



Fragmento: Process Fragment Library

– Documentation –

Flexibly reacting to requirements coming from laws, regulations, and internal policies becomes an essential part of business process management. Compliance refers to the measures that need to be taken in order to adhere to these requirements. In order to comply, companies need to perform profound changes in their organizational structure, processes and their supporting infrastructure. At this, process-awareness is a basic prerequisite for ascertaining to operate in a compliant way.

The Fragmento repository focuses on the application of the emerging concept of process fragments in the field of compliance management in process-based applications. We propose to specify compliance controls as reusable process fragments. We understand a process fragment as a connected, possibly incomplete process graph which may also contain additional artifacts like the fragment context. A process fragment is not necessarily directly executable as some parts may be explicitly stated as opaque in order to mark points of variability.

Fragmento is available Open Source under Apache 2 License [Apache04]. This document describes the requirements for its development and provides implementation details.

Requirements and specification

In this section we present “Fragmento”, the **Fragment-oriented** process artifact repository. Fragmento is dedicated to management of processes and process fragments for usage in the field of compliance. We begin by describing the basic requirements for the repository. Next, we sketch the code base that the repository is built on. Following this, we describe advanced functions and extensions that the repository supports. The application architecture and the concepts which are realized in Fragmento have been presented in the community [SKL+10]. Following discussions confirmed that the concepts and the supporting infrastructure we developed are a useful contribution to service-oriented information systems.

Basic requirements

A repository for process artifacts should account for storage & retrieval and version management of all artifacts related to a process. In principle, it should support the CRUD operations: Create, Read, Update, and Delete. However, deleting contents from a repository may lead to broken references.

First of all, we can distinguish different types of artifacts that are related to a process or to a process fragment running in a service-based environment.

- A process or process fragment model, either in standard BPEL or in an extended or modified version of BPEL
- An according WSDL document (for specifying the interface)
- A deployment descriptor according to the process or the process fragment
- Additional information for a process modelling tool (e.g., for storing graphical information for a modelling tool like (X/Y) coordinates of activities within a <flow> construct)
- A view transformation rule (explained later)
- Annotation documents (e.g., a policy specified in WS-SecurityPolicy)

The repository assigns unique identifier to each artifact being stored. The usage of unique identifiers allows creating relations between artifacts. The repository should provide an advanced management of relations and should furthermore allow annotation of process fragments to a given process for describing constraints. The above listed types of artifacts are typically serializable to XML, therefore the repository should be able to store XML artifacts. Additional metadata, such as a description, keywords and a name are also applicable to all of the listed types. Figure 1 shows the conceptual model of the different types of artifacts and their relation.

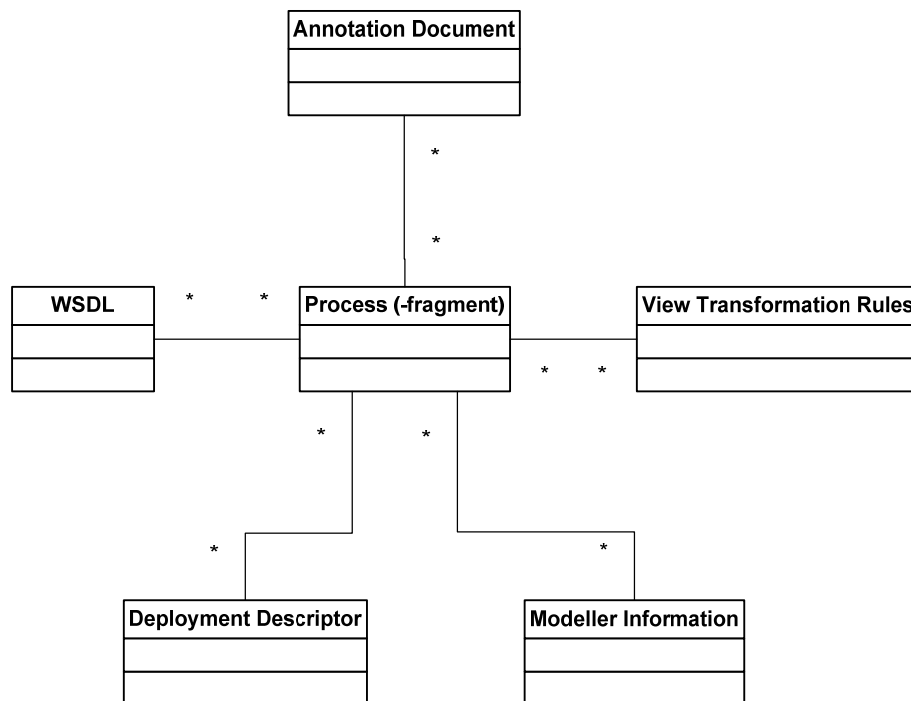


Figure 1 Conceptual model for Fragmento

From this analysis we can derive the requirements concerning the *model of the artifacts* that should be stored and managed in the repository. An artifact consists of the following parts:

