Project Number:	**215219**
Project Acronym:	**SOA4All**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D2.6.1 Specification of the SOA4All Process Editor

Activity:	Activity 1 – Fundamental and Integration Activities
Work Package:	WP2 – SOA4All Studio
Due Date:	M12
Submission Date:	28/02/2009
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organization Responsible of Deliverable:	SAP
Revision:	1.0
Author(s):	Ivan Delchev SAP Juergen Vogel SAP Dr. Sven Abels TIE Shiva Puram TIE

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

Dissemination Level		
PU	Public	X

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	21.01.09	Initial version, ToC defined	Ivan Delchev (SAP), Juergen Vogel (SAP)
0.2	27.01.09	Functional requirements defined	Ivan Delchev (SAP)
0.3	28.01.09	Requirements from Integration with SOA4All architecture	Sven Abels (TIE)
0.4	28.01.09	State-of-the-Art in Process Modeling	Shiva Puram (TIE)
0.5	02.02.09	Architecture Description added	Ivan Delchev (SAP)
0.6	03.02.09	Use-cases added	Ivan Delchev (SAP)
0.7	09.02.09	Polishing of sections, executive summary	Ivan Delchev (SAP)
		Internal Review	Adrain Mos (INRIA), Stuart Campbell (TIE)
0.8	24.02.09	Changes due to internal reviewers' comments	Ivan Delchev (SAP), Sven Abels (TIE)
1.0	27.02.09	Final version	Ivan Delchev (SAP), Sven Abels (TIE)
1.0	28.02.09	Overall quality format revision	Malena Donato (ATOS)

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

Acronym	Definition
AJAX	Asynchronous JavaScript and XML
CRUD	Create / Read / Update / Delete
D	Deliverable
EC	European Commission
EXT GWT	Extended GWT (Rich Internet Application Framework for GWT)
GWT	Google Web Toolkit
JSON	JavaScript Object Notation
MVC	Model-View-Controller
RIA	Rich Internet Application
UI	User Interface
WP	Work Package
WSDL	Web Service Description Language
WSMO	Web Service Modeling Ontology
XPDL	XML Process Definition Language

Executive Summary

The SOA4All Process Editor (short: Composer), which is the intended output of Task 2.6, will be a Rich Internet Application (RIA) for process modeling, built especially for the custom light-weight modeling language of SOA4All. The Composer will be tightly integrated in the SOA4All Studio architecture as a plug-in and will employ the same look and feel.

The Composer will allow non-technical users to compose light-weight processes from semantically annotated web services. The user will be able to perform common modeling activities such as creating, editing and deleting processes as well as rating, commenting, collaborating on and organizing them. Context-aware wizards will be available to guide the user during service composition. The process models will be stored in the modeling language format and will be executable in the SOA4All infrastructure.

What differentiates the SOA4All Composer in comparison with other modelling tools is the Web 2.0 approach – the use of the web as a platform and making the most of its intrinsic advantages and the utilization of semantic and contextual information. By allowing the consumption, re-mixing and composition of services and information from multiple sources, the SOA4All Composer will create network effects through an “architecture of participation” and the SOA4All ecosystem will become better as more people use it.

1. Introduction

1.1 Purpose and Scope of this Deliverable

The purpose of Deliverable 2.6.1 is to identify the requirements, define the functionality, and provide an architecture design of the SOA4All Process Composer (short: SOA4All Composer). The Composer will be integrated in the SOA4All Studio as a plug-in component and provide the complete functionality and expressive power of the light-weight modeling language available to end users in a graphical manner. Moreover, the SOA4All Composer will be positioned in the context of the SOA4All Studio and in the complete SOA4All architecture by identifying relations to the rest of the SOA4All components.

1.2 Relations to Other Parts of SOA4All

As part of WP2, Task 2.6 and specifically Deliverable 2.6.1 "Specification of the SOA4All Process Composer" are naturally connected to WP2 tasks as follows:

- Task 2.1 provides the functionalities and methodologies that service prosumers (producers/consumers) require in order to provide new services or to enrich the existing ones. It will permit them to model and annotate services easily. The SOA4All Composer will use services provided from Task 2.1 functionality and model them into service compositions. Semantic descriptions and annotations will be used to ease service composition modeling.
- Task 2.2 deals with service discovery and the use of contextual factors in order to provide users with the most relevant results. It envisions formal goals defined in WSMO, abstract services, keyword discovery etc. Furthermore, it envisions the use of wizards that will support users by transforming user requirements into formal and structured goals. The SOA4All Composer will provide components that support and make use of the above-listed functionality such as including abstract goals as parts of service compositions. Results from discovery will be immediately ready for drag & drop inclusion in process models.
- Task 2.3 will deliver a Monitoring and Management Tool Suite for the monitoring and management of services and will provide support for the interpretation of provenance information. The SOA4All Composer will include monitoring information such as QoS during design time in order to give additional information to users.
- Task 2.4. The SOA4All Composer is related to Task 2.4, which provides components and functionality to ease the development and the integration of tasks that have a need for user-interactions. Task 2.4 will deliver an UI and an infrastructure framework that will make it easier to, for example, create web 2.0 user interfaces, to store and manage data in the SOA4All environment and to use communication functionalities. This SOA4All user gateway, the so-called "SOA4All Studio" will serve as a connection point for T2.X functionality. UI components and related functions include drag & drop, which will be reused by 2.6 to actually implement the SOA4All Composer UI allowing people to drag & drop process elements. In addition to this, a Dashboard view will seamlessly integrate the Process Composer UI into the overall SOA4All Studio environment. A Graph Visualization Widget will be used by the Process Composer to visualize a process model on screen.
- Task 2.5 will evaluate the user interfaces developed in this work package and as part of this will assess the Process Composer UI. The usability evaluation will serve as feedback for consecutive development activities.
- Task 2.7. Finding and composing services from the billions of envisioned services requires sophisticated recommendation mechanisms. Task 2.7 will support users by

recommending services throughout their interaction with the SOA4All Studio. Contextual information from users' interactions and social network will enhance the suggestion mechanisms. The SOA4All Composer will interface with these functionalities in order to provide users with most-relevant recommendations.

