

Flexible Services

EDEN

D3.1.3: XFormsDB Language Design for Templates and Transactions

Document Name	EDEN - Deliverable document of D3.1.3: XFormsDB Language Design for Templates and Transactions
Project/WP Title:	EDEN/WP3: Business Network Evolution
Document Type, Security	P (Public)

Document Title:	EDEN - D3.1.3: XFormsDB Language Design for Templates and Transactions
Agreed date of delivery	August 30, 2008
Actual date of delivery	May 31, 2009
Editor	Markku Laine/TKK
Version and	version 0.2
Date Last Change	June 3, 2009
File:	

Participants	Name	e-mail
Elisa	?	?
Nokia Research Center	Oskari Koskimies	oskari.koskimies@nokia.com
TKK/ME	Markku Laine	mplaine@cc.hut.fi
TKK/ME	Evgenia Samochadina	samochadina@gmail.com

Table of Contents

Table of Contents	2
Tables and Figures	3
List of tables	3
List of Acronyms and Abbreviations	4
1 Introduction.....	5
2 Objectives.....	5
3 Requirements.....	5
3.1 Component requirements.....	7
3.2 Container requirements.....	7
3.3 Web Page requirements	7
3.4 Application requirements.....	8
3.5 User Interface requirements	8
3.6 The structure of the files	10
3.7 The Template markup language	13
4 Status	18
4.1 Templates.....	18
4.2 Transactions	18
5 Summary	18

Tables and Figures

List of tables

Table 1 XFormsDB Template Language summary.....	9
Table 2 Different folder structures: Pros & Cons.....	11

List of figures

Figure 1: A high-level overview of the architecture.	6
Figure 2 Folder structure of the application	13
Figure 3 Folder structure of the template	13
Figure 4 Template Language: Web Page source code	14
Figure 5 Template Language: Component source code	15

List of Acronyms and Abbreviations

XIDE	XFormsDB IDE, Service Management UI

1 Introduction

This document contains a short description of the deliverable *D3.1.3: XFormsDB Language Design for Templates and Transactions*, which is part of the Flexible Services program’s Ecosystem Design and Evolution (EDEN) project. The deliverable is closely related to tasks *T3.1.1: Management UI initial design* and *T3.1.2: Management UI implementation*.

The document starts by presenting the objectives of the deliverable. Then the current status of the deliverable is reported. Finally, Chapter 5 summarizes the results of the deliverable presented in this document.

2 Objectives

The deliverable has the following objectives:

- To define XFormsDB Language for Templates which is intermediate language for Service Management UI (XFormsDB IDE). It should be used to describe reusable parameterized elements, which are written on XFormsDB and combine them into a Web Application.
- To define XFormsDB Language for the Transactions. It should be possible to create queries, which should be executed in sequence in a transaction fashion (if one fails, the whole sequence fails).

XFormsDB Language for Templates, which is called Template Language, is required and utilized by XFormsDB IDE (XIDE) in scope of work in tasks *T3.1.2: Management UI initial design* and *T3.1.2: Management UI implementation*.

3 Requirements

The main concepts of the Template Language are the following:

Component	A reusable self-sufficient or insufficient object that represents a piece of functionality. <i>Components</i> can have parameters in order to maximize reusability.
Container	A placeholder for <i>Components</i> and other <i>Containers</i> on a <i>Web Page</i> . <i>Containers</i> are not reusable. <i>Containers</i> act as a grouping mechanism and can share information to <i>Components</i> and <i>Containers</i> placed inside them.
Web Page	An entire document, such as XHTML. <i>Web Pages</i> are the first level <i>Components</i> . They can contain <i>Containers</i> and <i>Components</i> that are placed inside <i>Containers</i> . <i>Web Pages</i> have the same parts and functionalities as <i>Components</i> except it has some additional properties special for <i>Web Pages</i> . <i>Web Pages</i> can also share information to <i>Components</i> and <i>Containers</i> placed inside them.
Application	A set of <i>Web Pages</i> that represents some “bigger” functionality, such as a home page or an e-commerce shop.