1. A unique identifier
2. Metadata (Name, description, keywords etc.)
3. An XML document

4. A type (Fragment, WSDL etc.)
5. Relation(s) to other artifacts

For providing a flexible way to specify *relations* among artifacts a relation has to consist of the following parts:

1. A source (i.e. an artifact that is source of the relation)
2. A target (i.e. a different artifact)
3. A relation type (e.g. annotation)
4. A description for further characterization

Furthermore, efficient *searching* for artifacts is a fundamental requirement and should be well supported. Concerning search functionality we specify the following requirements:

- Search for a match in the description
- Search for a match in the content (i.e. within the XML document)
- Search by date of creation (by specifying an interval)
- Search by artifact type
- Search by date of creation, restricted to a particular type
- Search for artifacts which are related to a particular artifact

The repository should provide basic operations for the *management* of the artifacts:

1. Creation of an artifact
2. Checking out of an artifact (and locking the current version)
3. Releasing a lock (i.e. undoing the check out)
4. Checking in (and thereby creating a new version and possibly new relations)
5. Retrieving the version history of an artifact
6. Retrieving a particular version
7. Retrieving the latest version

For the management of the relation which can be defined among the artifacts, the repository should also provide basic operations:

1. Creation of a relation
2. Updating a relations
3. Deleting a relation

The definition of relations allows on the one hand specifying, which artifacts belong together. On the other hand they also allow defining an annotation of one artifact to another, e.g. the annotation of a process fragment to a process. According operations are required for resolving such relations. For instance, an operation for retrieving a complete process *bundle* would provide a comfortable way to retrieve a process and all related artifacts.

For integration with other components the repository should provide its *interfaces* as a Web Service, using WS-I basic profile for interoperability (see <http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>), i.e. it should provide a SOAP/HTTP binding for the operations that it exposes. For easy integration with other tools, the operations for retrieving an artifact should be provided in a RESTful manner, i.e. accessible via HTTP/GET.

XML-oriented Repository as code base

Fragmento uses a code base for a repository that has been developed by the MASTER project (cf. [MASTER09]). The repository by MASTER is a general, though XML-oriented repository, and therefore well-suitable for the described purpose. The code base of this repository is open source (see [XMLO09]), the project-specific extensions are not available but are also not required for running it).

When a new artifact is created, the repository creates a new “Versioned Object”. This object is the container for the different versions of the artifact (cf. Figure 2). The MASTER repository supports the model shown in Figure 3 for the versioning of artifacts. The Versioned Object does also provide a Version History object that allows accessing the root version (i.e. the first version) and the base version (i.e. the latest version). The internal representation of a version of an artifact is a “Version Descriptor” object. This is a container that stores the date of creation, metadata and a reference to the XML document of the artifact.