In addition to tasks positioned in the same WP as the SOA4All Process Composer, tasks in WP6 also exhibit a close relation to Task 2.6 as follows:.

- Task 6.3. The Process Composer will provide the complete functionality and expressive power of the light-weight context-aware process modeling language of Task 6.3 available to end users in a graphical fashion. The stored process models will be represented as editable and executable process descriptions. This relationship is pivotal - the Composer must expose the modeling language in an intuitive and easy-to-use manner to its users.
- Task 6.4. In addition, the functionality of the Service Adapter of Task 6.4 will be used for service context-based adaptation at design-time. The purpose of this tool is the adaptation of services according to context (e.g. personal preferences, business rules, etc.) and advanced mechanisms such as incremental revealing of service descriptions involving trust and a negotiation process between parties.

1.3 Innovation in Process Editing

The SOA4All Composer will include many innovative features and technical advances into a user-friendly service composition tool. Distinct features include:

- RIA architecture with desktop-like responsiveness
- Intuitive and user-context-aware modeling User Interface (UI)
- Modeling support via wizards and process templates
- Use of abstract goals as service composition building blocks, not only concrete services
- Generation of executable service compositions, runnable in the SOA4All environment
- Based on web principles and technology thus promoting scalability, openness for extension and reuse
- Employs Web 2.0 concepts such as tagging, sharing, user collaboration, commenting, ranking etc. to bring the value of collective intelligence, also called "wisdom of the crowds"
- Considers semantic annotations and context information
- Specifically designed to work with the SOA4All light-weight modelling language

1.4 Structure of this Deliverable

This deliverable is organized in the following way:

- Section 1 gives an introduction to the scope and content of this deliverable
- Section 2 investigates in detail the current state-of-the-art in the process modeling area
- Section 3 elaborates the requirements for the SOA4All Composer and their origin.
- Section 4 gives the functional specification of the Composer

- Section 5 covers the foreseen architecture of the tool, its relation to other SOA4All components and the potential use cases.
- Finally the Conclusions section summarizes the deliverable, presents the drawn conclusions and suggests next steps

2. State-of-the-Art for Process Editors

Before designing the SOA4All Process Editor, an analysis is given for giving an overview about the current state of the art in process composition. This overview is broken down into two essential parts: Firstly, a list of web based process editors is presented and secondly a list of desktop based products.

Obviously, the SOA4All process editor will be a web based application in order to stay consistent with the overall SOA4All principles. However, nevertheless it is important to also briefly look at desktop based solutions as they could contain a large number of approaches and ideas that 2.6 could benefit from.

2.1 Web-Based Process Composers

This section will give an overview about existing tools in the market that may be used for process composition. The following subsections will start with a short description and a list of main features. Each section will then give a comparison to the SOA4All vision and will highlight specific elements that SOA4All could learn from this product (if appropriate). Finally, an overview table for each tool is given to quickly summarize the main facts.

2.1.1 Lombardi Blueprint

2.1.1.1 Description

The Lombardi Blueprint was initially released in April 2007 and is completely based on web based modeling. From a technical perspective it uses Ajax and the Google Web Toolkit to provide a seamless interface for process modeling. It has been developed to help users that have little business process modeling experience construct high-level diagrams during the discovery phases of a strategic planning cycle. Lombardi supports Business Process Management Notation (BPMN). The latest release of Blueprint also includes a Visio importer and can import a variety of drawings. Visio drawings have no underlying semantics, but nevertheless drawings created using the BPMN stencil can be imported to Blueprint as BPMN diagrams.

High-level capabilities of Lombardi Blueprint

- On-demand browser-based access, delivered in a “Software as a Service” (SaaS) format. Blueprint is compatible with Windows Internet Explorer and Mozilla Firefox. Further there's a round-trip integration with Lombardi TeamWorks.
- Interactive process discovery and mapping collaboration with shared workspaces
- Prioritization of problems based on severity and frequency. Further an opportunity ranking is provided that is based on an impact score
- High-level map and detailed workflow modeling (BPMN-based) views of processes. This implies a one-click creation of the project overview presentation and business case.
- Complete history and audit trail for all changes for compliance purposes.

2.1.1.2 Comparison to the SOA4All approach

The process-modeling tool Blueprint uses BPMN as modeling language. From SOA4Alls point of view, the whole set of BPMN elements is not easily understandable for non-modeling-experts. It is envisioned to create a new language that can intuitively be used. Further SOA4All will make use of semantic technologies to structure the content (e.g tagging etc.). SOA4All will include more guidance along the modeling activity based on context or recommendations.

2.1.1.3 Things that SOA4All can learn from this product

Lombardi Blueprint provides an excellent example of an intuitive editing component that is usable by non-experts. Although the editor is still far from perfection (as described above), it still provides a very good example for some aspects such as the easy process definition interface. This product can be seen as an example for an easy-to-use process definition user interface for the SOA4All Composer.

2.1.1.4 Key-Facts

Feature	**Properties**
Organization / Consortium	Lombardi Software Inc.
License	Commercial License
Online / Offline Editor	Online
Modeling Language	Business Process Management Notation (BPMN)
BPEL Export (Yes / No)	Yes
URL	http://www.lombardisoftware.com/bpm-blueprint-features.php

Table 1: Lombardi Blueprint

2.1.2 Oryx

2.1.2.1 Description

Oryx is a web-based editor for modeling business processes based on the BPMN language. Oryx can create BPMN models and share them with other people in a team by publishing the models that have been created. Although Oryx is focused on BPMN, it also provides some capabilities of creating models in other languages such as EPC or the Petri net mark-up language. It also allows extending the existing styles, which allows users to add new modeling languages [Oryx].

High-level capabilities of Oryx:

- Web-based editor for modeling business processes in BPMN providing on-demand browser-based access, delivered in a Software-as–a-Service (SaaS) style.
- Support of OpenID for identifying users. OpenID is required in order to save models that have been created.
- SVG support is needed for editing functionality
- New functions may be added via a plug-in mechanism
- Different languages are supported out of the box: BPNM, EPC, PNML

2.1.2.2 Comparison to the SOA4All approach

Unlike SOA4All, Oryx does not aim in integrating any semantics or in integrating the results into existing services. As such it is not capable of connecting existing web services. Instead of this, it is focused on the graphical modeling process itself. In this sense, Oryx is similar to a graphics editor. Currently Oryx does not support export of information into a standard business process modeling language such as BPEL. However, according to the documentation of the project, developers are currently working on this functionality. Oryx itself is a set of JavaScript routines loaded into a web browser as part of a single document describing the whole model where Models are represented in RDF format, where as SOA4All is a RIA based on Web 2.0.

