10  BPEL process with an extension for MORSE traceability ..................... 39
Listing 11  XML schema for MORSE traceability ................................................. 40
Abstract

This deliverable defines extensions to the Business Process Execution Language (BPEL) in order to enable the implementation of the compliant service composition specifications. For this task we are building on the solution concept outlined in deliverable D4.1 which is called process fragment. Our approach aims at ensuring a faster and more consistent specification and integration of compliance checks into business processes. In this deliverable, we elaborate on the background, the solution concept and provide implementation details in order to define a solution for reusability and compliance in BPM.

1. Introduction

1.1. Purpose and scope

In the former COMPAS deliverable [D4.1] we have discussed state-of-the-art approaches for improving reusability in the field of processes and service compositions. These approaches can basically also be applied to enrich business processes with compliance. All of these approaches have certain advantages, but also drawbacks in their application in practice, e.g., process variants enable customization and reuse of an entire process model, but do not allow the reuse of only one specific part or artefact of a service composition. Our analysis has shown that none of the examined approaches is on its own sufficient to provide a solution for augmenting business processes with compliance in order to cover all compliance requirements described in the COMPAS use cases, cf. Sections 1.2 and 1.3 of [D6.3].

Moreover, there is no universally accepted standard or method in the area of Business Process Management (BPM) that enables a flexible management of compliance requirements addressing the whole business process life cycle. Thus, we have investigated the application of the concept of process fragments to the field of compliance of processes and service compositions, while incorporating reusability. For the time being we have identified two main usage scenarios for the application of process fragments. On the one hand process fragments can be used during design as reusable building blocks. They can either be glued into an existing process model, or they can be used to build up a process from scratch. On the other hand process fragments can be used as annotation language in order to constrain the behaviour of process instances during execution which can be checked within the subsequent monitoring as well as mining phase. In the COMPAS terminology we have introduced two terms for distinguishing between the different usage scenarios (cf. terminology on [D7.1]). The term Business Process Logic Fragment (BPLF) addresses the usage as reusable building blocks and Annotation Business Process Fragment (ABPF) addresses the usage for constraining behaviour. In this document, we are focusing BPLFs and the extensions to BPEL which are required for supporting them. The concept of ABPFs is more related to monitoring and thus discussed in [D5.4].

The purpose of this document is to specify and define extensions to the BPEL language in order to enable the implementation of the compliant service composition specifications. Furthermore, this deliverable provides the basis for one of the two specifications of the COMPAS project regarding standardisation proposals. The D4.2 prototype components for dealing with the BPEL extensions are integrated in the supporting infrastructure (cf. prototype [D4.4]). D4.2 provides prototypes in form of XSDs and examples for these BPEL extensions, APIs that demonstrate how to handle these extension e.g., in process engines or repositories, and a fragment transformer tool (cf. Section 5.1.5). [D4.4] provides an integrated runtime that can run on its own. It includes the APIs from D4.2.
The checking of process fragments within the monitoring and mining phase and according extensions of the event model are not within the scope of this deliverable. These are described in [D5.4].

1.2. Document overview

In Section 2 we give the reader an overview of the meaning of this deliverable for the overall concept in COMPAS and how it is related to other work packages. In the main part of this deliverable we discuss three distinct aspects of BPEL extensions for compliant services. The first aspect (Section 3) describes the ways and methodology of extending BPEL in general. This includes a survey on prominent BPEL extensions, followed by a discussion of the formalities for the specification of a standard-conform extension. The second aspect (Section 4) elaborates on the concept of process fragments for compliance at a high level of abstraction. We discuss the requirements and constraints which we want to address with our concept and we also motivate the extensions which we finally propose. The third aspect (Section 5) is concerned with the actual BPEL extensions for compliant services. In this section, we discuss the meta model for process fragments for compliance and we discuss how this meta model maps to BPEL. For enabling the infrastructure to manage and handle the fragments we also provide an XML Schema and an XML example for the BPEL extensions for process fragments. Subsequently we discuss the extensions to BPEL for enabling traceability.

1.3. Definitions and glossary

The most important terminology concerning the COMPAS project is listed on the public COMPAS Web-Site [D7.1] available at http://www.compas-ict.eu section terminology. In the following the definitions of terms valid only in the scope of this deliverable are specified:

*Standard-conform BPEL extension:* A standard-conform BPEL extension is an enhancement of functionality of the Web Services Business Process Execution Language specified in the OASIS WS-BPEL 2.0 standard by following the extension proceedings defined in the standard. On its own, the BPEL extension is not useful or functional.

1.4. Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABPF</td>
<td>Annotation Business Process Fragment</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td>BPLF</td>
<td>Business Process Logic Fragment</td>
</tr>
<tr>
<td>BPM</td>
<td>Business Process Management</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Modeling Notation</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain-Specific Language</td>
</tr>
<tr>
<td>ERD</td>
<td>Entity-Relationship Diagram</td>
</tr>
<tr>
<td>MDD</td>
<td>Model-Driven Development</td>
</tr>
<tr>
<td>MORSE</td>
<td>Model-Aware Service Environment</td>
</tr>
<tr>
<td>UUID</td>
<td>Universally Unique Identifier</td>
</tr>
</tbody>
</table>
2. Integration into the overall COMPAS architecture

We envision process fragments as a solution for specifying or constraining control flow and data flow within a process. For realising our concept of reusable process fragments for compliance we leverage the extensibility of the de facto standard for Web service orchestration (BPEL, [OASIS07]). The requirements that we can address with process fragments are focussed on the process, however, compliance also has an impact on the applications and humans which are involved in the process. For instance, a compliance requirement that demands encrypted storage of customer billing data for at least seven years cannot be realised solely by a process fragment, as this requirement is related to a database that is external to the process engine that executes the fragment. Annotations using domain-specific languages (DSLs) can be seen as a complementing solution concept for achieving overall compliance. Actually, process fragments for compliance can be seen as a DSL as well, as its meta model defines a language specific for the domain of BPEL process structures for compliance.

Before a process fragment can be used for implementing a compliance requirement within a process, it has to be created. One possibility for creating a process fragment is to have a domain expert who analyses already implemented process models and recognises repeating patterns which have the potential of reusability. For this, all activities which have been identified as relevant for a specific purpose, the corresponding control structures and the context (such as variables or partner links) have to be extracted. After extraction, the activities of the fragment may need to be reordered and attributes on activities may need to be removed or parameterized, in order to make it more general. Another possibility for creating a process fragment is building it from scratch, based on the specification of its purpose and the functional and non-functional requirements it shall meet. For a more detailed description of the life cycle of a process fragment in the context of compliance please refer to [SLM+10].

For usage of a process fragment as a reliable implementation of a compliance requirement, it is desirable to have proven that the process fragment is actually implementing what it has been designed for. The approach in COMPAS to provide such a proof is to formally specify compliance requirements with the formal language developed in WP2 (cf. [D2.3]) which can be checked against a process with tools for verification which are developed in WP3 (cf. [D3.3]). However, the concept for process fragments that we present in this deliverable differs from the specification of process models defined in the BPEL specification [OASIS07]: A process fragment has special characteristics, such as broken links, i.e. control edges that have either no source or no target activity. We also distinguish between mandatory and optional entries into, and exits from a fragment. Also we propose an extension for constrained activity placeholders for supporting variability and increasing reusability.

We are currently investigating if the tools for verification (WP3) are able to check all fragments that we create, and thereby provide a proof that they actually implement the formally specified compliance requirement. There is the possibility that there exist different sub sets of process fragments, on the one side fragments which are verifiable, on the other hand fragments which cannot be verified or which can only be verified after modification. The classification and specification of reusable artefacts is future work in WP4. Thus this issue will be discussed in upcoming deliverable [D4.3]. It is important to mention that the verification can only work when all necessary information is provided. Otherwise verification will return a result like “impossible to check”, annotated with what is missing in the verification request.
With respect to the overall COMPAS architecture shown in Figure 1, we have the following procedure in mind: Imagine a process fragment (stored in a repository, cf. [D4.4]) that implements a compliance requirement by means of activities and control dependency among them. This process fragment can be integrated into an existing process with the intention of making the process compliant to the corresponding compliance requirement. Thus, after the fragment has been integrated into a process, the process should actually comply with the compliance requirement that the fragment implements. However, the process designer who has integrated the fragment into the process does not have a proof yet that it has been integrated in the correct manner and at the correct place. Therefore, formal rules (developed in WP2) that represent this compliance requirement in a formal manner can be checked against the modified process model using the verification tools developed in WP3 for assuring compliance. In summary, the approach and concepts presented in this deliverable enable the augmentation of existing processes with compliance requirements.

Figure 1 Overall COMPAS architecture
3. Extending BPEL

This section describes the ways and methodology of extending BPEL in general. We begin with a survey on prominent BPEL extensions, followed by a discussion of the formalities for defining of a standard-conform BPEL extension.

3.1. BPEL extension background

Along with the increasing adoption of BPEL for Web service orchestration and for definition of workflows, the demands on the functionality are increasing. Both, industry and academia are frequently developing extensions for supporting new functionality and concepts that are not available in standard BPEL. It depends on the particular purpose of the approach which characteristics of the language are extended. Some extensions intend an improvement of flexibility, while others may target at improving reusability. BPEL has been designed to be extensible (cf. Section 3.2), therefore the language can be extended easily.