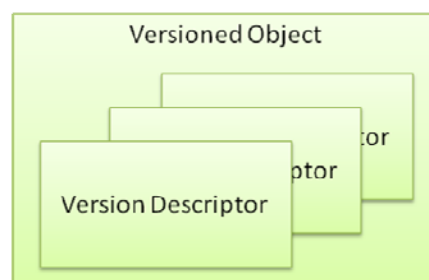


Figure 2 Versions of an artifact

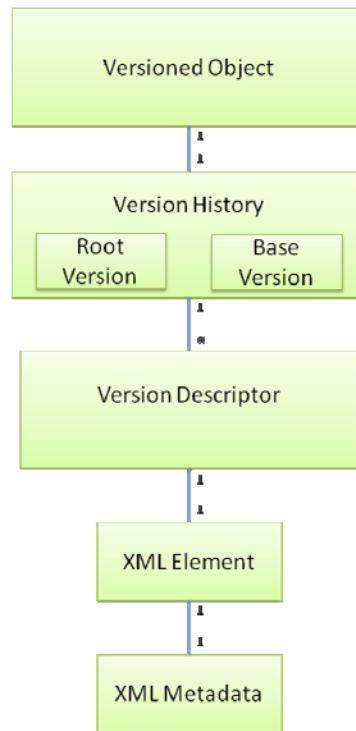


Figure 3 Model of the version management

The MASTER repository does also allow creating relations among version descriptors (cf. Figure 4). This feature can be used for creating a bundle of artifacts, and it can also be used for creating an annotation of a fragment to a process, and for creating a textual annotation, respectively. It is important to note that the relations are created among Version Descriptors and not among Versioned Objects. This circumstance allows distinguishing between the relations of artifacts in different versions, but it also requires management of the affected relations when a new version of an artifact is created.

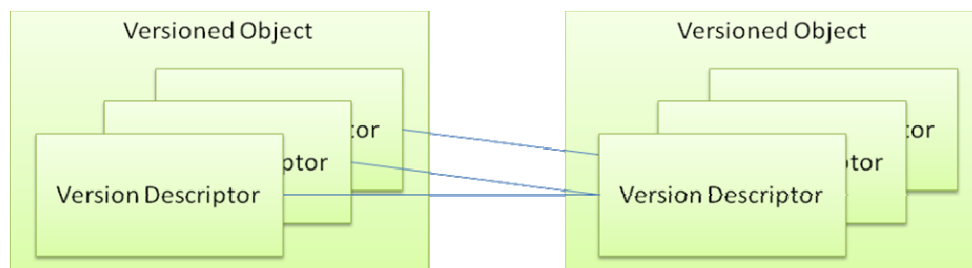


Figure 4 Relations among artifacts

Extensibility

Fragmento provides support for management of process artifacts which goes beyond the functionality of a general purpose repository. The additional functionality also represents the rationale for separating the storage of models in two different repositories. We have identified three different groups of functions that are helpful in the management of processes, process fragments and related artifacts:

1. **Validators:** The artifacts related to process enactment have to comply with particular requirements: a WSDL document has to conform to a given format and a BPEL process has to comply with particular rules. The integration of custom validators provides a guarantee that an artifact complies with particular syntactical requirements. The validation can be executed as a prerequisite for check in, or otherwise be executed on demand.
2. **View Transformations:** In a view transformation particular transformations are applied to a process model. The view transformation implementation in Fragmento supports omission of activities or attributes. The outcome of a view transformation is referred to as “process view” [SLS10]. We have identified several scenarios, in which a view transformation supports the management of a process: A view on a process can be used for abstraction. This can for instance be used for providing a perspective on a process that is personalized for specific aspects that are relevant to an auditor, or for generating a public view on a process that hides confidential details.
3. **Custom query functions:** Beyond the metadata of an artifact (supported in Fragmento), other information is of interest for search, like the interfaces a process should provide, or the structure of a process fragment (not supported in Fragmento). The structure of a process fragment refers to the way in which process activities are connected together, arranged or organized [ML09]. Query of structural information is especially useful if the compliance encoded in the process fragments imposes constraints on the control flow of the process activities [SLM+10]. To support advanced query mechanisms, Fragmento provides an extension mechanism so that such functionality can be added.

Design and implementation

Application design

Concerning the backend, Fragmento is built on the technology stack that has been introduced by the MASTER repository [XML09]. That is, a Tomcat application server [Apache09b] which is hosting the repository application. Hibernate [Hibernate09] is used as data abstraction layer. Furthermore the Spring Framework [Spring09] is employed for object lifecycle management and a PostgreSQL database [Postgre09] is utilized for storage. Concerning the development of the Web service interfaces, Axis 2 libraries [Apache09a] are used. The Web client is built using Java Server Pages (JSP) and Tag Libraries [Display09] for the view, while Servlets are used as the controller for handling client requests. Fragmento is overall written in Java.

Besides some minor code adaptations concerning the management of relations and the implementation of search functionality, the code base of the MASTER repository has been used and extended. Figure 5 shows an outline of the functionality that has been implemented for Fragmento in addition to the used code base. The extensions package hosts the extensions for custom queries, view transformations and custom validators. The io package hosts the XML parsers and serializers that are used for parsing and serializing XML messages that are sent to and from the Fragmento Web service. The service package holds the actual implementation of the Fragmento Web service. The util package provides classes and functions that are utilized within the overall implementation. For instance, the Artifact.java class is utilized for display of an artifact on a web page. The web package contains the Servlets that manage client request of the web client. The webservice package contains the classes for the Web service interface which are automatically generated from the WSDL document of the Fragmento Web service. Most of the additional code is concerned with the Web service interfaces and the Web client. The latter has been developed from scratch.