2.1.2.3 Things that SOA4All can learn from this product

Oryx provides a very flexible editor that allows a realization of very flexible models. However, this flexibility also makes it appear more like a graphic tool than a regular business process editor. The support of OpenID and the possibility of exchanging models with other persons is useful and should be considered by SOA4All as well.

2.1.2.4 Key – Facts

Feature	**Properties**
Organization / Consortium	Hasso-Plattner-Institute
License	Open Source
Online / Offline Editor	Online
Modeling Language	BPMN , EPC
BPEL Export (Yes / No)	No
URL	http://bpt.hpi.uni-potsdam.de/Oryx

Table 2: Oryx

2.1.3 Appian Anywhere

2.1.3.1 Description

Appian Anywhere is a browser based web application for modeling business processes. It is built around the BPMN standard and allows users to add additional graphical elements to the model such as pictures or photos. It allows users to group processes into different areas. For example, a “support request”-process may be grouped into one set of process steps for the support manager and one set of processes for the support ticket agent. Appian Anywhere is available as a Software-as-a-Service (SaaS) model. [Appian]

High-level capabilities of Appian:

- On-demand browser-based access, delivered in a Software-as-a Service (SaaS) format
- Draw diagrams that instantly run as applications
- Application dashboards track processes and put documents at your fingertips and manage work with custom screens that fit your style
- Get real time analytics on process and business data with drill-down investigation.
- Manage all your documents forms and training materials in the document repository

2.1.3.2 Comparison to the SOA4All approach

Appian Anywhere is much more technical than Lombardi Blueprint or Oryx. Its focus is on the technical side of the process construction. In comparison to the SOA4All functionality, Appian Anywhere does not support the integration of semantics and does not support sharing information in a team. According to Appian, the integration of external processes is possible although this has not been tested by the T6.3 team.

2.1.3.3 Things that SOA4All can learn from this product

The grouping functionality of Appian helps users keep an overview about processes and about the topics that are covered by those processes. It is therefore recommended to include a similar functionality into the SOA4All Composer.

2.1.3.4 Key – Facts

Feature	**Properties**
Organization / Consortium	Appian
License	Proprietary License
Online / Offline Editor	Online
Modeling Language	BPMN
BPEL Export (Yes / No)	Yes
URL	http://www.appian.com/product/anywhere.jsp

Table 3: Appian Anywhere

2.2 Desktop-Based Process Editors

This section describes several desktop based products in the area of process composition. There are plenty of tools available in the market today but due to length limitations of this deliverable, this section will focus on a selection of well-known tools in this domain.

The section uses the same structure as the web-based section.

2.2.1.1 Soyatec – eBPMNDescription

As a member of the Eclipse Foundation, Soyatec has based their product on the Eclipse Rich Client Platform. As such, eBPMN is available as a desktop application only. The product itself is therefore fully integrated into Eclipse and may be combined with other plugins. As stated by the name, the eBPMN Designer is based on the BPMN language. It mainly targets developers and requires at least a small amount of knowledge on how to use the Eclipse IDE in order to create new models and projects [Soyatec].

High-level capabilities of eBPMN:

- Full object model of the BPMN 1.0 specification
- Tightly integrated into Eclipse
- Developer oriented

2.2.1.2 Comparison to the SOA4All approach

In comparison to the SOA4All approach, eBPMN has a different target group since it does not aim in providing a modeling approach for everyone, nor does it aim at sharing process models with other people. eBPMN is mainly targeting technical people that want to create a business model for reflecting their IT processes. It is a purely desktop application.

2.2.1.3 Things that SOA4All can learn from this product

eBPMN allows users to create many different processes and allows to add time specific constraints. Although those are useful features, they are not crucial for SOA4All and can rather considered to be 'nice to have' functionalities.

2.2.1.4 Key – Facts

Feature	**Properties**
Organization / Consortium	Soyatec
License	Free and Commercial Versions
Online / Offline Editor	Offline
Modeling Language	BPMN
BPEL Export (Yes / No)	Yes
URL	http://www.soyatec.com/ebpmn/features.html

Table 4: Soyatec eBPMN

2.2.2 Eclipse BPEL

2.2.2.1 Description

The Eclipse BPEL project aims in creating a graphical BPEL designer for modeling business processed within the Eclipse RCP framework. As such, the BPEL project follows a similar approach to eBPNM. The project is currently in an incubation stage meaning that it is under strong development and ongoing change [EclipseBPEL].

High-level capabilities of Eclipse BPEL:

- Creation of GEF & EMF based diagrams that are compliant to the BPEL specification (after exporting)
- Grouping processes and defining conditions
- Providing a runtime framework for BPEL deployment into several BPEL engines
- Debugging BPEL processes with a step-by-step debugging approach

2.2.2.2 Comparison to the SOA4All approach

Similar to eBPNM, the Eclipse BPEL project mainly targets developers and technical experts. The user interface tries to make the process generation simple and provides a compact user interface. However, the process modeling is far from being usable by non-technical people and it does not allow sharing processes or invoking services from a service directory.

2.2.2.3 Things that SOA4All can learn from this product

The idea of providing an integrated step-by-step debugger helps process designers to validate the process functionality and to quickly detect problems in the process. As such SOA4All would highly benefit from a similar debugging functionality.

2.2.2.4 Key – Facts

Feature	**Properties**
Organization / Consortium	Eclipse
License	Open Source
Online / Offline Editor	Offline
Modeling Language	BPEL
BPEL Export (Yes / No)	Yes
URL	http://www.eclipse.org/bpel/

Table 5: Eclipse BPEL

2.2.3 Oracle BPEL Process Manager

2.2.3.1 Description

The Oracle BPEL Process Manager provides a developer-friendly and reliable solution for designing, deploying and managing BPEL business processes. It has been one of the first well-known and highly distributed process editors in the market and is available as a stand-alone application, although being implemented on top of Eclipse.