Component/ Web Page / Application template A preliminary version of the element which is stored in the system template library and ready to be used and/or modified.

The following diagram illustrates the main classes of the XFormsDB Template language. These classes are Component, Container, Web Page, and Application.

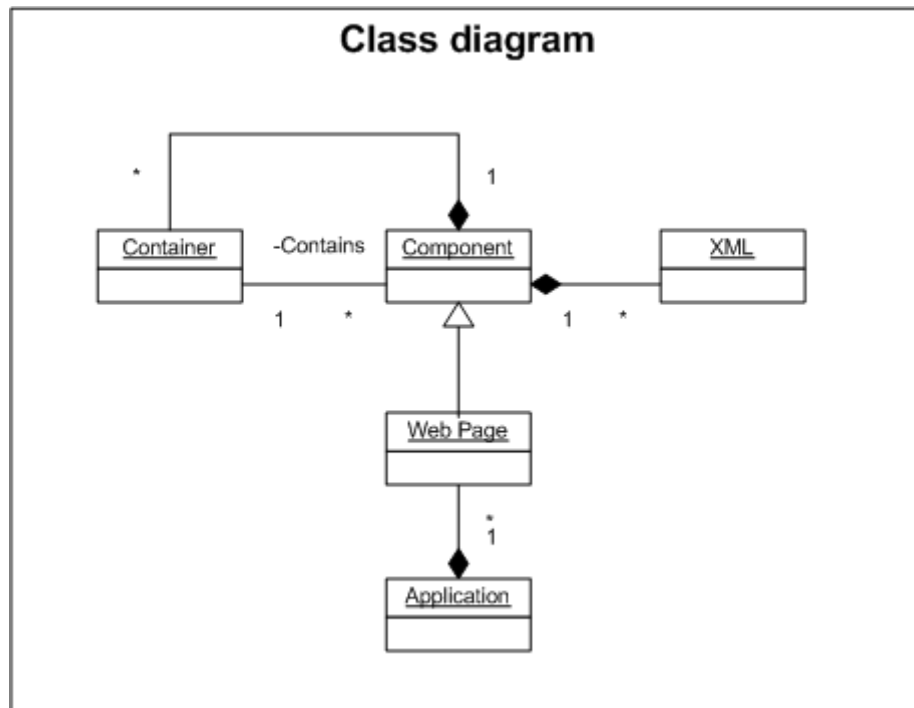


Figure 1: A high-level overview of the architecture.

The relationships between classes are as follows: 1) Components can be placed inside Containers, 2) Components can contain XML and Containers that have to be hardcoded into Component's body, 3) Web Page extends Component.

Based on the concepts definitions and XIDE functional requirements, which were described during the task *T3.1.2: Management UI initial design*, the list following list of Template Language requirements were created:

3.1 Component requirements

- 3.1.1 Components **must** be reusable in terms of using one Component source code for representing different Components of the same types.
- 3.1.2 Components **must** have a unique ID (for referencing purposes).
- 3.1.3 Components **must** have template library to provide predefined Component templates for their using and modifying.
- 3.1.4 Components **must** be either self-sufficient or insufficient. Insufficient Components should be marked and classified (e.g., using tags).
- 3.1.5 Components **must** include metadata (e.g., icon, preview image, ID, description, tags, and settings) about itself.
- 3.1.6 Components **must** include a body section i.e. UI controls.
- 3.1.7 Components **can** include parameters, which **must** be typed (e.g., XML Schema types, XML fragment, Component). The visualization of UI controls varies depending on the type of the Component.
- 3.1.8 Components **can** contain Containers.
- 3.1.9 Components **can** contain hard-coded XML.
- 3.1.10 Components **can** contain data instances, queries (XPath or XQuery), and CSS.
- 3.1.11 Access to Components or parts of Components **can** be restricted using role-based authorization.
- 3.1.12 Components **can** be modified and saved as new, private Components.