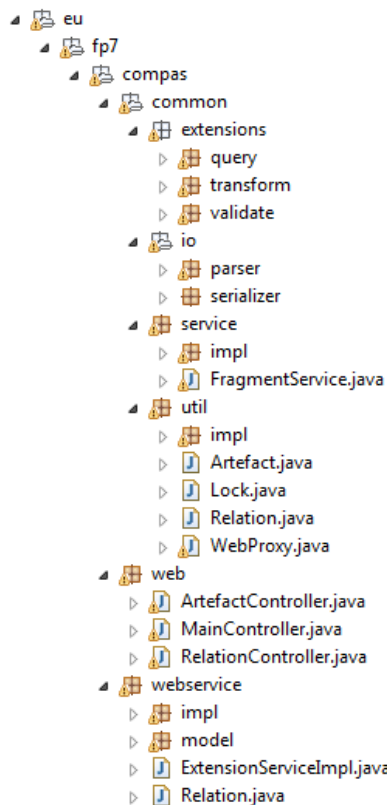


Figure 5 Fragmento package structure

Web service interfaces

The Web service interface of the Fragmento repository provides the following operations:

Concerning artifacts:

1. **createArtifact**: This operation is used to create a new artifact in the repository. The operation returns the identifier of the created version descriptor.
2. **retrieveArtifact**: This operation is used to retrieve a particular version of an artifact, without performing a check out. This operation optionally allows specifying or referencing a view transformation rule that is applied to the artifact before it is returned.
3. **retrieveArtifactBundle**: This operation returns an artifact and all the artifacts that are related to it.
4. **retrieveArtifactHistory**: This operation returns a list of version descriptor identifiers that represent the version history of an artifact.
5. **checkoutArtifact**: This operation sets a lock on the requested artifact and returns it. It also returns a lock identifier which is required for check in.
6. **checkinArtifact**: This operation creates a new version of an artifact. For authorization also the corresponding lock identifier has to be passed. Based on the parameter `keepRelations`, the relations of this artifact also apply to its new version (i.e. Fragmento creates new relations). It returns the identifier of the new version.
7. **browseArtifacts**: This operation implements the search function. Based on the input parameters this operation returns a list of version descriptors that match the query

(considering only the head revision, i.e. the latest versions). As input parameters are accepted: artifact type, interval of creation, or search string for description or document content.

8. `retrieveArtifactLatestVersion`: This operation returns the latest version of an artifact. In order to retrieve past revisions, the operation `retrieveArtifact` has to be used.
9. `browseLocks`: This operation returns a list of all locked artifacts.
10. `releaseLocks`: This operation can be used to release a lock.

Concerning relations:

1. `createRelation`: This operation allows creating a relation from one artifact to another. The description can be characterized by a type and a description. This operation can also be used to create annotations.
2. `retrieveRelation`: This operation returns the details of a relation.
3. `browseRelations`: This operation provides search functionality for relations. Valid input parameters are a source (a version descriptor identifier), a target, a type, or an interval for its creation.
4. `updateRelation`: This operation provides the update mechanism for relations. As relations are not versioned, there is no check out or check in function.
5. `deleteRelation`: This operation completely deletes a relation.

Web client

Fragmento provides all required functionality via Web service interfaces which can be exploited for integration with rich client applications, e.g., based on Eclipse. For easily accessing the Fragmento repository and managing the artifacts and relations stored therein we also provide a Web client. Figure 6 shows the start page of the Fragmento Web Client.