The Oracle BPEL Process Manager contains more than a designer of business processes. It also contains a BPEL execution engine and a sophisticated integration of external web services. A large set of documentation and examples allows developers to define business processes or to connect different web services into processes easily. [OracleBPEL]

High-level capabilities of Eclipse BPEL:

- Creation of BPEL processes using a graphical editor
- BPEL execution engine
- Real-world examples and detailed documentation
- Highly scalable solution, widely applied in different information systems

2.2.3.2 Comparison to the SOA4All approach

In comparison to the SOA4All approach, the Oracle BPEL Process Manager does not consider an integration of semantic information. In addition to this, the usage of the Process Manager required some technical knowledge about business process design and is not suitable for non-experts.

The BPEL Process Manager is a very powerful and rather heavyweight application, allowing the realization of almost all business processes for experts. In comparison to this, SOA4All will focus on non-experts by providing a lightweight process editor.

2.2.3.3 *Things that SOA4All can learn from this product*

The documentation and provision of examples is outstanding in the Oracle BPEL Process Manager. It helps people get started quickly with a complex product. It is therefore required to create a similar set of examples for SOA4All users showing them how to create a simple business process within just a few steps.

2.2.3.4 *Key – Facts*

Feature	Properties
Organization / Consortium	Oracle
License	Proprietary License
Online / Offline Editor	Offline
Modeling Language	BPMN
BPEL Export (Yes / No)	Yes
URL	http://www.oracle.com/technology/bpel

Table 6: Oracle BPEL Process Manager

2.2.4 SAP Netweaver

2.2.4.1 *Description*

NetWeaver BPM supports a model-driven approach to managing business processes throughout their lifecycle. It provides an integrated design and runtime environment that enhances collaboration between business and IT through shared BPMN models, providing descriptive context for both implementation design and performance monitoring. Business-IT alignment and agility are also enhanced by close integration between BPM and business rules throughout the lifecycle as well: modeling, execution, and management.

While focused on edge processes, NetWeaver BPM can also be used to compose core application processes, such as local variations within a global enterprise. With its close integration to SAP's Enterprise Service Repository and existing enterprise services, SAP is trying to evolve NetWeaver BPM into the BPMS of choice for SAP Business Suite customers [SAP Netweaver].

High-level capabilities of SAP Netweaver include:

- Creation and debugging of executable business process models.
- Each business process model clearly defines the rules and exceptions governing the process steps
- The process editor provides graphical modeling of activity flows using BPMN, with implementation properties of each selected node in the diagram defined via point-click property selection.
- NetWeaver BPM supports message flows and intermediate events, including attached error events.
- BPMN allows event- and exception-handling to be described explicitly in the process diagram.

- The process editor distinguishes human tasks from automated services.
- BPMN diagrams created provides visual context for process monitoring at runtime.

2.2.4.2 Comparison to the SOA4All approach

In comparison to the SOA4All approach, SAP NetWeaver BPM requires technical knowledge about business process modeling and execution and is thus not suitable for non-experts. Furthermore, SAP NetWeaver BPM does not make use of semantic technologies in order to support the discovery of services. In comparison to this, SOA4All will focus on non-experts by providing a lightweight process editor

2.2.4.3 Things that SOA4All can learn from this product

SAP NetWeaver BPM does not only concentrate on services but also considers human tasks as an important artifact in process modeling. It is desired to use a similar approach in the SOA4All Composer.

2.2.4.4 Key – Facts

Feature	**Properties**
Organization / Consortium	SAP
License	Proprietary License
Online / Offline Editor	Offline Editor
Modeling Language	BPMN
BPEL Export (Yes / No)	Yes
URL	http://www.sap.com/platform/netweaver/components/sapnetweaverbpm/index.epx

Table 7: SAP Netweaver

3. Requirements

This section defines the functional and non-functional requirements of the SOA4All Composer. For achieving a good overview, it is split into the following subsections:

- use case overview – a description of typical tasks of the Composer
- domain-oriented design requirements
- requirements derived from the lightweight nature
- user interface requirements
- requirements for achieving a SOA4All Studio integration

3.1 Use Case Overview

Two pivotal use cases have been identified in the context of the SOA4All Composer – “process model storage and retrieval” and “process modelling”. The emphasis is on the interactions of the involved functionality.

For clarity reasons some functionalities have been omitted and some external functionalities have been included.

3.1.1 Storage and Retrieval of Process Models

Creation, deletion, saving and loading of processes is naturally a pivotal use case for the SOA4All Composer. It involves interactions with external components and functionality. Figure 1 shows this use case modelled in UML notation.