The BPEL specification (cf. [OASIS07]) mentions a simple example for an extension. In this example, an additional identifier attribute is added to each element in the process. This identifier can be used in modelling for referencing constructs, but it does not need to be understood by the engine that executes the process, as there is no specific runtime behaviour associated with this extension. Currently, there are dozens of different extensions available in the products offered by vendors, as described for instance in [IBM09] or in [ORACLE07]. Also the open source offerings provide many different extensions of the BPEL language, such as [APACHE09]. In this sub section, we give an overview of the most prominent BPEL extensions.

Many business process scenarios require interactions with humans. The WS-BPEL Extension for People (BPEL4People) addresses this requirement and defines a set of extensions for supporting the integration of humans in a workflow (cf. [AAD+07a]). This includes the possibility to define people’s activities, people groups, tasks and notifications. In fact, BPEL4People is building on another extension which is called WS-HumanTask (cf. [AAD+07b]). WS-HumanTask is used in BPEL4People for the actual implementation of a people activity. BPEL4People has been submitted to OASIS for standardization. The standardization is important for enabling portability of process models among platforms of different vendors. Standardization is also necessary for assuring interoperability of the involved components (process engine, task manager, task list client).

The BPEL extension for sub-processes (BPEL-SPE, cf. [KKL+07]) is a BPEL extension that addresses reusability. A new kind of extension activity (call activity) can be used for invoking sub-processes. A sub-process can either be defined within a process (inline), or as a separate entity (standalone). An important difference compared to a standard service invocation is that a sub-process is executed in the context in which it is defined: an inline sub-process can access variables from the scope it is defined in. Furthermore, the relation between a sub-process (the callee) and parent process (the caller) extends the error handling and the compensation model of BPEL. For instance, a fault thrown in a sub-process needs to be propagated to the parent process. We have discussed this extension in more detail in [D4.1] in Section 2.2.13.

BPEL for Java (BPELI, [BGK+04]) is an extension that proposes a combination of BPEL and Java code. The intention is to provide a way for integrating small pieces of Java code into a BPEL process definition. The main effect of this extension is more convenience when programming a BPEL process. Another effect is the possibility to integrate small pieces of
functionality directly into the process without having to create a dedicated Web service for it. Therefore a new kind of extension activity (snippet) is discussed. The authors also discuss usage of Java expressions in loop conditions, variable initializations, etc. The specification introduces a few minor changes to BPEL as well as several extensions in order to fit the languages.

On the one hand the extensions above may provide additional functionality, better reusability of code or more convenience in process design. But on the other hand the extensions also have an important drawback, as they are limiting the portability of a process if they are used. In [IBM09] an advice is given, when to use, and when not to use the BPEL extensions that they provide in their product. It is stated that their extensions are a significant enhancement to the BPEL language, but that a user has to decide if they should be disabled. One reason for not using an extension is when the process should be usable or editable in another set of tools. Another reason is when it is planned to change to a different execution platform. It is also not recommended when information about process models are exchanged with business partners who are running a different set of tools. To sum this up, when extending the language you have to keep in mind that the whole toolset that is used for management of the processes is affected. This includes a process design tool and the process engine up to monitoring or mining tools. Furthermore, the ability to exchange process information among different BPM toolsets might get lost as they all have to be able to handle the extension. This results in a loss of portability of a process and in a decrease of the overall flexibility, respectively.

An extension typically (but not necessarily) impacts both, design time and runtime. In [WGS+09] we showed that a design time only extension may be quite useful as well. A design time only extension is a language extension that makes use of a model transformation approach or that invokes dedicated services in order to integrate additional features. In [WGS+09] we have proposed an extension to BPEL for allowing usage of pointers to variables, as in standard BPEL only call-by-value semantics are supported. For this extension a model transformation approach is used. This allows introducing higher level constructs at design time which are then transformed into standard BPEL constructs before execution. This is, however, only possible, when an extension is expressible with a set of standard constructs. The transformation allows running processes that use such language extension in a normal BPEL process engine.

3.2. BPEL extension formality

The BPEL 2.0 specification describes the formalities for extending the language in a standard-conform manner (cf. [OASIS07], Section 14). The extension possibilities range from new XML attributes and XML elements to entirely new activities. However, an extension must not contradict or change the semantics defined in the specification, in order to be standard-conform. Then again, also invalid extensions may have their eligibility. If an extension is not valid according to the specification it does not imply that it is not useful. In the following we describe and exemplify the possibilities for extending BPEL which are defined in the specification.

3.2.1. Extension declaration

An extension has to declare its own namespace which may be used multiple times, though. By setting the attribute mustUnderstand the extension has to indicate whether it is mandatory that it is understood by a process engine or whether it is optional, i.e., it may be ignored. The
extension declarations are listed inside the `<extensions>` element which is a child element of the `<process>` element. Listing 1 shows a declaration of an extension.

```
<bpws:process name="loanApprovalProcess"
  targetNamespace="http://example.org/loanApproval"
  xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:com="http://www.iaas.uni-stuttgart.de/compliance">
  <bpws:extensions>
    <bpws:extension namespace="http://www.iaas.uni-stuttgart.de/compliance"
      mustUnderstand="yes" />
  </bpws:extensions>
  ...
</bpws:process>
```

**Listing 1  BPEL extensions declaration example**

### 3.2.2. Extension attributes

Adding an additional attribute is one of the possibilities for extending the language. For instance, it is often desirable to add a unique identifier to each construct. By declaring an according extension this can be done in standard-conform manner. Listing 2 shows the usage of an extension attribute for integration of unique identifiers.

```
<invoke partnerLink="loanService"
  operation="check"
  inputVariable="loanApprovalInput"
  com:uniqueIdentifier="808f5db0add3b29d3" />
```

**Listing 2  Extension attribute example**

### 3.2.3. Extension elements

Another possibility for extending the language is the usage of extension elements. This is preferable when the information is not directly related to one single construct, and also if the amount of additional information is higher than a single value. For instance, the BPEL extension WS-HumanTask [AAD+07b] uses this extension mechanism for the declaration of human interactions and people groups for supporting staff query functionality. Listing 3 shows an example for this.

```
<process ...
  xmlns:htd="http://www.example.org/WS-HT">
  <extensions>
    <extension namespace="http://www.example.org/WS-HT"
      mustUnderstand="yes"/>
  </extensions>
  <htd:humanInteractions>
    <htd:logicalPeopleGroups>
      <htd:logicalPeopleGroup name="approvers">
        <htd:documentation> The group entitled to approve a loan request. </htd:documentation>
      </htd:logicalPeopleGroup>
    </htd:logicalPeopleGroups>
  </htd:humanInteractions>
</process>
```

---

File: D4.2_BPEL-extensions-for-compliant-services.doc  Page 12 of 47
3.2.4. Extension assign operations

The `<assign>` activity can be used for assignments of variables and for evaluation of expression. The language specification provides a mechanism for extending these data manipulation operations. However, the specification does neither provide a detailed explanation, nor an example or the intention of this extension. The authors of [BGK+04] propose another mechanism for advanced assign operations which is to use Java statements within a BPEL process (BPELJ).

3.2.5. Extension activities

The BPEL specification also allows introducing new activity types. Since version 2.0 new activities have to be placed inside a `<extensionActivity>` element, in former versions this was not required. The `<extensionActivity>` element may only contain one child which is the new activity (which may be a structured activity, though). Listing 4 shows an example for an extension activity which creates a trusted timestamp for auditing reasons.

```xml
<extensionActivity>
  <com:trustedTimestamp name="processStart"
    outputVariable="timestamp">
  </extensionActivity>
```

Listing 4  Extension activity example
4. Process fragments for compliance

In this section, the concept of process fragments for compliance is elaborated on a high level of abstraction. We discuss the requirements and constraints which we want to address with our concept and we also motivate the extensions for compliance which we propose. The content of this section is partially based on our recent publication on the integration of compliance into business processes using the notion of process fragments. For more details please refer to [SLM+10]. The concepts presented in this section are not only applicable to BPEL, they can also be taken as a basis for formulating the concept of process fragments for compliance in other graph-based process languages, such as BPMN [OMG08]. The technical details and the actual implementation of the BPEL extensions are discussed afterwards, in Section 5.

4.1. Towards reusable implementations of compliance requirements

We have identified two distinct groups of compliance requirements, the one specifying what should be done, the other specifying how things should be done [SLM+10]. The first group requires the integration of additional activities or changes to a given process model. We address this group of requirements with the concept of process fragments. The second group necessitates the employment of annotations that state how things should be executed. An annotation can basically be stated in any language that is applicable for the domain they are stemming from (domain-specific language, DSL). For instance, requirements related to security can be stated in the security DSL which is developed in WP5. Of course, the monitoring infrastructure has to provide the capability for checking these requirements accordingly. As we mentioned before, process fragments for compliance can be seen as a DSL as well, as its meta model defines a language specific for the domain of BPEL process structures for compliance. Although we focus on process fragments in BPEL, we first introduce a technology and implementation independent meta model for process fragments in Section 5.1.1 and show its application to BPEL afterwards in Section 5.1.2.