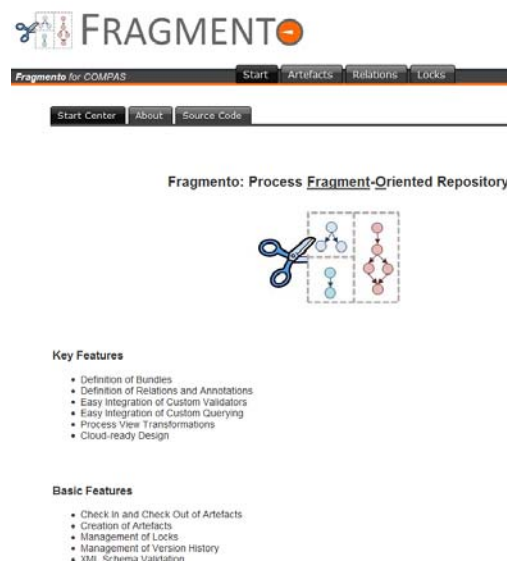


Figure 6 Start page of the Fragmento web client

We have created a double-stage navigation. On top level, the user can choose between the management of artifacts, relations or locks. On the second level the particular management functions for the according top level selection are shown. Figure 7 shows the user view for management of artifacts, and Figure 8 shows the management of relations. In the final version, the relations Annotation, Container, WSDL, Deployment, Modeler Data and Transformation are supported. These types implement the conceptual model of Fragmento.

Type	Artefact ID	Description	Show	Show Bundle	Show History	Check Out
WSDL	5691	trusted-timestamp-in-standard-code.wsdl	Show	Show Bundle	Show History	Check Out
WSDL	5684	thales.loanapprovalClient.001Artefacts.wsdl	Show	Show Bundle	Show History	Check Out
WSDL	5677	thales.loanapproval.001Artefacts.wsdl	Show	Show Bundle	Show History	Check Out
WSDL	5670	Approval-in-standard-code.wsdl	Show	Show Bundle	Show History	Check Out
WSDL	5663	Segregation-of-duty-in-standard-code.wsdl	Show	Show Bundle	Show History	Check Out
Transformation Rule	5656	Removal of identifier extensions	Show	Show Bundle	Show History	Check Out
Process	5649	Loan Approval Client Draft	Show	Show Bundle	Show History	Check Out
Process	5642	Loan Approval Process Draft	Show	Show Bundle	Show History	Check Out
Modeller Data	5635	trusted-timestamp-in-standard-code.bpelex	Show	Show Bundle	Show History	Check Out
Modeller Data	5628	thales.loanapprovalClient.001.bpelex	Show	Show Bundle	Show History	Check Out
Modeller Data	5621	thales.loanapproval.001.bpelex	Show	Show Bundle	Show History	Check Out
Modeller Data	5614	Segregation-of-duty-in-standard-code.bpelex	Show	Show Bundle	Show History	Check Out
Fragment	5607	trusted timestamp	Show	Show Bundle	Show History	Check Out
Fragment	5600	trusted timestamp in standard BPEL	Show	Show Bundle	Show History	Check Out
Fragment	5593	segregation of duty; separation of duty, 4-eyes principle; in standard...	Show	Show Bundle	Show History	Check Out
Fragment	5586	segregation of duty; separation of duty, 4-eyes principle; BPEL4People	Show	Show Bundle	Show History	Check Out
Fragment	5579	Avoidance of infinite waits in standard BPEL	Show	Show Bundle	Show History	Check Out
Fragment	5572	approval fragment with extension identifiers	Show	Show Bundle	Show History	Check Out
Fragment	5565	approval fragment in standard BPEL code with extension identifiers	Show	Show Bundle	Show History	Check Out
Annotation	5558	WS-SecurityPolicy: User Name with Certificates, Sign, Encrypt	Show	Show Bundle	Show History	Check Out
Annotation	5551	WS-SecurityPolicy: Use of SSL Transport Binding	Show	Show Bundle	Show History	Check Out
Annotation	5544	WS-SecurityPolicy: UsernameToken without password	Show	Show Bundle	Show History	Check Out