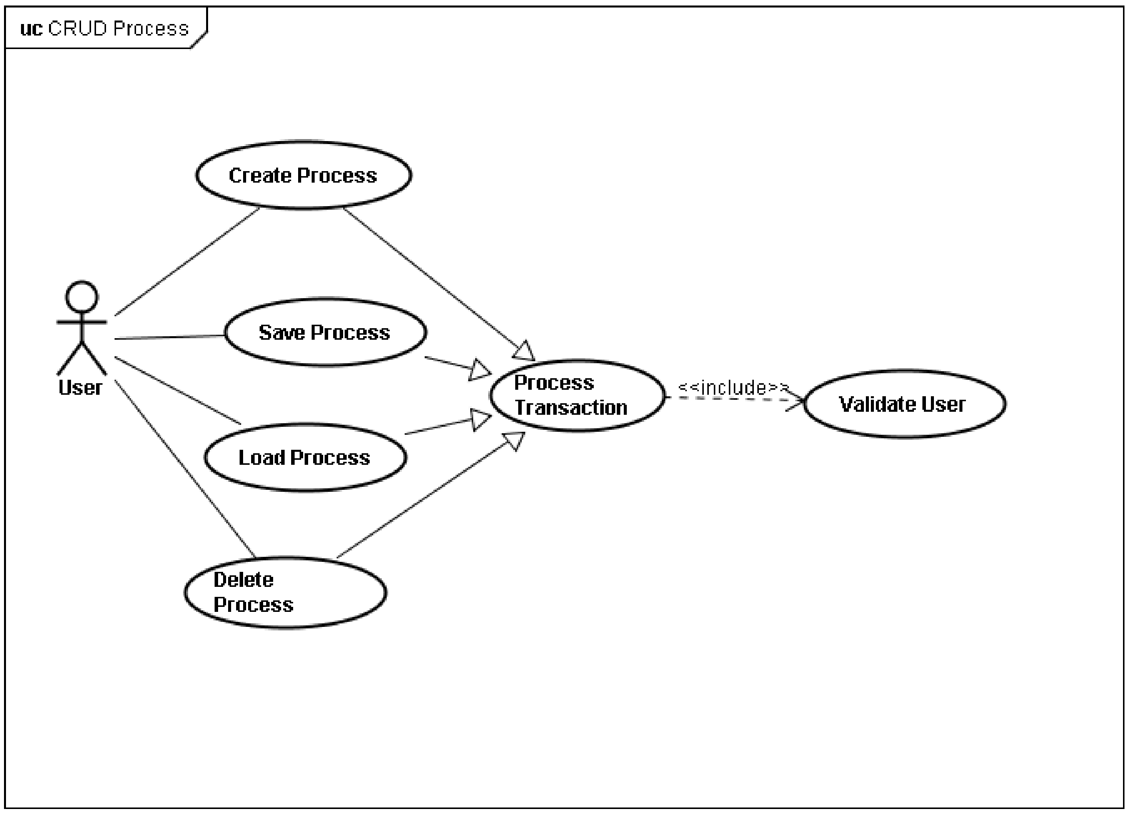


Figure 1 Use Case: Storage and Retrieval of Process Models

3.1.2 Process Modeling

Modeling processes involves choosing suitable services or defining abstract goals in a variety of ways, configuration and composition through connections with configurable semantics and using patterns, templates or wizards. Figure 2 shows this use case modelled in UML notation.

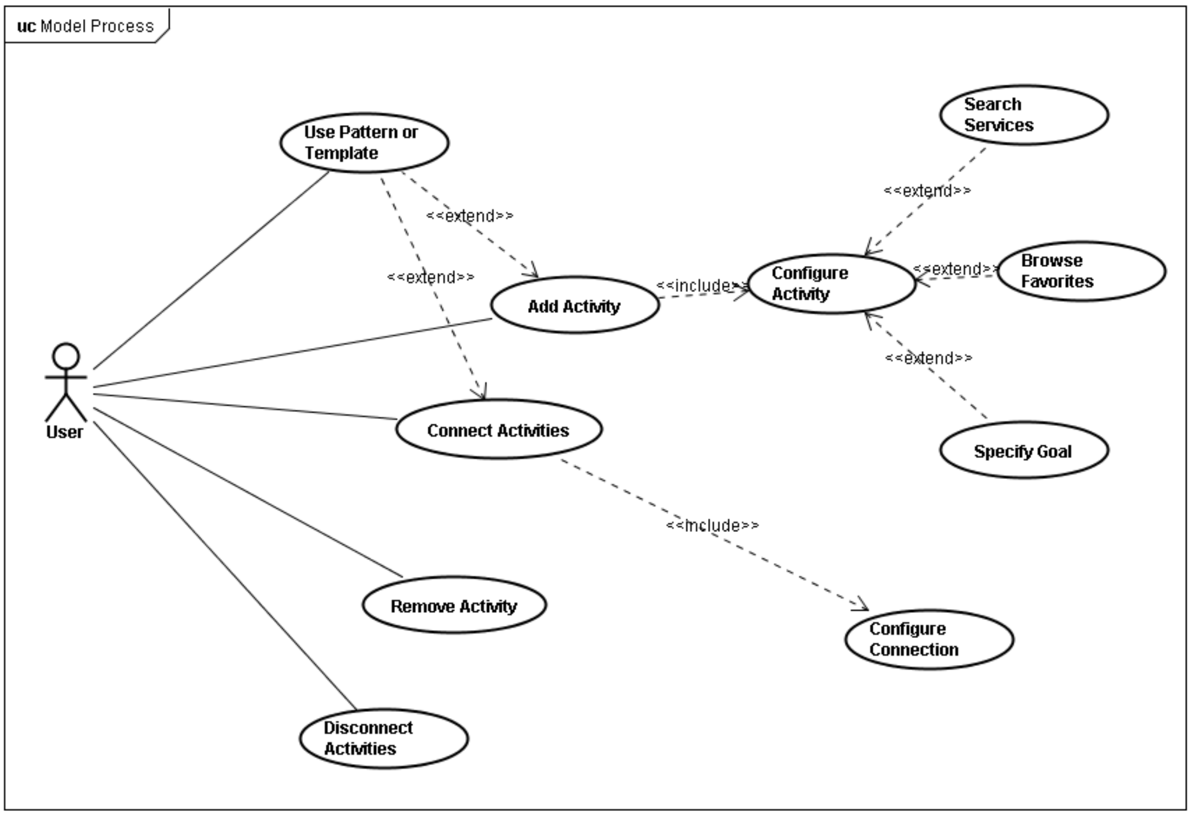


Figure 2 Use Case: Modeling of Processes

3.2 Domain-Oriented Design Requirements

The target group of the SOA4All Composer is users that have a solid idea on the processes that they want to model but that are not necessary experts in this domain and that are also not necessarily skilled in software development or IT-modeling systems. They can therefore be considered to be domain experts.

As such the task of defining service compositions can be regarded as a generic end user development, or design problem. The construction of appropriate representation and interaction mechanisms is a typical meta-design problem [Fischer04], where designers are creating an environment for other people to design solutions, in this case service composition problems.

In literature, most experts of such design environments suggest the use of representations, which are aligned with the domain tasks of their users. They also suggest to provide libraries of problem solution templates and to offer advisory support to the users to reduce the risks of errors. Such environments are known as *Domain-Oriented Design Environments* (DODEs) [Fischer04]. The SOA4All process composer can be seen as a domain oriented software that focuses on the process modeling / process composition domain although it will of course be sector independent (i.e. allowing users to model all types of processes).