Our investigation has shown that a frequent compliance requirement of the first group is to check a particular situation. Typically this requirement occurs even multiple times within a process, e.g. in four eyes principle situations or in approval chains typically a check is performed. This assumption is also confirmed by the compliance requirements of the THALES case study (cf. [D6.1]):

1. If a credit exceeds 1 million euros, the post-processing supervisor must check whether the operation is profitable
2. Final approval has always to be done by a manager

All the mentioned situations are demanding for the same steps to be executed, though with different parameters of course. We can now abstract from the specifics of those situations and thereby create a process fragment for assessing a situation and deciding between a positive and negative outcome (see Figure 2). Such a process fragment is a reusable building block for process modelling which enables a faster and more consistent integration of the compliance requirement that it implements, i.e. in this case an approval.
In Figure 2 this abstract, reusable fragment is shown. Note, that this is not meant to be a definitive graphical notation for process fragments. It is used here informally as an aid to ease understanding. In terms of control flow the fragment states that if a certain activation condition is met, a particular situation is checked and assessed. The fragment has one entry, and two exits (accept, reject). Basically, different parameters can be externalized from the fragment to further increase the reusability of the fragment:

- An activation condition which states, in which cases the check should be performed
- The person (or service) who should assess the situation can be parameterized by a role-based or person-based staff query or by a service endpoint reference
- The particular situation that should be checked can be externally described by variables that are passed to the approval activity (i.e. to “Perform check”)

![BP Logic Fragment for approval](image)

**Figure 2  BP Logic Fragment for approval**

After a process fragment has been integrated into a process, the process should actually comply with the according compliance requirement. However, the process designer who has integrated the fragment into the process does not have a proof yet that it has been integrated in the correct manner and at the correct place. As we mentioned in Section 2, formal rules (stated in the compliance rule language that is developed in WP2, cf. [D2.3]) that represent this compliance requirement can be checked against the modified process using the verification tools developed in WP3. Please note, the verification of this fragment using the WP3 tool is discussed in [D3.2].

### 4.2. Analysis of concepts for reusability and compliance

Now that we have demonstrated the basic idea of the concept of process fragments as reusable implementations of compliance requirements, we can discuss current concepts for reusability and compliance, before sketching a more fine-grained concept.

Our former analysis on approaches for improving reusability in processes and service compositions has lead to the conclusion that each of the discussed approaches has certain
advantages, but also drawbacks in its application in practice (cf. [D4.1]). For instance, we have discussed the concept of sub-processes which allow reuse on the level of application and business logic, yet we think this concept is rather monolithic (a sub-process has a single initiating receive activity) and therefore not flexible enough. Hence we want to create a flexible concept for reuse of business logic.

Moreover we have learned in our analysis that several concepts used in the analysed approaches are also quite useful for tackling compliance and can be seen as a source of inspiration for further requirements (cf. [D4.1], Section 4.2). For making this deliverable more self-contained we repeat these concepts at this place:

i. Parameterization: it can help in achieving compliance checking by making use of substitution mechanism.
ii. Constraint based approach: it can be used to specify parts of policies and parts of regulations for example. In this case, the compliance checking problem can be seen as a constraint satisfaction problem.
iii. Declarative approach: it enables designers to describe the actions that a business process needs to contain, but not their sequence. As a consequence, it becomes easy to perform compliance checking by reasoning on high-level description of policies rather than on codes associated to policies.
iv. Business rules: they constitute the main component in a business application. Compliance to requirements is the main concern of business rules. The way business rules are extracted and represented will have a significant impact on compliance checking.
v. Annotation: it can be considered as a means to relate artefacts. Annotations can play an important role in checking compliance by discovering possible mappings (through annotations) between considered components. This is in the line of semantic-based approaches to compliance checking.
vi. Transformation: it happens that components involved in compliance checking are described in different languages. Transformation can be considered as a pre-processing step; checking will be performed on statements specified in a common language.
vii. Patterns: they present solutions for recurring requirements in a specific context. Patterns allow bridging the gap between experts and developers. Since patterns are used to capture solutions at a high-level of abstraction, they can be used in the process of compliance checking for those issues that require abstraction.

4.3. Mapping of compliance concepts to process fragments

In the following we analyse the concepts listed Section 4.2 and map them to concepts in a process model, for achieving a concept for process fragments for compliance.

Achieving compliance requires some kind of constancy and adherence to more or less strict rules. However, with process fragments we want to create a lightweight and flexible approach for reuse of process code, similar to patterns (vii.). To solve this contradiction we can make use of the concept of parameterization (i.). By parameterizing parts of a fragment we can combine flexibility with strictness. All parts (in terms of variables etc.) which may be changed are explicitly declared as parameterizable. All other parts are taken as strict and unchangeable. If those strict parts are changed during integration into a process anyhow, it cannot be guaranteed that the process fragment is still implementing the compliance requirement that it has been designed for. In this case, the compliance fragment is broken.
Basically, annotations (v.) can be used as concept for realizing the parameterization. For instance, an annotation can state a constraint which service may be used for a particular activity, or which static value may be changed. We are aware that process fragments are not capable of realizing all compliance requirements concerning business processes. Process fragments can be used to specify or constrain control flow and data flow within a process, however we cannot use them to ensure compliance of the services involved in the process, and the people respectively. For instance, a compliance requirement that demands password-protected storage of customer billing data for ten years cannot be realized by a process fragment, as this requirement is related to a database that is external to the process engine that executes the fragment. In COMPAS we propose (textual) annotations (v.) using domain-specific languages and the corresponding code generation and model transformation (vi.) for implementing compliance requirements that are not expressible as process fragments.

The concepts of constraint based approaches (ii.) and declarative approaches (iii.) are indeed very powerful concepts. Constraints can be used to specify obligations and permissions. As long as the specified constraints are satisfied, compliance to particular requirements can be assured. Also the declarative approach enables a designer to describe the actions that have to occur. There are already several approaches that map these concepts to processes in general (cf. [D4.1]). We want to transfer the basic idea to the concept of process fragments. Typically, a process model strictly specifies the execution order of the contained activities. We can relax this strictness by introducing placeholders for declaring regions in which the execution order is not strictly specified. However, our intention is not to reinvent case handling (e.g., as discussed in [VA97]), but to create an approach that targets compliance and that enables compliant composition. Thus, for these regions we envision the possibility to annotate constraints (v.) that specify the characteristics of what may happen (positive), what must happen (obligatory) and what is permitted (negative). The intention of a constrained region is that only constructs that satisfy certain constraints may be inserted. This constrained region is aimed at use in modelling, but could basically also be used in execution for dynamic checking, if the constraints are satisfied.

Concerning flexibility, we want our approach to overcome the limitations of sub-processes that we have mentioned in the previous sub section. We want to allow multiple entry and exit control edges, without declaring one of them to explicitly instantiate the fragment. However, we have to find a way to define composition constraints for specifying correct usage of the fragment. For instance, the fragment for approval that we showed in the beginning of this section (cf. Figure 2) has one entry for starting the check, and two exits after the check has been performed. One exit is for the approval, the other for rejection. All entries and exits of the fragment have to be wired into a process for correct execution, or composed with other fragments, respectively. However, it is also conceivable to define entries and exits of a fragment which do not have to be wired for correct execution, meaning they are optional. Hence, by distinguishing between mandatory and optional entries and exits we are able to express how the fragment has to be wired and how it may be wired. Furthermore, a fragment may only be wired using the predefined entries and exist, because adding entries or exits which are not predefined in the fragment might break its design purpose, for instance the approval could be bypassed.

As for business rules (iv.), approaches are already in place (e.g., [RD05]) and can be applied to processes as well as to process fragments in the same way. In this sense, we see business rules as possible extension to the concept of process fragments, but they are not our focus. In fact, in [WGS+09] we showed that it is possible to integrate business constants for compliance into a BPEL process without even requiring an extension of the language, using transformation techniques (vi.). For instance, this allows us to extract the threshold amount in
a loan approval process that states, when a manager also has to check the credit worthiness (in addition to a clerk).

From [D2.1] we have learned that there are various kinds of languages for compliance and for expressing constraints, respectively. Furthermore, various domain-specific languages are available for expressing very particular kinds of requirements. Also general purpose languages, such as the Object Constraint Language (OCL) could be used for stating how a region may be replaced. For the time being we have not committed to one (or multiple) particular constraint expression language.

### 4.4. Concept for process fragments for compliance

In deliverable [D4.1] we have defined a process fragment as a “connected sub-graph of a process graph. Applied to BPEL [OASIS07], it can also contain additional artefacts, such as variables, references to related Web Services orchestrated within the process, annotations (which are very important for stating compliance) etc. Some parts of a fragment may be subject to parameterization, or explicitly stated as opaque, in order to mark points of variability and degree of freedom for reuse. Thus, in contrast to a sub-process, a process fragment is not necessarily directly executable.”

Based on our research on process fragments we presented in [SLM+10] a refined and more general definition of a process fragment which we want to adopt for COMPAS:

“A process fragment in our work is defined as a connected graph, however with significantly relaxed completeness and consistency criteria compared to an executable process graph. A process fragment is made up of activities, activity placeholders (so-called regions) and control edges that define control dependency among them. A process fragment may define a context (e.g., variables) and it may contain a process start or process end node, but it is not required to do so. It may contain multiple entering and leaving control edges for integration into a process or with other process fragments. A process fragment has to consist of at least one activity and there must be a way to complete it to an executable process graph.”