Figure 7 Artifacts management in Fragmento web client

Fragmento for COMPAS

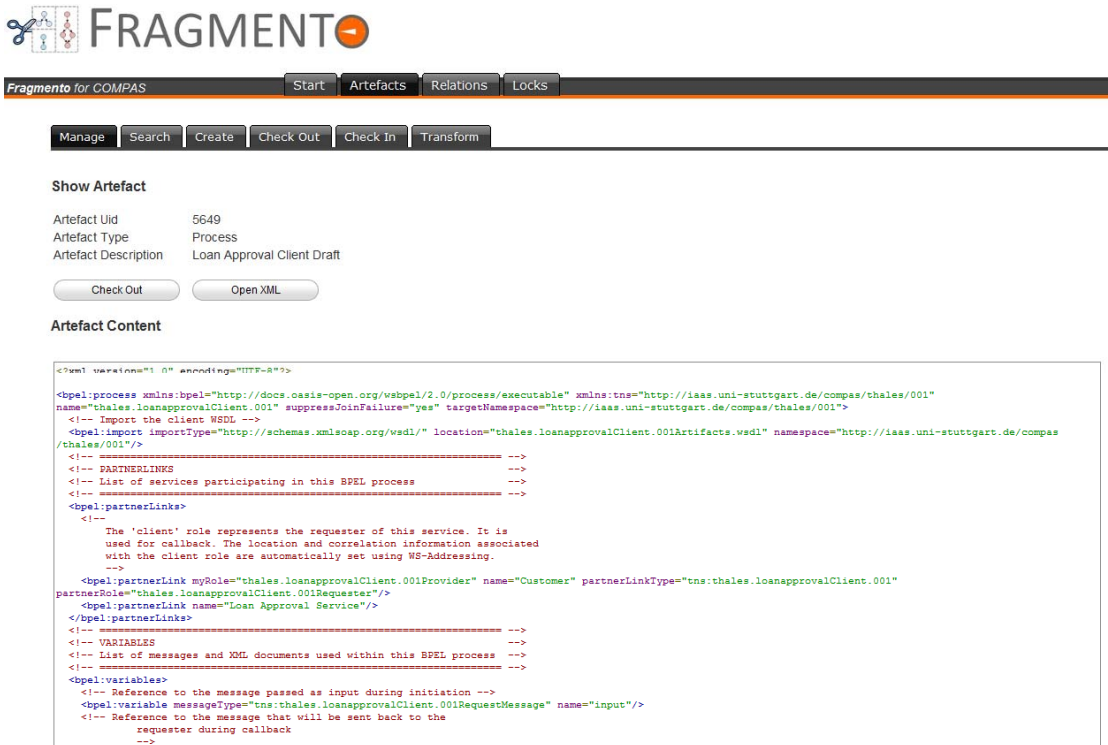
Start Artifacts Relations Locks

Manage Search Create Update

Relation Type	Relation ID	Relation Description	Relation From ID	Relation To ID	Update	Delete
modeller	5304	Modeller information for BPEL Designer	5222	5201	Update Relation	Delete Relation
modeller	5305	Modeller information for BPEL Designer	5229	5250	Update Relation	Delete Relation
modeller	5306	Modeller information for BPEL Designer	5236	5257	Update Relation	Delete Relation
modeller	5307	Modeller information for BPEL Designer	5201	5208	Update Relation	Delete Relation
annotation	5308	Security Policy	5138	5250	Update Relation	Delete Relation
annotation	5309	Security Policy	5138	5257	Update Relation	Delete Relation
wsdl	5310	WSDL for the fragment	5271	5201	Update Relation	Delete Relation
wsdl	5311	WSDL for the fragment	5278	5173	Update Relation	Delete Relation
wsdl	5312	WSDL for the process	5285	5250	Update Relation	Delete Relation
wsdl	5313	WSDL for the process	5292	5257	Update Relation	Delete Relation
wsdl	5314	WSDL for the fragment	5299	5208	Update Relation	Delete Relation
container	5347	A process	5250	5342	Update Relation	Delete Relation
container	5348	another process	5257	5342	Update Relation	Delete Relation
modeller	5517	Modeller information for BPEL Designer	5435	5414	Update Relation	Delete Relation
modeller	5518	Modeller information for BPEL Designer	5442	5463	Update Relation	Delete Relation
modeller	5519	Modeller information for BPEL Designer	5449	5470	Update Relation	Delete Relation
modeller	5520	Modeller information for BPEL Designer	5414	5421	Update Relation	Delete Relation
annotation	5521	Security Policy	5351	5463	Update Relation	Delete Relation
annotation	5522	Security Policy	5351	5470	Update Relation	Delete Relation
wsdl	5523	WSDL for the fragment	5484	5414	Update Relation	Delete Relation
wsdl	5524	WSDL for the fragment	5491	5386	Update Relation	Delete Relation
wsdl	5525	WSDL for the process	5496	5463	Update Relation	Delete Relation
wsdl	5526	WSDL for the process	5505	5470	Update Relation	Delete Relation

Figure 8 Relations management in Fragmento web client

We have also integrated a support for viewing the XML part of the artifact within the Web page, as shown in the artifact detail view in Figure 9.