Following the design guidelines for Domain-Oriented Design Environments, it suggests an interactive service composition model, where the following requirements are fulfilled:

R01: Search

Users have to be able to search for appropriate services or composition templates – either by textual search or browse, or combination of both

R02: Result

Users should be able to narrow down results of the search according to personal profile characteristics and context parameters

R03: Advisory Functionalities

The environment should support users with advisory and supporting functionality such as attraction of elements to the right slots and repulsion from wrong ones, simplified representation of templates and composite services and goals, filtering of incompatible elements etc.

3.3 Requirements Derived from the Lightweight Nature

One of the main objectives of SOA4All is to make service composition user-friendly. The proposed SOA4All approach (using template-based service composition represented by lightweight process models) seems to fit the class of problems with natural mapping to a diagrammatic representation. However, the complexity of modeling interactions cannot be fully represented by a single, simple representation.

One way forward is to abstract these aspects away and hide them from users. The problem with this approach is that understanding the hidden aspects is often crucial for understanding the overall behavior of the system. Mehandjiev and Bottaci [Mehandjiev96] have proposed the use of assumption descriptions (explicit textual statement of the way missing aspects are implemented by the system) to address this problem, but the mainstream approach seems to rely on choosing appropriate metaphors for the representation in the hope that these evoke the correct common sense assumptions about the missing aspects.

The problem with the latter solution is that some of this “common sense” knowledge seems to depend on the background and other characteristics of the target group of users. We would therefore need to proceed with user profiling techniques and systematic user-aware design of our service composition representations, using formative and summative evaluation techniques as appropriate.

The case studies and outline processes can be linked o create representations tuned to the needs of users and case studies. Achieving the objectives of SOA4All requires the delivery of a service composition interface that is tuned to the skills and tasks of our target users. The long-term aim of SOA4All is to open up service construction to everyone, yet at first instance our target users are those found in the SOA4All case studies (WP7-9).

The case studies are still in stages of initial definition, yet the following characteristics of our end users are now clear:

- Most target users will have a professional background meaning that they have a clear idea of what they want to model without being necessary modeling experts.
- Some target users will not be professional software developers and would not have received significant training in programming nor system design.

These considerations have influenced the lightweight modeling language developed in Task 6.3 significantly and have led to a concise set of constructs, which yet achieve significant modeling expressiveness. For a complete listing of the language constructs, please refer to

D6.3.1.

In addition to this, those considerations have a direct influence on the Composer:

R04: “Common Case Fast”-rule

Because of its lightweight nature, the Composer does not have to support all functions or a fully blown process editor that is used for modeling all possible processes. Instead, a concentration on the most commonly used functions is sufficient.

R05: Template support

The Composer needs to support template-based modelling (see D1.4.1A and D6.4.1).

R06: Non-technical details

The Composer can use expert terms but it should not use technical terms that might not be understood by the domain experts. The actual technical language that is used internally should be hidden from the user in the modeling process.

R07: Support drag & drop

The Compose should support drag & drop operations to allow users to assemble processes using a well-known metaphor.

3.4 User Interface Requirements

The following design principles derived by Johnson et al provide the basis for all SOA4All Studio applications [Johnson00]:

- The focus shall be on the users and their tasks, not the technology
- Function shall be considered first, presentation later
- It shall be conformed to the users' view of the task
- The users' task shall not be complicated
- Learning shall be promoted
- Information shall be delivered, not just data
- It shall be designed for responsiveness
- It shall be tried out on users

Considering the requirements that have been mentioned earlier, the following new requirements can be extracted for the process editor:

R08: Functions first, presentation later

The Composer has to provide all identified functionality while the actual layout may be improved at a later stage of the project. Finalized service compositions must be deadlock-free and executable. Users should be able to reach a correct service composition state at any stage through functionality supported by the Composer.

R09: User-based design

When creating the user interface, the target users should be kept in mind and given more importance than the general SOA4All UI guidelines. Context-based support according to personal profile characteristics and context parameters should be available.

R10: Fast reaction

The process composer needs to have near-instant responsiveness. .

R11: User validation

The process composer UI should be created in cooperation with users and should undergo repetitive user validations

3.5 Requirements for Achieving an SOA4All Studio Integration

The SOA4All Studio will provide a unified interface for all SOA4All users, bundling different tools including the Composer. In addition to a general framework and UI design, the SOA4All Studio provides a set of components that may be used by other tasks such as 2.6 in order to help them creating user interfaces and in order to keep a holistic look & feel.

In terms of the T2.6 integration into the SOA4All Studio, the following elements should be considered:

- Communication
- Storage
- Management
- Integration into the SOA4All Studio Dashboard
- Application of the generic SOA4All Studio design
- Reuse of the SOA4All UI library (i.e. UI widgets)

3.5.1 Storage

All storage and loading activities that will be performed in 2.6 will be based on the semantic spaces provided by WP1 (see D1.3.1 and D1.4.1A). Task 2.4 will provide a simplified interface for loading and storing data. This interface will provide four methods that can easily be invoked by 2.6 when storing and loading process files: storeRDF, queryRDF, storeFile, queryFile. A detailed specification can be found in Deliverable 2.4.1.

R12: Semantic Spaces usage

All storage processes should be performed using the 2.4 interface for storing elements in the SOA4All semantic space.

3.5.2 Management

The SOA4All Studio will provide a management interface as described in D2.4.1. This management interface allows the authentication of users and the management of user profiles. This will be used by the Composer when checking process owners and when editing existing processes. The interface that may be used is described in D2.4.1, Section 5.2. It requires calling the corresponding methods with either WebService or RESTful calls from within the Composer.

R13: User Profile Management

User profiles and management activities should be performed using the infrastructure management services developed in T2.4 (see D2.4.1).

3.5.3 Integration into the SOA4All Studio Dashboard

The SOA4All Studio provides a dashboard view of all elements. This Dashboard is the starting place for new SOA4All users. It shows them all parts and allows them to quickly jump to the different components. For example, the user will see the Composer as a link in the

Dashboard menu and on the Dashboard starting page. Clicking this link will directly lead to the Composer.

R14: Dashboard Integration

The Composer has to reuse the Dashboard plugin interface and to integrate into the SOA4All Dashboard.