Compared to our prior definition we have relaxed the requirement that a process fragment has to be a sub-graph of a process graph. This requirement imposed that an executable process existed in which the fragment was contained. From our current understanding it is also possible to create a fragment from scratch. In this case, there is no need for a process containing the fragment. For avoiding the specification of fragments which make no sense we have added the requirement that there must be a way to complete it to an executable process graph, though. So still, a process fragment is not necessarily directly executable and it may be partially undefined. We have removed the part about parameterization from our prior definition, as parameterization is special requirement for compliance and therefore not required in a general definition for process fragments. Concerning the naming we have changed “opaque activities” (based on the BPEL specification) into “regions” for describing placeholders. The refined definition is not specific for BPEL and can basically be applied to other process languages that can be represented by a process graph, as well. For instance it can also be applied to the Business Process Modeling Notation (BPMN, [OMG08]).

---

1 A process graphs describes a model for a process, where nodes represent activities of a process and edges represent control dependencies between them.
Based on the solution concepts discussed in the previous sub section, we have furthermore, proposed particular additions to our refined definition. For usage of process fragments in the field of compliance we propose in [SLM+10] the following additional characteristics:

i. A process fragment may be parameterizable in order to mark points of variability. The parts (variables etc.) of a process fragment which may be changed can be explicitly declared. This way it can be better ensured that, after integration into a process, a process fragment still implements the compliance requirement that it has been designed for. For some cases it may also be conceivable to limit the range of valid values for the parameters.

ii. The placeholders contained in a process fragment (i.e. regions) may be constrainable. By constraining the regions it can be stated how those placeholders may be filled. Just like the concept of parameterization this concept allows us to ensure more stability of the fragment. However, an important difference to parameterization is that constrained regions can be used to specify how a fragment may be composed. This influences flexibility on the one hand and ensures on the other hand that the design intention of the fragment does not get broken.

iii. Optional and mandatory entries and exits of a process fragment should be distinguishable. We believe it is of fundamental importance to declare how the fragment has to be wired within a process for ensuring compliant behaviour. Mandatory entries and exits have to be wired for making the fragment function properly, optional ones can be neglected.

For the aid of understanding we provide a graphical example for a process fragment for compliance in Figure 3. Please note that it is not meant to be a definitive graphical notation for process fragments for compliance.

![Refined BP Logic Fragment for approval](image-url)
5. BPEL extensions for compliant services

Based on the refined concept for process fragments introduced in the previous section, we discuss in Section 5.1 the meta model for process fragments for compliance and we discuss how this meta model maps to BPEL. For enabling the infrastructure to manage and handle the process fragments we also provide an XML schema and an XML example for the BPEL extensions for process fragments. Subsequently we discuss in Section 5.2 the extensions to BPEL for traceability. The XML schema files and XML examples provided in this Section enable the creation of process fragments using state of the art tools for editing XML documents (e.g., the free XML Editor in Eclipse). Furthermore, they allow the implementation of tools for management and handling of process fragments and thus make up the actual prototype.

5.1. BPEL extension for process fragments

5.1.1. Process fragment meta model

In this section, we introduce a meta model for process fragments that is independent from platform and technology for the realisation and implementation of process fragments. Figure 4 shows the process fragment meta model as entity-relationship diagram (ERD). An ERD consists of entity sets, specifying the different concepts contained, and the relationships between the entity sets defining how the concepts are associated and related to each other. In Table 1 the entity sets used within the ERD, cf. Figure 4, are listed and described.

First of all we would like to point out that for modelling the ERD we used the notation introduced by Peter Pin-Shan Chen [Che76]. This notation allows the distinction of the types of participation (total or partial) of entities of the corresponding entity sets in a relationship. The total participation is represented by a perpendicular short stripe at the end of the relationship where the corresponding entity set is located. This means that every entity belonging to that entity set has to participate in the relationship. The partial participation is represented by a small circle located at the end of the relationship where the entity set resides. This means that not every entity belonging to that entity set has to participate in the relationship. Concerning the cardinality please note that the “local” end of the line between a relationship set and an entity set specifies the cardinality constraints for a corresponding entity set.

Taking a look at Figure 4, the two different generalizations represented in bold draw the attention of the reader. The total and disjoint generalization between the entities Link Type, Control Link Type and Data Link Type specify that a higher-level entity of entity set Link Type must belong either to the lower-level entity set of Control Link Type or to the lower-level entity set Data Link Type. However the partial and disjoint generalization between the entities Activity Type and Region Type define that a higher-level entity of entity set Activity Type need not belong to the lower-level entity set Region Type.
Figure 4  Process fragment meta model as entity-relationship diagram
Figure 5  WS-BPEL 2.0 meta model [Dub07]

Disclosure: This meta model, shown in Figure 5, was derived from OASIS documents but is in no way associated to a OASIS recommendation nor it has been reviewed by OASIS members. It is provided as-is with no warranty of suitability or correspondence to OASIS recommendations. It is only provided to facilitate the understanding of OASIS specifications. It was created from the OASIS document available online at [http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html](http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html) and licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License [CC09].
<table>
<thead>
<tr>
<th>Entity Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entry Point Type</strong></td>
<td>An entity <strong>Entry Point Type</strong> represents a concept for compositability of process fragments. When the process designer would like to glue a process fragment into an already existing process or while creating a new process by composing different process fragment the knowledge of connecting process fragment external and internal control and data flow is required. Thus entry points are used to define and specify where and how control and data flow from outside the process fragment should be connected with the inside to preserve the compliance requirement implemented by the BPLF.</td>
</tr>
<tr>
<td><strong>Exit Point Type</strong></td>
<td>On the contrary to the description of the entity set <strong>Entry Point Type</strong> an entity <strong>Exit Point Type</strong> is utilized to specify where and how control and data flow inside the process fragment has to be connected with the outside.</td>
</tr>
<tr>
<td><strong>Activity Type</strong></td>
<td>The entity set <strong>Activity Type</strong> represents all types of activities and tasks that can be contained within a process model. This includes unstructured as well as structured types of activities and tasks. Please note the partial and disjoint generalization between the entity sets <strong>Activity Type</strong> and <strong>Region Type</strong>. This means that an entity belonging to the entity set <strong>Activity Type</strong> does not need to belong to the entity set <strong>Region Type</strong>.</td>
</tr>
<tr>
<td><strong>Region Type</strong></td>
<td>The entity set <strong>Region Type</strong> is required for enabling the definition of degrees of freedom for specified parts within a process fragment. There is the possibility to limit the degrees of freedom by applying constraints, cf. relationship set CT-constrains-RT. Due to the partial and disjoint generalization the lower-level entity set <strong>Region Type</strong> inherits the attribute name as well as all relationship participations of <strong>Activity Type</strong>.</td>
</tr>
<tr>
<td><strong>Gateway Type</strong></td>
<td>An entity belonging to entity set <strong>Gateway Type</strong> incorporates the concepts required for branching of control flow as well as joining of different branches concerning control flow. This includes exclusive, inclusive and parallel branching and loops.</td>
</tr>
<tr>
<td><strong>Link Type</strong></td>
<td>Due to the total disjoint generalization between <strong>Link Type</strong>, <strong>Control Link Type</strong> and <strong>Data Link Type</strong> this entity set is used as a higher level entity set and each entity within this entity set must either belong to <strong>Control Link Type</strong> or <strong>Data Link Type</strong>. The lower level entity sets inherit the attribute name and all relationship participations of <strong>Link Type</strong>.</td>
</tr>
<tr>
<td><strong>Control Link Type</strong></td>
<td>The entity set <strong>Control Link Type</strong> represent the concept of specifying the control flow within a process fragment by connecting entities belonging to entity sets of type <strong>Activity Type</strong>, <strong>Region Type</strong>, <strong>Gateway Type</strong>, <strong>Exit Point Type</strong> or <strong>Entry Point Type</strong>. Please take a look at the corresponding relationship sets in Table 2.</td>
</tr>
<tr>
<td><strong>Data Link Type</strong></td>
<td>This entity set incorporates the concept of defining data flow within a process fragment by connecting entities belonging to the entity sets <strong>Activity Type</strong>, <strong>Region Type</strong> and <strong>Data Container Type</strong>. Please see the corresponding rows for relationship sets in Table 2.</td>
</tr>
<tr>
<td><strong>Constraint Type</strong></td>
<td>The entity set <strong>Constraint Type</strong> enables the specification of constraints on other entities belonging to the entity sets utilized within a process fragment. For details please take a look at Table 2 and the corresponding rows therein.</td>
</tr>
</tbody>
</table>
Condition Type

Entities that belong to the entity set Condition Type are representing the concept of defining conditions regarding data flow or branches within the data flow. Please see Table 2 for details concerning the definition of conditions.

Parameter Type

The main goal of COMPAS is to enable the augmentation of processes and service compositions with compliance requirements while considering reusability. One possibility to enable reusability is parameterization. The entity set Parameter Type introduces and applies parameterization to the approach of process fragments.