FRAGMENTO

Fragmento for COMPAS

Start Artifacts Relations Locks

Manage Search Create Check Out Check In Transform

Show Artefact

Artefact Uid: 5649
 Artefact Type: Process
 Artefact Description: Loan Approval Client Draft

Check Out Open XML

Artifact Content

```

<?xml version="1.0" encoding="UTF-8"?>
<bpel:process xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable" xmlns:tns="http://iaas.uni-stuttgart.de/compas/thales/001"
name="thales_loanapprovalClient.001" suppressJoinFailure="yes" targetNamespace="http://iaas.uni-stuttgart.de/compas/thales/001">
<!-- Import the client NSDL -->
<bpel:import importType="http://schemas.xmlsoap.org/wsdl/" location="thales_loanapprovalClient.001Artifacts.wsdl" namespace="http://iaas.uni-stuttgart.de/compas/thales/001"/>
<!-- ===== -->
<!-- PARTNERLINKS -->
<!-- List of services participating in this BPEL process -->
<!-- ===== -->
<bpel:partnerLinks>
<!--
The 'client' role represents the requester of this service. It is
used for callback. The location and correlation information associated
with the client role are automatically set using WS-Addressing.
-->
<bpel:partnerLink myRole="thales_loanapprovalClient.001Provider" name="Customer" partnerLinkType="tns:thales_loanapprovalClient.001"
partnerRole="thales_loanapprovalClient.001Requester"/>
<bpel:partnerLink name="Loan Approval Service"/>
</bpel:partnerLinks>
<!-- ===== -->
<!-- VARIABLES -->
<!-- List of messages and XML documents used within this BPEL process -->
<!-- ===== -->
<bpel:variables>
<!-- Reference to the message passed as input during initiation -->
<bpel:variable messageType="tns:thales_loanapprovalClient.001RequestMessage" name="input"/>
<!-- Reference to the message that will be sent back to the
requester during callback
-->
  
```

Figure 9 Display of an artifact in Fragmento web client

As shown in Figure 10, we implemented a generic wizard, which guides the user through the different steps for creation of any type of the pre-defined relations: Annotation, Container, WSDL, Deployment, Modeler Data and Transformation. This wizard implements the annotation editor. There are basically five steps which have to be performed to create a relation. At first, the type of relation has to be chosen. In the next steps the source and the target of the relation have to be chosen. Then, additional descriptions can be entered. Having this information, the wizard can establish the desired relation between the chosen artifacts.

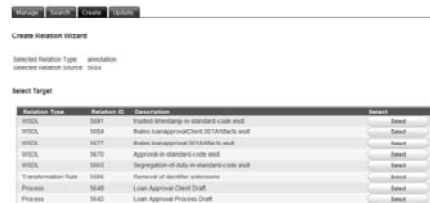
Step 1:
Selection of the type of relation



Step 2:
Selection of the source



Step 3:
Selection of the target



Step 4:
Description of the relation



Step 5:
Relation has been established



Figure 10 Wizard for creation of relations and annotations

Initial filling of Fragmento with artifacts

We created a test suite that can be used with SoapUI [SoapUI] to provide an initial filling of Fragmento with sample artifacts. The test suite (FragmentoInitialFilling-soapui-project.xml) can be easily imported in the tool via “file\import project”. Figure 11 shows the project structure of the test suite. First, annotation documents, fragments, processes and related artifacts are created in Fragmento. Then, relations between them are created by the test suite (annotation relations, bundle relations, etc.).

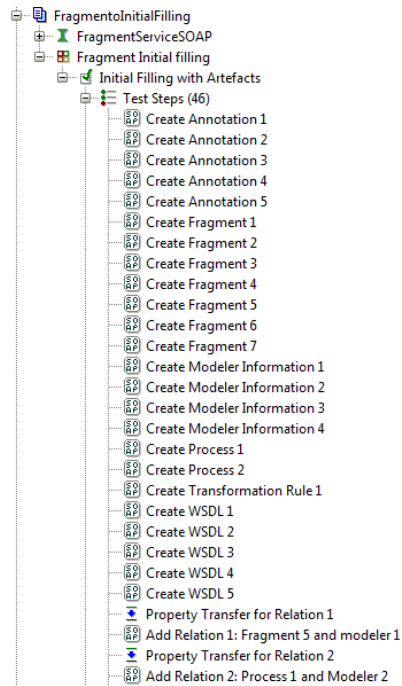


Figure 11 Project structure of the test suite of Fragmento

A simple right-click on the test suite “Fragment Initial filling” and selection of the test runner start the filling (“launch Testrunner \launch”). Then, the sample files are sent via SOAP/HTTP to the web service interface of the repository. The most recent version of the test suite is available at [Frgmto10]. This suite contains the latest versions of process fragments.