3.5.4 Application of the Generic SOA4All Studio Design

After registering as a plugin, the Dashboard will automatically add entries to the menus and SOA4All Studio bars that will allow users to invoke the Composer. The following figure has been taken from D2.4.1 and shows how the Dashboard will integrate the Composer plugin and which parts may be controlled by the plugin.

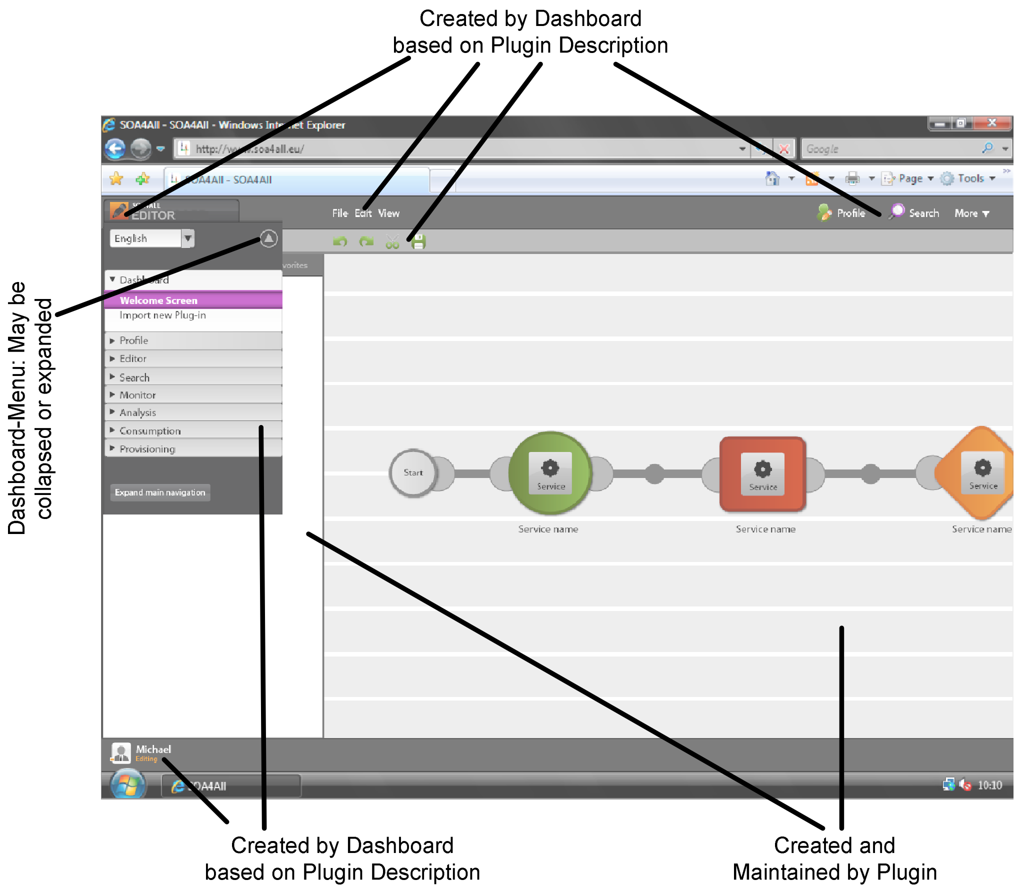

Created by Dashboard based on Plugin Description
Dashboard-Menu: May be collapsed or expanded
SOA4All - SOA4All - Windows Internet Explorer
EDITOR
File Edit View
Profile
Search
More
English
Dashboard
Welcome Screen
Import new Plug-in
Profile
Editor
Search
Monitor
Analysis
Consumption
Provisioning
Expand main navigation
Start
Service
Service name
Michael
Created by Dashboard based on Plugin Description
Created and Maintained by Plugin

Figure 3: Plugin Integration

The Dashboard will provide style sheets and icons that may be used by the Composer in order to create a holistic look & feel. In addition to this, many elements will automatically be provided by the Dashboard using the underlying GWT and Ext GWT frameworks.

R15: Holistic design adoption

The Composer should use a design that adopts to the Design Templates that are specified in deliverable 2.4.1.

3.5.5 Reuse of the SOA4All UI library (i.e. UI widgets)

Another element that will be provided by task 2.4 of the SOA4All Studio is an extensive UI library. This library consists of several UI widgets that may be used by other tasks and work packages such as a widget for user ratings. All widgets have been described and defined in details in D2.4.1.

However, those widgets that are most important for the SOA4All Composer are the graph visualization widget and the drawing widget. Those widgets allow the Composer to create flexible elements with specific shapes and to place them on a panel. Those elements will therefore be the base for the Composer in order to create process diagrams that are follow the graphical design described below.

R16: Component Reuse

The Composer should use the UI components and widgets defined in 2.4. This will allow the Composer to stay consistent to other SOA4All developments in terms of the elements that are used within to UI.

3.6 Summary of Requirements

The following table summarizes the requirements and also assigns a priority of either must, should or nice-to-have:

Requirement	Classification
R01: Search	Must
R02: Result	Should
R03: Advisory Functionalities	Should
R04: “Common Case Fast”-rule	Must
R05: Template support	Must
R06: Non-Technical details	Must
R07: Drag & Drop	Should
R07: Functions first, presentation later	Must
R08: User-Based design	Should
R09: Fast reaction	Must
R10: User validation	Must
R12: Semantic Spaces usage	Must
R13: User Profile Management	Must

R14: Dashboard Integration	Must
R15: Holistic design adoption	Should
R16: Component Reuse	Should

Table 8: Summary of all Requirements

4. Functional Specification

This section defines the functionalities of the different components of the SOA4All Composer, which are to be realized in Task 2.6. These features correspond to the requirements set by the modeling language specification, the user modeling needs and the integration with the other components of the SOA4All Studio. Some of the functionalities serve as a thin wrapper around functionalities developed outside Task 2.6, but will nevertheless be listed here in order to indicate that users have direct access to them from the Composer.