Data Container Type

We take explicit handling of data into consideration and therefore we include a concept to buffer and store data. Thus we added the entity set Data Container Type for this purpose.

Table 1  Entity sets of process fragment meta model

In Table 2 the relationship sets used within the ERD (cf. Figure 4) are listed and the association of the concepts of the process fragment meta model via these relationship sets is explained.

<table>
<thead>
<tr>
<th>Relationship Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint Type constrains Exit Point Type (CT-constrains-ExPT)</td>
<td>This relationship set enables the specification of zero or more constraints for each exit point within a process fragment. Please note that all entities belonging to entity set Exit Point Type have to participate in this relationship. However not all entities belonging to entity set Constraint Type have to participate in this relationship, because in a process fragment model the specification of constraints is also possible for regions, entry points, etc. Please see the following row as well as the other rows regarding the corresponding relationship sets in this table.</td>
</tr>
<tr>
<td>Constraint Type constrains Entry Point Type (CT-constrains-EnPT)</td>
<td>CT-constrains-EnPT relationship set allows the definition of zero or more constraints regarding each entry point within a process fragment. The entities of the entity set Constraint Type do not all have to participate in this relationship. On the contrary all entities belonging to the entity set Entry Point Type have to participate in this relationship.</td>
</tr>
<tr>
<td>Control Link Type connects Entry Point Type and Activity Type (CLT-connects-EnPT-and-AT)</td>
<td>This relationship set enables the specification of control flow between one entry point and one activity, or one entry point and one region by using one control link within a process fragment. Please note that all entities which belong to the entity set Entry Point Type, but not all entities belonging to the entity sets Control Link Type, Activity Type and Region Type have to participate in this relationship.</td>
</tr>
<tr>
<td>Control Link Type connects Exit Point Type and Activity Type (CLT-connects-ExPT-and-AT)</td>
<td>This relationship set allows the specification of control flow between one exit point and one activity or one exit point and one region by using one control link within a process</td>
</tr>
<tr>
<td>Constraint Type constrains Region Type (CT-constrains-RT)</td>
<td>CT-constrains-RT relationship set allows the definition of zero or more constraints regarding each region within a model of a process fragment. Please note that all entities belonging to the entity set Region Type, but not all entities of the entity set Constraint Type have to participate in this relationship.</td>
</tr>
<tr>
<td>Control Link Type connects Activity Type and Gateway Type (CLT-connects-AT-and-GT)</td>
<td>The relationship set CLT-connects-AT-and-GT enables the specification of control flow within the model of a process fragment, each between one activity or region and one gateway by using a control link. Please keep in mind that all entities belonging to the entity set Gateway Type, but not all entities belonging to the entity sets Activity Type, Region Type or Control Link Type have to participate in this relationship.</td>
</tr>
<tr>
<td>Control Link Type connects Activity Types (CLT-connects-AT)</td>
<td>The CLT-connects-AT relationship set allows the definition of control flow within a model of a process fragment. Control flow can either be defined between two activities or two regions or one activity and one region by using one control connector. Please note that neither all entities belonging to entity set Control Link Type nor all entities belonging to entity sets Activity Type or Region Type have to participate in this relationship.</td>
</tr>
<tr>
<td>Gateway Type has Condition Type (GT-has-CdT)</td>
<td>This relationship set enables the definition of conditions for gateways, meaning one condition for each gateway within the model of a process fragment. All entities belonging to the entity set Gateway Type, but not all entities belonging to the entity set Condition Type have to participate in this relationship. For each gateway used within a process fragment a condition has to be specified which forms the basis for the corresponding branching or joining of the control flow.</td>
</tr>
<tr>
<td>Constraint Type constrains Parameter Type (CT-constrains-PT)</td>
<td>The relationship set CT-constrains-PT allows the definition of constraints on parameters within a model of a process fragment. In this context it should be noted that for each parameter one or more constraints can be specified. This can for instance be used to specify the range for a specific parameter used for a process fragment. Please consider that all entities belonging to the entity set Parameter Type, but not all entities belonging to entity set Constraint Type have to participate in this relationship.</td>
</tr>
<tr>
<td>Relationship Set Description</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Control Link Type has Condition Type (CLT-has-CdT)</td>
<td>CLT-has-CdT relationship set enables the definition of zero or one condition for each control link within a model of a process fragment. Please note that neither all entities of the entity set Control Link Type nor all entities belonging to the entity set Condition Type have to participate in this relation.</td>
</tr>
<tr>
<td>Parameter Type is used in Condition Type (PT-is-used-in-CdT)</td>
<td>For the purpose of increasing reusability of process fragments this relationship set allows using one or more parameters for defining each condition within a model of a process fragment. Please take into consideration that neither all entities belonging to the entity set Parameter Type nor all entities belonging to the entity set Condition Type have to participate in this relationship.</td>
</tr>
<tr>
<td>Constraint Type constrains Link Type (CT-constrains-LT)</td>
<td>Additionally constraints within a process fragment model can also be specified for control links and data links. Precisely stated for each control link or data link within a process fragment model one or more constraints can optionally be defined. Please note that neither all entities belonging to the entity set Constraint Type nor all entities belonging to the entity set Link Type have to participate in this relationship.</td>
</tr>
<tr>
<td>Data Link Type connects Data Container Type (DLT-connects-DCT)</td>
<td>This relationship set allows the definition of data flow between data containers by connecting two data containers via one data link. Please consider that neither all entities of entity set Data Link Type nor all entities of entity set Data Container Type have to participate in this relationship. This relationship set is required due to the demand for technology and implementation independent process fragment model. The explicit specification of data flow allows covering both, explicit and implicit data flow in a process.</td>
</tr>
<tr>
<td>Data Link Type connects Data Container Type and Activity Type</td>
<td>Due to the handling of the data flow it is required to enable the definition of data flow between data containers and activities within a process fragment model. Thus, this relationship set facilitates the specification of data flow within a process fragment model either between one activity and a data container or between one region and a data container by using a data link. Please note that neither all entities of entity sets Data Container Type, Activity Type and Region Type nor all entities of entity set Data Link Type have to participate in this relationship.</td>
</tr>
</tbody>
</table>

Table 2  Relationship sets of process fragment meta model
5.1.2. Process fragment meta model for BPEL

In this section we apply the technology and implementation independent meta model for process fragments described in Section 5.1.1 to BPEL. Figure 5 shows the meta model of BPEL version 2.0 [Dub07] without the extensions required for the specification of process fragments in BPEL. During the process of gluing process fragments into an existing process, or when composing them, the concepts included in the process fragment meta model for BPEL which are not part of the BPEL standard specification [OASIS07] have to be transformed into standard BPEL code.

Table 3 (see below) gives an overview of the relation and mapping of concepts from the technology independent meta model to BPEL. The main purpose of the description within Table 3 is to specify which artefacts and concepts of the technology independent meta model for process fragments can be directly mapped to already existing concepts and artefacts contained in the BPEL meta model and for which artefacts an extension of the BPEL meta model is required. Please see the column description in Table 1 for the detailed explanation of the semantic and usage of the entity sets discussed below.

<table>
<thead>
<tr>
<th>Entity Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entry Point Type</strong></td>
<td>The concept of defining entry points within a process fragment model has to be covered by an extension, because such a concept is not covered by the BPEL specification [OASIS07]. Please note that the definition of one or more entry points within a process fragment in BPEL is not mandatory, because for example a process fragment can also be defined as the reusable first part of a business process and therefore contain a process start node, i.e. BPEL activities &lt;receive&gt; and &lt;pick&gt; with createInstance attribute set to yes.</td>
</tr>
<tr>
<td><strong>Exit Point Type</strong></td>
<td>The current version of the BPEL specification does not contain a concept for defining exit. Thus this concept also has to be covered by the BPEL extensions for compliant services. Please take into consideration that the modelling of exit points within process fragments in BPEL is not mandatory, because a process fragment can for example also be used to describe the last part of a business process.</td>
</tr>
<tr>
<td><strong>Activity Type</strong></td>
<td>The entity set Activity Type is represented in the BPEL meta model by the concepts for the structured activities namely Flow and Sequence as well as for basic activities namely Invoke, Receive, Reply, Pick, Empty, Exit, and Wait, cf. Figure 5.</td>
</tr>
<tr>
<td><strong>Region Type</strong></td>
<td>A concept for defining parts of a business process as undefined and defining degrees of freedom is not included in the current version of BPEL [OASIS07], but it is required within the meta model for process fragments in BPEL. So such a concept with the semantic described in Table 1 has to be introduced with the BPEL extension for compliant services. Note, a concept similar to a region is the opaque activity which is defined in Abstract BPEL. However, this concept is more related to information hiding and in a way different to regions.</td>
</tr>
<tr>
<td><strong>Gateway Type</strong></td>
<td>As described in Table 1 the semantics of the gateway concept is branching and joining of control flow including loops, etc. Taking a look on both Figure 4 and Figure 5 and comparing them the first thing you will notice is that in the technology independent meta model this concept is represented through the entity set Gateway Type. On the contrary, in the class diagram of the meta model for BPEL the class Activity is the super class for classes representing structured activities and basic activities as well as classes representing branching</td>
</tr>
</tbody>
</table>
and loops. We decided to follow a different approach in our technology independent meta model for process fragments in order not to limit the mapping only to BPEL, but also enable mapping to other languages in the area of Business Process Management, e.g., BPMN. Nevertheless, the concept for branching and joining control flow including loops is already contained in the BPEL specification by, \textit{If}, \textit{ElseIf}, \textit{RepeatUntil}, \textit{ForEach} and \textit{While}, cf. Figure 5.