Extensions

Three groups of extensions were introduced, (i) custom validators, (ii) view transformations and (iii) custom query functions. The following extensions are currently implemented:

Concerning custom validators, we have integrated a validator for checking XML against an XML Schema. Hence, we can use this kind of validation for checking a fragment against the XML Schema for BPEL Process Fragments. The validator has been configured for validation of the common artifact types used in Fragmento, e.g., WSDL documents.

Concerning view transformation we have integrated a process view transformation framework [SLS10]. We provide full support for one sample application: The application is to remove identifiers which have been added for traceability. This function transforms a process that was extended for

traceability into standard BPEL code, in order to be usable in a standard process design or execution environment that does not provide support for this kind of extension. Figure 12 shows the Web client interface to access this function.

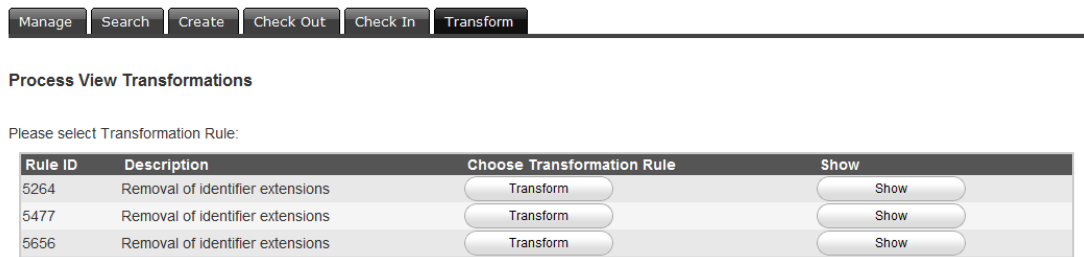


Figure 12 Process view transformations in Fragmento

The current version of the prototype does not contain an extension for custom queries. However, we provided the theoretical foundations and a discussion of implementation considerations [Pos10].

References

- [Apache04] Apache Software Foundation: Apache License Version 2.0, 2004, <http://www.apache.org/licenses/LICENSE-2.0.html>.
- [Apache09a] Apache Software Foundation: Apache Axis2, <http://ws.apache.org/axis2/>
- [Apache09b] Apache Software Foundation: Apache Tomcat, <http://tomcat.apache.org/>.
- [Display09] Displaytag Team: Display tag library, <http://displaytag.sourceforge.net/>.
- [Frgmto10] Fragmento – Fragment-oriented Repository. Project Website, 2010. <http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/start.htm>
- [Hibernate09] Red Hat: Hibernate, <https://www.hibernate.org/>.
- [MASTER09] MASTER Project Deliverable D2.3.1: Technical architecture and APIs for single trust domain, 2009, http://www.master-fp7.eu/index.php?option=com_docman&task=doc_download&gid=21&Itemid=60.
- [ML09] Z. Ma, F. Leymann: BPEL Fragments for Modularized Reuse in Modeling BPEL Processes. In: Proceedings of the 5th International Conference on Networking and Services (ICNS), IEEE Computer Society, 2009.
- [Pos10] A. Poszlovszki: Process Fragment Recognition and Emphasis. Diploma thesis no. 3001, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2010. ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-3001/DIP-3001.pdf
- [Postgre09] PostgreSQL Global Development Group: PostgreSQL, <http://www.postgresql.org/>.

- [SKL+10] D. Schumm, D. Karastoyanova, F. Leymann, S. Strauch: Fragmento: Advanced Process Fragment Library. Proceedings of the 19th Int. Conference on Information Systems Development (ISD 2010), Springer, 2010.
- [SLM+10] D. Schumm, F. Leymann, Z. Ma, T. Scheibler, S. Strauch: Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. In: Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI'10), 2010.
- [SLS10] D. Schumm, F. Leymann, A. Streule: Process Viewing Patterns. Proceedings of the 14th IEEE International EDOC Conference (EDOC 2010), IEEE Computer Society Press, 2010.
- [SoapUI] eviware: SoapUI, May 2010. <http://www.soapui.org/>
- [Spring09] SpringSource: Spring, <http://www.springsource.org/>.
- [XMLO09] XMLO – The eXtensible Markup Language repository, 2009. <http://www.iaas.uni-stuttgart.de/forschung/projects/master/xmlo.php>.