3.2 Container requirements

- 3.2.1 Containers **cannot** be reusable i.e. there is no need for a Container to have a unique ID.
- 3.2.2 Containers **can** be saved as new private Components including their contents, which are reusable.
- 3.2.3 Containers **cannot** include parameters (but they **must** inherit child Components' parameters when saved as new, private Components).
- 3.2.4 Containers **must** have the almost the same parts and functionalities as Components (see points 3.1.5, 3.1.8, 3.1.10, 3.1.11).
- 3.2.5 Containers **can** contain Components and other Containers.

3.3 Web Page requirements

- 3.3.1 Web Pages must not be reusable.
- 3.3.2 Web Pages must have template library to provide predefined Web Page templates for their using and modifying.
- 3.3.3 Web Pages must be self-sufficient.

- 3.3.4 Web Pages can be saved as new private Components including their contents.
- 3.3.5 Web Pages cannot include parameters (but they must inherit Components' parameters when saved as Components).
- 3.3.6 Web Pages must have the same parts and functionalities as Components (see points 3.1.5, 3.1.8, 3.1.10, 3.1.11).
- 3.3.7 Web Pages must contain a head and a body section.
- 3.3.8 Web Pages can contain hard-coded XML.
- 3.3.9 Web Pages must be linkable, i.e., have an URL.

3.4 Application requirements

- 3.4.1 Applications must not be reusable.
- 3.4.2 Applications must have template library to provide predefined Application templates for their using and modifying.
- 3.4.3 Applications must contain one or more Web Pages.
- 3.4.4 Applications must be working web Applications.

3.5 User Interface requirements

- 3.5.1 It must be possible to search for Components', Web Pages', and Applications' templates using, for example, key words, tags, and/or rating.
- 3.5.2 It must be possible to navigate the web Application using, for example, a tree hierarchy or selection elements directly from design view.
- 3.5.3 It must be possible to design, edit, preview, and publish web Applications.
- 3.5.4 It must be possible to visually drag and drop (add, move, and delete) Components and (delete) Containers in the design view.
- 3.5.5 It must be possible to group Components and Containers and save them as a new, private Component together with their contents.
- 3.5.6 Private Components can be used only by the author. Each Component can be shared (published) with other users of the system.
- 3.5.7 Public Templates must be rated. Each user can rate the Component / Web Page / Application that he/she used.
- 3.5.8 It must be possible to edit Components', Containers', Web Pages' properties, data instances, queries (XPath and XQuery), and CSS.
- 3.5.9 It must be possible to manage access right to Components, Containers, and Web Pages.
- 3.5.10 It must be possible to view or edit the source code of Components, Containers, and Web Pages.

- 3.5.11 The UI must indicate to the user where a Component, Container, or Web Page can be placed in the design view.
- 3.5.12 It must be possible to centrally administer Components (e.g., in a library), which are used in the web Application project.
- 3.5.13 It must be possible to centrally administer Web Pages and Applications in an account view.
- 3.5.14 It must be possible to create a copy of existing Application or Web Page and then modify this copy and use it as a template.
- 3.5.15 When it is possible Data Instances must be automatically updated after adding, moving, and deleting the Component.
- 3.5.16 It must be possible to create new Components, Web Pages, and Applications by directly uploading the source code.
- 3.5.17 User must be able to start/stop/restart publishing, upload new version, and download code of the Application.

Finally the summary table was created. Table 1 shows different elements of the Template Language and what they consist of.

Table 1 XFormsDB Template Language summary

	Component	Container	Web Page	Application
Contain				
Head	(X)	(X)	X	
Body	(X)	(X)	X	
Typed parameter	X		X	
Hard-coded XML (body)	X		X	
Container (body)	X	X	X	
Data Instance (head)	X	X	X	
User Interface (body)	X	X	X	
CSS (head)	X	X	X	
Query (head)	X	X	X	
Access rights (head&body)	X	X	X	
Component call (body)		X		
Metadata				
id	X			
title	X	X	X	X
description	X	X	X	X
tags	X			
Feature				
Reusable	X		(X)	(X)
Independent/Self-sufficient	X	X	X	X
Dependent/Insufficient	X	X		

Convertible to a component		X	X	
URL			X	X

3.6 The structure of the files

In scope of Template Language design, the task of folder structure design appeared. Since applications and templates should be stored on a server the structure of folders and files should be defined.