\begin{tabular}{|l|p{15cm}|}
\hline
\textbf{Link Type} & As BPEL only allows the explicit definition of control flow but not data flow, there is no need for a concept only used as high-level entity set in BPEL like we use it in the technology independent meta model for process fragments, cf. Figure 4. \\
\hline
\textbf{Control Link Type} & Due to the fact that the BPEL specification does not cover the graphical representation of the BPEL artefacts there is no such concept contained in the meta model of BPEL. The explicit modelling of control flow is enabled by the concepts \textit{link}, \textit{target} and \textit{source}, cf. Figure 5. So there is no extension required in this respect, because the existing concepts in BPEL suffice. \\
\hline
\textbf{Data Link Type} & BPEL only allows the implicit specification of data flow realised by the concepts \textit{variable} and the activity type \textit{assign}. These concepts suffice for our needs regarding the specification of process fragments in BPEL. \\
\hline
\textbf{Constraint Type} & The entity set \textit{Constraint Type} enables the specification of constraints on other entities belonging to the entity sets utilized within a process fragment. For details please see the corresponding rows in Table 2. \\
\hline
\textbf{Condition Type} & The current version of the BPEL specification already allows the specification of terms by the concepts \textit{Expression} and \textit{Condition}. Thus an extension is not required. \\
\hline
\textbf{Parameter Type} & Due to the fact that the current version of the BPEL specification does not enable the usage of parameters the introduction of the parameter concept is necessary. \\
\hline
\textbf{Data Container Type} & The concept of a data container is realised within the BPEL meta model by \textit{variable}, cf. Figure 5. Please note that BPEL just allows the implicit definition of data flow by using the concepts \textit{variable} and \textit{assign}. Nevertheless this functionality suffices for the needs when specifying process fragments for compliance in BPEL. \\
\hline
\end{tabular}

\begin{table}[h]
\centering
\caption{Mapping of technology independent meta model to BPEL}
\end{table}

\subsection{5.1.3. Process fragment XML schema}

In this section, we propose a way for serializing process fragments for compliance as an extension to BPEL. We show an according XML schema (cf. Listing 5 and Listing 6) and we also provide an example in XML code (cf. Listing 7). Yet, before showing the XML schema we want to introduce the extensions that we propose to the BPEL language. We have taken care to (i) adhere to the language extension formalities discussed in Section 3.2 and (ii) not to change the semantics of standard BPEL constructs and in order to create (iii) a standard-conform BPEL extension after all.

All extensions we propose are design time language extensions, without exception. When integrating a process fragment into a process, the fragment has to be tailored to the specific needs of the context in which it is used. For integration we have a straightforward approach in mind which is called gluing [SLM+10]. Gluing means that the process fragments are physically copied and integrated into the process. For traceability and also for maintenance
reasons we can use unique identifiers which allow distinguishing between the original parts of the process and the integrated compliance parts. Such identifiers can be used when specifying a mapping between formal rules from WP2 and process fragment elements, by the way. Mandatory entries and exits have to be wired with the process or composed with other fragments for achieving an overall complete and executable process model, and thereby get removed. During the task of integration the placeholders are replaced with real functionality, i.e. the parameters are replaced with specific values, and regions are filled with activities or other fragments. The verification that constraints on regions and parameters are satisfied, depends on the language that is used for specifying constraints, however this does not require an extension to BPEL. The only extensions that remain after composition are the identifiers on the constructs for monitoring and traceability. In the following the extensions are described:

**ext:id**:
Annotations can be either made inline (for an example see Section 5.2), or by referencing. For easy reference from the outside a construct must have a unique identifier, preferably a universally unique identifier (UUID, cf. [LMS05]). However, using the name of a construct as identifier causes problems as the name might be changed often which requires all references to be updated accordingly. The usage of XPath and inline annotations is another possibility which is well applicable when it is integrated into the process during code generation (cf. Section 5.2), but it is not feasible for process modelling. The problem of applying this approach during design time is that all XPath expressions have to be synchronized with the process model each time the process is changed. In order to keep the fragment clean from many constraints and parameters we propose the extension of all BPEL elements with an identifier attribute for reference from the outside. An annotation mechanism that references this identifier allows the specification of constraints on regions and parameters on BPEL constructs from the outside. In XML serialization of the process fragment and also in its meta model in XML schema the constraints on regions and specified parameters are not visible, as they are specified external to the process fragment. Keeping this information outside the process enables a looser coupling between the fragment and the constraints, and thereby increases reusability. For the identifier extension we have chosen a different namespace (with prefix ext) than for the other extensions, as this extension can be ignored by an execution engine, i.e. mustUnderstand="no" (cf. Section 3.2).

**frg:fragmentRegion**:
The other extensions (regions etc.) have to be replaced before execution. By using mustUnderstand="yes" for the other namespace (with prefix frg) an engine will reject running a process in which not all placeholders are replaced by standard BPEL constructs. A fragmentRegion is a construct that needs to be replaced with (connected) standard BPEL structures. This means a region can also be used for composition of process fragments. The constraints that are imposed on a region are not within scope of the extensions to BPEL and thus we do not specify which languages have to be used for stating the constraints at this point.

**frg:fragmentEntry**:
A fragment entry is a placeholder which is removed when composing multiple fragments, or when the fragment is integrated into a process. A fragment entry must have one or more leaving control links, and it must not have any incoming control links. An attribute (type="mandatory | optional") specifies if the entry has to be wired or if it (and its leaving control links) can be removed. If a fragment entry has multiple leaving control links, then all of those control links must have their source in one and the same activity in the final composition.

**frg:fragmentExit**:
Analogous to an entry, a fragment exit is a placeholder for composition. It must have at least one incoming control link, and it must not have any leaving control links. A
fragment exit has the same attribute (`type`) as a fragment entry with analogous semantics. If a fragment entry has multiple incoming control links, then all of those control links must have their target in one and the same activity in the final composition.

**frg:fragmentScope**: A process fragment may define a context (e.g., variables). In BPEL the `scope` activity is used for this purpose. However, we have to distinguish between the scope of a fragment and a BPEL scope for avoiding confusion on the one hand, and for providing clear semantics on the other hand. A fragment scope can be used as container for context constructs, such as variables, partnerLinks, faultHandlers, etc. However the context of a fragment has to be merged with the context of the process in which it is used during integration. Therefore a fragment scope has the same characteristics as a normal BPEL scope in terms of XML schema, but not the same semantics.

**frg:fragmentFlow**: BPEL is a hybrid language, both graph-based and block-structured [KMW+09]. For supporting the concept of multiple possible entries and exits of a fragment we have decided to focus on the graph-based part, i.e. we have taken the BPEL `flow` construct as basis. Just like a fragment scope is not the same as a BPEL scope, a fragment flow is not the same as a normal BPEL flow. When integrating the fragment into a process, the control links which are nested in the `fragmentFlow` have to be merged with the control links which are nested in the ancestor BPEL `flow`, and the `fragmentFlow` has to be removed.

**XML schema for process fragment extensions:**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns="http://www.iaas.uni-stuttgart.de/ext/fragment"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
   targetNamespace="http://www.iaas.uni-stuttgart.de/ext/fragment">
  <xsd:import
    namespace="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    schemaLocation="bpel.xsd"/>
  <xsd:element name="fragmentExit">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="bpws:targets"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute name="type" type="tWiring" use="required"/>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="fragmentEntry">
    <xsd:complexType>
      ...
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
<xsd:complexType name="fragmentFlow">
  <xsd:complexContent>
    <xsd:extension base="bpws:tFlow"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="fragmentScope">
  <xsd:complexContent>
    <xsd:extension base="bpws:tScope"/>
  </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="tWiring">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="mandatory"/>
    <xsd:enumeration value="optional"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>