During the design it was found that applications and templates could have files of different types and purpose. These files could be assigned to different parts of the application (e.g., to page). There could be related references to the files in the source code. There are several processes performed with the application/template, which influence on the file structure design:

- Adding template into the page
When template is added its files should be placed into corresponding place in the application folder structure. Related reference to the template's files should remain valid.
- Publishing the application
When the transformation from Template Language to the XFormsDB application is performed, these links should stay valid or be changed according to the new file positions.
- Saving a page (or part of the page) as a new template
The same problem as in previous case occurs.

After analyzing these problems 2 options of file structure were proposed.

Option 1: Folder structure based on file type and file owner

Each file type (e.g., css, resources, db, etc.) has its own folder in the hierarchy. In these folders files are structured according to their owner.

```

web_application_name
|-- resource
|   |-- component_name (hash, unique within components)
|   |   |-- *.jpg, *.png, *.js
|   |-- index (unique within application)
|   |   |-- *.jpg, *.png, *.js
|   |-- *.jpg, *.png, *.js
|-- css
|-- data
|-- query
|-- db
|-- index.xformsdb, *.xformsdb

```

Option 2: Folder structure based on file type only

Each file type (e.g., css, resources, db, etc.) has its own folder in the hierarchy. Files are not structured inside these folders.

```

web_application_name
|-- resource
|   |-- *.jpg, *.png, *.js (all files of these types)
|-- css
|-- data
|-- query
|-- db
|-- index.xformsdb, *.xformsdb

```

Pros and Cons of both approaches are listed in the Table 1.

Table 2 Different folder structures: Pros & Cons

Option 1: file grouped by type and owner	
Pros	Cons
Nice file structure. Useful when editing template/application by hands	Needs reference update when page is saved as new template
Information about resource owner (for the application and pages) can be received from folder structure	
Does not require reference updates when publishing application. Can be used now without any extra coding, since saving-as-new-component feature hasn't developed yet	
Reference update is a machine problem - does not require user's effort (xml and css parser)	
At the beginning reference update can be done in a simple way (like find-and-replace)	
Option 2: file grouped by type only	
Pros	Cons
Does not require reference update	Unique file names are not useful for a user (long and non-logical)
Does not have additional level of hierarchy in the reference	All files of the same type are stored in a mess
	Resource owner management is done in the UI based on information stored in DB. When user downloads file structure, he/she does not receive this information (e.g., doesn't know what page is the owner of the image)

Finally option 2 was selected. Figure 2 shows detailed structure of the application according to idea of option 2. The application has 1 web page called *index*, which has one container *container_id_1* defined. One simple component called *component_1* is placed into the container.

```

webapp_1
|-- resource
|   |-- component_1 (hash, unique within components)
|   |   |-- *.jpg, *.png, *.js
|   |-- container_id_1 ( unique within application)
|   |   |-- *.jpg, *.png, *.js
|   |-- index
|   |   |-- *.jpg, *.png, *.js
|   |-- *.jpg, *.png, *.js
|-- css
|   |-- component_1 (hash, unique within components)
|   |   |-- *.css
|   |-- container_id_1 (unique within application)
|   |   |-- *.css
|   |-- index
|   |   |-- *.css
|   |-- *.css
|-- data
|   |-- component_1 (hash, unique within components)
|   |   |-- *.xml
|   |-- container_id_1 (unique within application)
|   |-- index
|   |   |-- *.xml
|   |-- *.xml
|-- query
|   |-- component_1 (hash, unique within components)
|   |   |-- *.xq, *.xpath
|   |-- container_id_1 (unique within application)
|   |   |-- *.xq, *.xpath
|   |-- index
|   |   |-- *.xq, *.xpath
|   |-- *.xq, *.xpath
|-- db
|   |-- component_1 (hash, unique within components)
|   |   |-- *.xml
|   |-- container_id_1 (unique within application)
|   |   |-- *.xml
|   |-- index
|   |   |-- *.xml
|   |-- *.xml

```

```
|-- index.xformsdb, *.xformsdb
```

Figure 2 Folder structure of the application

Figure 3 shows detailed structure of the template according to idea of option 2.

```
component_1
|-- resource
|   |-- component _1
|   |   |-- *.jpg, *.png, *.js
|-- css
|   |-- component _1
|   |   |-- *.css
|-- data
|   |-- component _1
|   |   |-- *.xml
|-- query
|   |-- component _1
|   |   |-- *.xq, *.xpath
|-- db
|   |-- template_1
|   |   |-- *.xml
|-- component _1.xformsdb
```

Figure 3 Folder structure of the template

3.7 The Template markup language

Here is example of the application and template based on current version of the Template Language. The application provides rss-reader functionality. The application consists of one web page, which has only one slot. One sufficient component is added to this slot, which produces all application functionality. It reads rss news from the URL given as a parameter and displays them. Source code of the web page can be found from Figure 4. Source code of the component can be found from Figure 5.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html SYSTEM "xformsdb1.dtd">
<template:webpage xml:lang="en" lang="en"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:exforms="http://www.exforms.org/exf/1-0"
  xmlns:template="http://www.tm1.tkk.fi/2009/template"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xforms="http://www.w3.org/2002/xforms"
  xmlns:xformsdb="http://www.tm1.tkk.fi/2007/xformsdb"
  xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
  <template:meta name="title">Web Page: News Widget</template:meta>
  <template:meta name="description">Web page for news
widget.</template:meta>
  <template:param name="pRSSFeedURL "
```

```

select="'http://rss.cnn.com/rss/edition.rss'" />
<template:head>
  <title>News Widget</title>
  <meta name="keywords" content="News, Widget" />
  <meta name="description" content="News Widget" />
  <meta name="robots" content="all" />
  <meta name="author" content="Markku Laine" />
  <meta name="copyright" content="Copyright &copy; Markku Laine
2009. All rights reserved." />
  <link rel="stylesheet" type="text/css" href="css/reset.css"
media="all" />
  <link rel="stylesheet" type="text/css" href="css/base.css"
media="all" />
  <link rel="stylesheet" type="text/css" href="css/index/base.css"
media="all" />
  <xforms:model />
</template:head>
<template:body>
  <div id="index_-_page_margins">
    <div id="index_-_page">
      <template:container>
        <template:meta name="title">Container:
Main</template:meta>
        <template:meta name="description">Container for
components to be placed on the page.</template:meta>
        <template:head />
        <template:body>
          <template:call-component
name="tpl_news_widget">
            <template:with-param
name="pRSSFeedURL" select="$pRSSFeedURL" />
          </template:call-component>
        </template:body>
      </template:container>
    </div>
  </div>
</template:body>
</template:webpage>

```

Figure 4 Template Language: Web Page source code

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html SYSTEM "xformsdb1.dtd">
<template:component xml:lang="en" lang="en"
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ev="http://www.w3.org/2001/xml-events"
  xmlns:exforms="http://www.exforms.org/exf/1-0"
  xmlns:template="http://www.tm1.tkk.fi/2009/template"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xforms="http://www.w3.org/2002/xforms"
  xmlns:xformsdb="http://www.tm1.tkk.fi/2007/xformsdb"
  xmlns:xxforms="http://orbeon.org/oxf/xml/xforms">
  <template:meta name="id">tpl_news_widget</template:meta>