Listing 5  XML schema for process fragment extensions
XML schema for unique identifiers:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.iaas.uni-stuttgart.de/ext/id"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.iaas.uni-stuttgart.de/ext/id">
  <xsd:attribute name="id" type="xsd:string"/>
</xsd:schema>
```

Listing 6  XML schema for unique identifiers

5.1.4. Process fragment XML example

The example shown in the listing below (Listing 7) is the serialization of the refined process fragment for approval, shown in Figure 3 in Section 4.4. The serialization is based the XML schema presented in the previous sub section, and instantiates the discussed meta model for process fragments.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bpws:process name="CheckFragment" targetNamespace="CheckFragment"
    xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:tns="CheckFragment"
    xmlns:frg="http://www.iaas.uni-stuttgart.de/ext/fragment"
    frg:schemaLocation="fragment.xsd"
    xmlns:ext="http://www.iaas.uni-stuttgart.de/ext/id"
    ext:schemaLocation="id.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.iaas.uni-stuttgart.de/ext/fragment fragment.xsd">
  <bpws:extensions>
    <bpws:extension
        namespace="http://www.iaas.uni-stuttgart.de/ext/fragment"
        mustUnderstand="yes"/>
    <bpws:extension
        namespace="http://www.iaas.uni-stuttgart.de/ext/id"
        mustUnderstand="no"/>
  </bpws:extensions>
  <bpws:import
      namespace="http://www.iaas.uni-stuttgart.de/ext/fragment"
      location="fragment.xsd"
      importType="http://www.iaas.uni-stuttgart.de/ext/fragment"/>
  <bpws:import
      namespace="http://www.iaas.uni-stuttgart.de/ext/id"
      location="id.xsd"
      importType="http://www.iaas.uni-stuttgart.de/ext/id"/>
</bpws:process>
```
Listing 7  Process fragment for approval in BPEL code

```xml
<bpws:source>
  <bpws:source linkName="link2"/>
</bpws:source>
</frg:fragmentRegion>
<brpws:invoke inputVariable="checkInput" name="performCheck"
  outputVariable="checkResult" partnerLink="taskManager"
  portType="tns:CheckFragmentCallback" operation="param"
  ext:id="016">
  <bpws:targets>
    <bpws:target linkName="link2"/>
  </bpws:targets>
  <bpws:sources>
    <bpws:source linkName="link3">
      <bpws:transitionCondition>
        <![CDATA[checkResult = true]]>
      </bpws:transitionCondition>
    </bpws:source>
  </bpws:sources>
</bpws:invoke>
<brpws:fragmentExit name="decline" type="mandatory" ext:id="017">
  <bpws:targets>
    <bpws:target linkName="link4"/>
  </bpws:targets>
</brpws:fragmentExit>
<brpws:fragmentExit name="approve" type="mandatory" ext:id="018">
  <bpws:targets>
    <bpws:target linkName="link5"/>
    <bpws:target linkName="link3"/>
  </bpws:targets>
</brpws:fragmentExit>
</frg:fragmentFlow>
</frg:fragmentScope>
</bpws:extensionActivity>
</bpws:process>
```
5.1.5. Transformation of process fragments to standard BPEL code

Process design tools (such as the View-based Modelling Framework, cf. [D1.3]) have to be extended accordingly for being able to handle the BPEL extensions for process fragments. For being able to use or at least to properly display a process fragment for compliance also in standard BPEL design tools (i.e., tools which do not support the extensions we propose), the fragment needs to be transformed to standard BPEL code. Such transformation functionality could increase the overall adoption of the concept, as it provides for basic interoperability between tools which implement the BPEL extensions and those tools which do not. This can be done by removing the extension declarations, removing the extension identifiers, replacing each fragmentRegion, fragmentEntry and fragmentExit by an empty activity, and the fragmentScope by a regular BPEL scope and the fragmentFlow by a regular BPEL flow activity. The names of these constructs have to be adjusted, respectively. However, one should keep in mind that the semantics are not the same and that this “process” is not a potential executable completion of a process fragment. The investigation on potential completions of a process fragment to a complete process is future work. Figure 6 shows a visualization of the process fragment from Listing 7, transformed to standard BPEL constructs and opened with the Eclipse BPEL Designer (cf. [ECLIPSE09]).

![Figure 6 Viewing a process fragment with standard BPEL constructs](image)

We have implemented the transformation using XSL Transformation (XSLT) stylesheets. We use these stylesheets in combination with an existing XSL transformer [NXSLT3] in order to generate standard BPEL code from a process fragment which can then be used in any standard BPEL design tool. In Listing 8 (see below) the transformation template which we apply in the transformation prototype is shown. We have also created an according batch file for enabling file transformation via drag & drop, see Listing 9 for the according batch code. The integration of this tool into the “View Transformation” mechanism in the fragment-oriented artefact repository [D4.4] is ongoing work.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs1:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:frg="http://www.iaas.uni-stuttgart.de/ext/fragment"
  xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
```
5.2. BPEL extension for traceability

In COMPAS we apply model-driven development for generating BPEL code (cf. [D1.2]). That is, model-instances of the various views from the View-based Modelling Framework are used by model-to-code templates for producing deployable process definitions. For relating to these model-instances and views at runtime we apply an integrated Model-Aware Service Environment (MORSE), presented in [D4.4]. Particularly, we (i) embed traceability information into the process as a BPEL extension and, together with (ii) eventing, we use (iii) so called “model-aware services”. Such model-aware services support BPEL processes in the sense that they can look up and work with the model-driven development (MDD) artifacts. For instance, processes have been generated from
model-instances and views, and are still related to them. During runtime, the model-aware services receive events from the eventing infrastructure that contain MORSE identifiers and query the model repository. The model repository manages the respective MDD projects and artifacts. In general, processes and process engines do not interact with the model repository or model-aware services. However, they can integrate with model-aware services in the sense that the latter receive events from the engine that hold the Universally Unique Identifiers (UUID) of MORSE objects. Thus, these UUIDs have to be supplied as traceability information to the process.

BPEL extensions provide a standard way to realise this. Listing 10 shows an excerpt of a BPEL process with a BPEL extension for mapping code elements of the BPEL process to MORSE object identifiers (for simplicity namespaces have been omitted). The traceability element that indicates the UUID of the build as an attribute, is a sequence of rows that maps BPEL elements to the UUIDs of corresponding MORSE objects. The XML Path Query Language (XPath) is chosen as the default query language for selecting the XML elements of the BPEL code. For extensibility, an optional queryLanguage attribute that has the same semantics as in BPEL (cf. Section 8.2 of WS-BPEL), can specify an alternative query language or XPath version. The XML Schema for traceability can be found in Listing 11.

```xml
<process name="DeploymentProcess">
  <extensions>
    <extension mustUnderstand="yes" namespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"/>
  </extensions>
  <import importType="http://www.w3.org/2001/XMLSchema" namespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd" location="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability.xsd"/>
  <morse:traceability buildId="56810150-5bd8-4e8e-9ec5-0b88a205946b">
    <row query="/process[1]">
      <uuid>c6d2a636-747d-4c1b-8b7a-b32f59f0ac8c</uuid>
      <uuid>e4963cf9-f4d3-4f72-abe5-f3a4e2e26c30</uuid>
      <uuid>808ffa5d-d03e-465f-b931-0ada1d3b29d3</uuid>
      <uuid>5ba40ed1-3039-47c1-ba69-6c7a0362907a</uuid>
    </row>
    <row query="/process[1]/sequence[1]/receive[1]">
      <uuid>d923399a-ef5d-455c-9fa7-8be23df55891</uuid>
    </row>
    <row query="/process[1]/sequence[1]/receive[1]/@variable[1]">
      <uuid>b52e218c-988e-418b-ad92-87aa533b1387</uuid>
    </row>
  </morse:traceability>
</process>
```
Listing 10 BPEL process with an extension for MORSE traceability

Note, that this traceability information can be used to annotate any XML based target code. It can be supplied as an inline extension and does not have to be defined in a separate file. As a consequence, our approach is not limited to BPEL but can directly be applied to other XML and Web service based technologies and standards. The traceability information can also be applied to programming languages such as Java or C#, e.g., as annotations to classes, interfaces, methods, and parameters. These annotations can be added to the source code or can be realised externally in an annotation file that decorates annotated classes. In our BPEL example the MORSE builder weaves UUIDs of the MORSE objects into BPEL code during generation time. At deployment time, the BPEL engine needs to support the BPEL extension, i.e., for the namespace that is used for the MORSE traceability extension there is an implementation at the BPEL engine in place. At runtime, this extension submits events that contain the identifiers of e.g., the process or process activities, an event type, and optional further properties. Some events of interest are process instantiation and termination and pre-events and post-events for the execution of activities. More information concerning events and the event model used in COMPAS can be found in [D5.3] and [D5.4]. Finally, the events are received by model-aware services that look-up the MORSE objects for e.g., monitoring, auditing, reporting, or business intelligence scenarios. More details concerning the MORSE approach can be found in [HZD09].

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema
 xmlns:morse="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability"
 xmlns:xmi="http://www.omg.org/XMI"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://xml.vitalab.tuwien.ac.at/ns/morse/traceability">
 <xsd:import namespace="http://www.omg.org/XMI"
 schemaLocation="XMI.xsd"/>
 <xsd:complexType name="Traceability">
  <xsd:choice maxOccurs="unbounded" minOccurs="0">
   <xsd:element name="row" type="morse:TraceabilityRow"/>
   <xsd:element ref="xmi:Extension"/>
  </xsd:choice>
  <xsd:attribute ref="xmi:id"/>
  <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
  <xsd:attribute name="build" type="xsd:string"/>
 </xsd:complexType>
</xsd:schema>
```
Listing 11  XML schema for MORSE traceability

```xml
</xsd:complexType>
<xsd:element name="traceability" type="morse:Traceability"/>
<xsd:complexType name="TraceabilityRow">
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element name="uuid" nillable="true" type="xsd:string"/>
        <xsd:element ref="xml:Extension"/>
    </xsd:choice>
    <xsd:attribute ref="xml:id"/>
    <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
    <xsd:attribute name="queryLanguage" type="xsd:string"/>
    <xsd:attribute name="query" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:element name="traceabilityRow" type="morse:TraceabilityRow"/>
</xsd:schema>
```
6. Prototype implementation details

The D4.2 prototype components for dealing with the BPEL extensions are integrated in the supporting infrastructure components (cf. prototype [D4.4]). D4.2 provides prototypes in form of XSDs and examples for these BPEL extensions, APIs that demonstrate how to handle these extension e.g., in process engines or repositories, and a fragment transformer tool (cf. Section 5.1.5). [D4.4] provides an integrated runtime that can run on its own. It includes the APIs from D4.2.