```

```

        <template:meta name="title">Component: News Widget</template:meta>
        <template:meta name="description">Component for news
widget.</template:meta>
        <template:meta name="tags">component public self-sufficient
news_widget</template:meta>
        <template:param name="pRSSFeedURL"
select="'http://rss.cnn.com/rss/edition.rss'" />
        <template:head>
            <link rel="stylesheet" type="text/css"
href="css/tp1_news_widget/base.css" media="all" />
            <xforms:model>
                <!-- XForms instances -->
                <!-- RSS feed instance -->
                <xforms:instance id="tp1_news_widget_-
_rss_feed_instance" src="$pRSSFeedURL">
                    <dummy xmlns="" />
                </xforms:instance>
            </xforms:model>
        </template:head>
        <template:body>
            <div id="tp1_news_widget_-_body">
                <div id="tp1_news_widget_-_header">
                    <h1><xforms:output ref="instance( 'tp1_news_widget_-
_rss_feed_instance' )/channel/title" /></h1>
                </div>
                <div id="tp1_news_widget_-_main">
                    <h3>Top Stories</h3>
                    <xforms:repeat nodeset="instance( 'tp1_news_widget_-
_rss_feed_instance' )/channel/item" id="tp1_news_widget_-
_rss_feed_item_repeat">
                        <div class="news">
                            <div class="title">
                                <xforms:trigger appearance="minimal">
                                    <xforms:label><xforms:output
ref="title" /></xforms:label>
                                <xforms:action ev:event="DOMActivate">
                                    <xforms:load ref="link" show="new"
/>
                                </xforms:action>
                            </xforms:trigger>
                        </div>
                        <div class="date">
                            <xforms:output ref="pubDate" />
                        </div>
                    </div>
                </xforms:repeat>
            </div>
            <div id="tp1_news_widget_-_footer" />
        </div>
    </template:body>
</template:component>

```

Figure 5 Template Language: Component source code

In the source codes listed below different Template Language elements can be found. Here are descriptions of these elements.

The **template:webpage** Element

- Using this element a web page can be defined. The element should consist of **template:meta**, **template:head** and **template:body**, where meta-information and the content of the page are defined.

The **template:component** Element

- Using this element a component can be defined. The element should consist of **template:meta**, **template:head** and **template:body**, where meta-information and the content of the component are defined.

The **template:head** Element

- Using this element the head part of the component, container or web page can be defined.
- For the web page this element contains information, which is usually placed in the head section of HTML document: information about the web page, such as title, keywords that may be useful to search engines, and other data that is not considered the content.
- For the component, container or web page links to the css and XFormsDB model are defined within this element.

The **template:body** Element

- Using this element the body part of the component, container or web page can be defined.
- For component/web page body part contains it's content.
- Container's body part can contain only **template:call-component** elements.

The **template:meta** Element

- Using this element meta-information of component, container or page can be defined. In the current version of Template Language list of possible meta-information items contains id, title, description and tags.

The **template:param** Element

- Using this element a parameter can be defined for the component or for the Web Page. The definition includes parameters name and default value. When the component is called from the page the value of the parameter can be defined within **template:with-param** element.
- User can define parameters from the XIDE. When user changes parameters value source code of the component is not changed.
- Types for the parameters will be added in the next phase.

The **template:container** Element

- Using this element a container can be defined. Container is a placeholder for components.
- The element should consist of **template:meta**, **template:head** and **template:body**.

The **template:call-component** Element

- Using this element a component can be called. Using of this element is only possible within **template:body** element of the container.

The **template:with-param** Element

- Using this element the value of parameter can be transferred into component. Using of this element is only possible within **template:call-component** of the container.

Other elements to be done

There are also several modifications that should be made to the design. One of them is adding the element that defines the API of a component i.e. defines which element (ID) sends what XML Events. For instance, if a component #2 wants to listen to events of a filter input field (of the component #1) it can add a listener to that.

Another one is adding parameter typing.

The biggest thing to be design still is how data instances are defined and how inserted components can be bound to certain data instances.

4 Status

4.1 Templates

The Template Language design is almost ready. Several minor things are still missing but its current version can be used as an intermediate language for XIDE. During the process of publishing web application developed in XIDE a transformation from Template Language into XFormsDB is required. Implementation of this transformation is now in progress.

4.2 Transactions

The design of the XFormsDB transactions was initially planned to take place in Q2/2008. However, the work is pushed to the second project year due to effort spent on supporting Flexible Services project management Web application.

5 Summary

The purpose of this deliverable was to design an XFormsDB Language for templates and transactions.

Template Language should be used as intermediate language for XIDE and allows defining and managing reusable parameterized components and web pages. Its design is almost finished so it can be used in XIDE.

The work related to the XFormsDB Language for Transactions has been pushed to the second year.