The D4.2 prototype consists of the following components:

- **Language Extensions:** The language extensions provided with this prototype formally define extensions to the BPEL language in form of XML schema definitions. The schema definitions are required for enabling infrastructure components (process artefact repository, process engine, process generation tool, cf. [D4.4]) to properly handle the extensions.

- **Process Fragment Transformer:** The process fragment transformer is a tool based on XSL transformation (XSLT). This tool transforms a process fragment for compliance (which uses the language extensions) into standard BPEL code, for allowing usage of process fragments also in tools which do not support the language extensions. The implementation of the process fragment transformer is described in Section 5.1.5. This tool can be used stand-alone, but we are also working on integration into the fragment-oriented repository Fragmento, cf. [D4.4].

- **Validation Integration:** The validation integration provides an API with according Java classes for enabling integration of the language extensions into the fragment-oriented repository Fragmento [D4.4]. This integration allows validating process fragments before (or after) they are stored in the repository.

- **Traceability Integration:** The traceability integration provides an API with according Java classes for enabling the emission of execution events augmented with traceability information. This API is used in the process engine (cf. [D4.4]) to properly execute processes that use the language extensions for traceability.

In the following we briefly introduce the main artefacts provided with this prototype.

Artefacts related to the Language Extensions:

- fragment.xsd: This file contains the XML schema definition for the process fragment extensions for BPEL. This schema is required for enabling infrastructure components (fragment repository, process generation tool) to properly handle the extensions (cf. Section 5.1.3).

- id.xsd: This file contains the XML schema definition for the id extension attribute for BPEL. As this extension is used in a different configuration it is stored in a separate file (cf. Section 5.1.3).

- traceability.xsd: XML schema definition for the traceability extension. This schema is required for enabling infrastructure components (model repository, process engine) to properly handle the extensions (cf. Section 5.2).
• CheckFragment.xml: This file represents a sample process fragment serialized in XML. This process fragment is valid against the BPEL 2.0 schema with the process fragment extension (cf. Section 5.1.4).

• bpel.xsd: This file contains the XML schema definition for BPEL 2.0 executable processes. It is included in the prototype for enabling validation of the process fragments in XML, and for validation of the process fragment XML schema definition, but not produced in COMPAS.

• XML.xsd: This file contains the XML schema for XML Metadata Interchange, required for validation of the traceability schema definition. It is included in the prototype for enabling validation of the traceability schema definition, but not produced in COMPAS.

Artefacts related to the Process Fragment Transformer:

• xform.xslt: This XSL transformation stylesheet is used in the process fragment transformer for the definition of the transformation rules (cf. Section 5.1.5).

• xform.bat: This batch file is used for enabling transformation of process fragments via drag & drop in the Windows Explorer. Of course, it can also be used in the command shell.

• SampleFragment.xml: Process fragment example using the language extensions, provided for testing.

• output.bpel: This is where the output of the transformation is stored.

• howTo.txt: This is the usage manual for the transformer.

• nxslt3.exe: This is the application used for performing the XSL transformation (cf. [NXSLT3]). It is included in the prototype for enabling the actual transformation, but not produced in COMPAS.

Artefacts related to the Validation Integration:

• Validator.java: This file contains the code for invoking the validation of a process fragment against the according XML schema (i.e., the language extensions).

• Config.java: This class contains the configuration management for the validation. In this class validation can be switched on and off for all types of artefacts stored in the repository.

• \lib: This folder contains required libraries

• \bin: This folder contains the java byte code (generated during build time)
Artefacts related to the Traceability Integration:

- The implementation for support of the traceability extension for BPEL described in Section 5.2 affects the following packages and the classes within the process engine (Apache ODE, cf. [D4.4]):

  - Package org.apache.ode.bpel.evt
    
    o TraceabilitySupport.java: The interface TraceabilitySupport enables the management of traceability information.
    
    o BpelEvent.java: The abstract class BpelEvent has been adapted to implement the interface TraceabilitySupport.

  - Package org.apache.ode.bpel.engine
    
    o TraceabilityInfo.java: This class represents the Traceability information concept including information regarding build, query, query language and Universally Unique Identifier (UUID).
    
    o TraceabilityUtils.java: The class TraceabilityUtils provides methods to extract the traceability element from the BPEL document.
    
    o BpelProcess.java: This class has been adapted to allow the attachment of the traceability element to events generated by the engine.

7. Conclusion

In this deliverable, we presented the specification and definition of extensions to the BPEL language in order to enable the implementation of the compliant service composition specifications. The D4.2 prototype components for dealing with the BPEL extensions (XSDs and examples for the BPEL extensions, APIs that demonstrate how to handle these extension) are already integrated in the supporting infrastructure, cf. [D4.4]. Furthermore, in this deliverable we provided the basis for specification of a standardisation proposal regarding extensions to BPEL for enabling reusable and compliant service orchestrations.

In our current work we use the language extensions for the definition of reusable process artefacts. The fragments can either be integrated into existing processes or be composed to complete processes. These processes can then be executed by a process engine (cf. prototype [D4.4]). The overall task objective (cf. Task T4.1, [DoW]) for definition of an executable language to enable compliance of service compositions and improve reusability of service compositions has been fulfilled.
8. Reference documents

8.1. Internal documents


8.2. External documents


1.2 – Framework (WS-Policy), W3C Member Submission, April 2006,
http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/

Rowley: BPELJ: BPEL for Java, White Paper, 2004,
http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-
bpelj/ws-bpelj.pdf

[CC09] Creative Commons: Attribution-Noncommercial-Share Alike 3.0 United

[CCM+01] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana: Web Services
Description Language (WSDL) 1.1. W3C Note, March 2001,
http://www.w3.org/TR/wsdl

[Che76] P. Chen: The Entity-Relationship Model: Toward a Unified View of Data.
http://doi.acm.org/10.1145/320434.320440

http://www.ebpmi.org/ws-per/ws-per/ws-bpel20.html

[ECLIPSE09] Eclipse BPEL Project: Eclipse BPEL Designer, 2009,
http://www.eclipse.org/bpel/

Environment. Proceedings of the 4th IEEE Asia-Pacific Services Computing

[IBM09] IBM WebSphere Integration Developer 6.2 Information Center: Working with
BPEL extensions, 2009,
om.ibm.wbit.620.help.bpm.ui.doc/concepts/concept.html

[Ka06] D. Karastoyanova: Enhancing Flexibility and Reusability of Web Service
Flows through Parameterization, PhD thesis, TU Darmstadt and Universität

Schmidt, and I. Trickovic: WS-BPEL Extension for Sub-processes – BPEL-
SPE. White Paper, September 2007,
http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-
bpelsubproc/ws-bpelsubproc.pdf

[KLN+06] D. Karastoyanova, F. Leymann, J. Nitsche, B. Wetzstein, and D. Wutke:
of the 4th International Conference on Business Process Management (BPM),
Vienna, Austria, September 2006.

Graph-Based and Block-Structured Business Process Modelling Languages.

[LMS05] P. Leach, M. Mealling, R. Salz: A Universally Unique Identifier (UUID) urn
Namespace, RFC4122, 2005.


[All links were last followed on December 15, 2009.]
## Appendices

### A. Appendix A

#### A.1. Related standards and technologies

<table>
<thead>
<tr>
<th>Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPMN</td>
<td>The Business Process Modeling Notation (recently: Business Process Model and Notation) is meant to be a higher level, graphical language for modelling business processes. Existing versions are 1.0 and 1.1, version 2.0 is in standardization progress.</td>
</tr>
<tr>
<td>WS-BPEL</td>
<td>The business process execution language for web services has been adopted by industry as the standard for service orchestration. Version 1.0 and 1.1 (BPEL4WS) have been driven by companies such as IBM, Microsoft and SAP and it has been standardized by OASIS and released as version 2.0, WS-BPEL.</td>
</tr>
<tr>
<td>WSDL</td>
<td>The Web Service Description Language defines a XML format for “describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information.” [CCM+01]. WSDL is used as interface description language for WS-BPEL, as every process is exposed as a Web service.</td>
</tr>
<tr>
<td>XSL</td>
<td>W3C. Extensible Stylesheet Language (XSL), “XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts: XSL Transformations (XSLT) is a language for transforming XML, the XML Path Language (XPath) is an expression language used by XSLT to access or refer to parts of an XML document. (XPath is also used by the XML Linking specification) XSL Formatting Objects (XSL-FO) is an XML vocabulary for specifying formatting semantics, 2009. <a href="http://www.w3.org/Style/XSL/">http://www.w3.org/Style/XSL/</a></td>
</tr>
</tbody>
</table>

Table 4  Related standards and technologies