Functionalities listed in the following sections include:

- Create, Read, Update, and Delete a Process
- Common Modeling Functionality
- Guided Modeling
- Process Model Patterns and Templates Support
- Drag & Drop
- Collaborative Commenting
- Exploration and Organization
- Undo / Redo
- Auto-Save
- Integration with external components

4.1 Create, Read, Update, and Delete (CRUD) a Process

The SOA4All Composer will support the standard creational activities such as create, read, update and delete a process model. The graphically modeled service composition will be translated into an extended XML Process Definition Language *(*XPDL) format defined in D6.3.1 and persisted in the Semantic Spaces storage layer investigated in Task 1.3. The opposite transformation will be supported in order to load and restore a process model from its definition. Semantic Spaces offer an interface to store and retrieve files in a distributed manner. Figure 4 presents the “Save As” functionality, made transparent to the user.

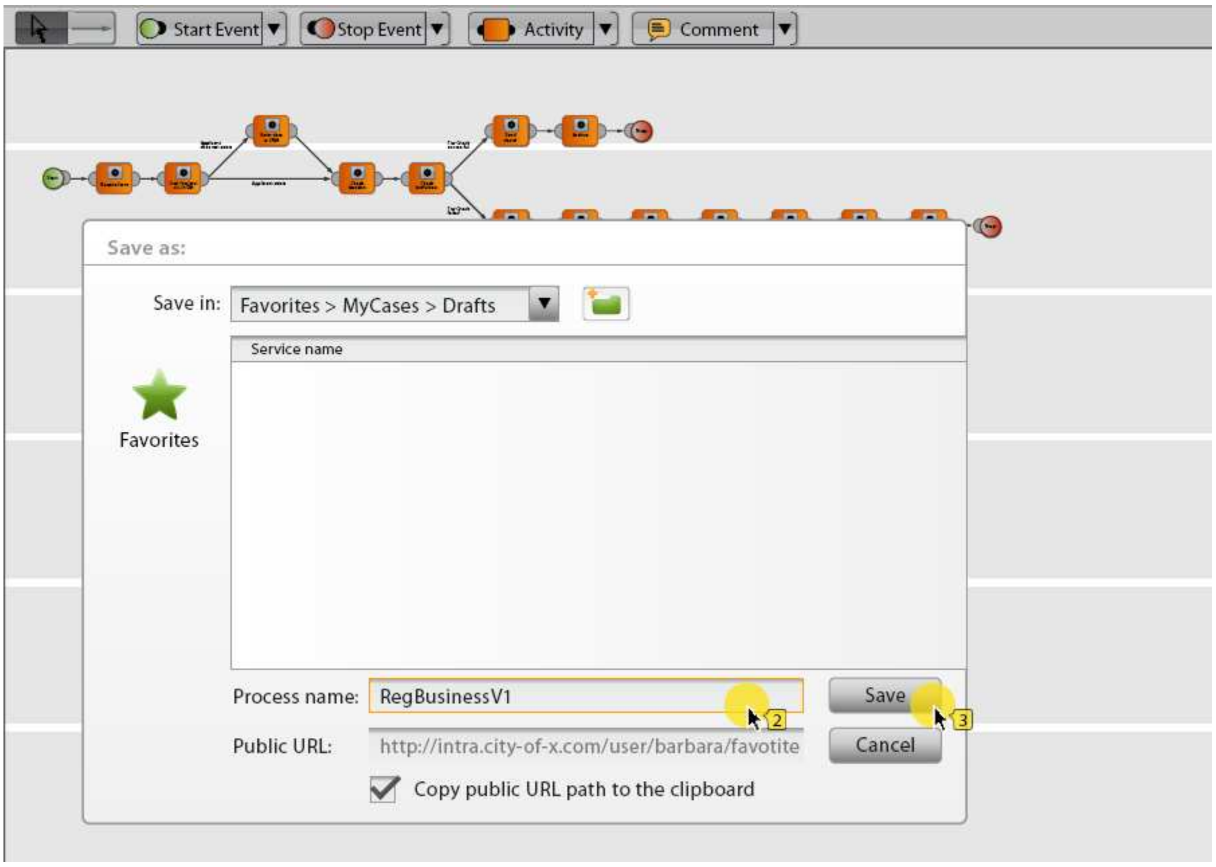


Figure 4 Save As in the SOA4All Composer

4.2 Common Modeling Functionality

The SOA4All Composer will offer common modeling functionality related to graphical composition of process models – addition and removal of process elements, rearranging, configuration etc.

Currently, D6.3.1 has defined activities and connectors as the main building blocks in the lightweight modeling language (for a formal model see D6.3.1). An activity may represent a concrete Web Service Description Language (WSDL)-based web service, a REST service endpoint, a Web Service Modeling Ontology (WSMO) goal, or a task that requires human input. Users will be able to connect separate activities with configurable connectors with AND, OR and XOR semantic in order to indicate connection flows and thus define complex composition patterns. Figure 5 shows how a connection between two activities can be created, with an emphasis on intuitiveness.

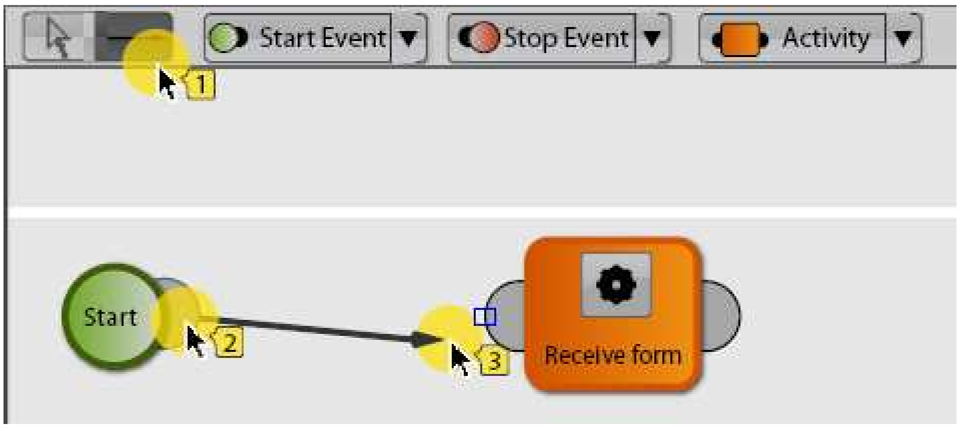


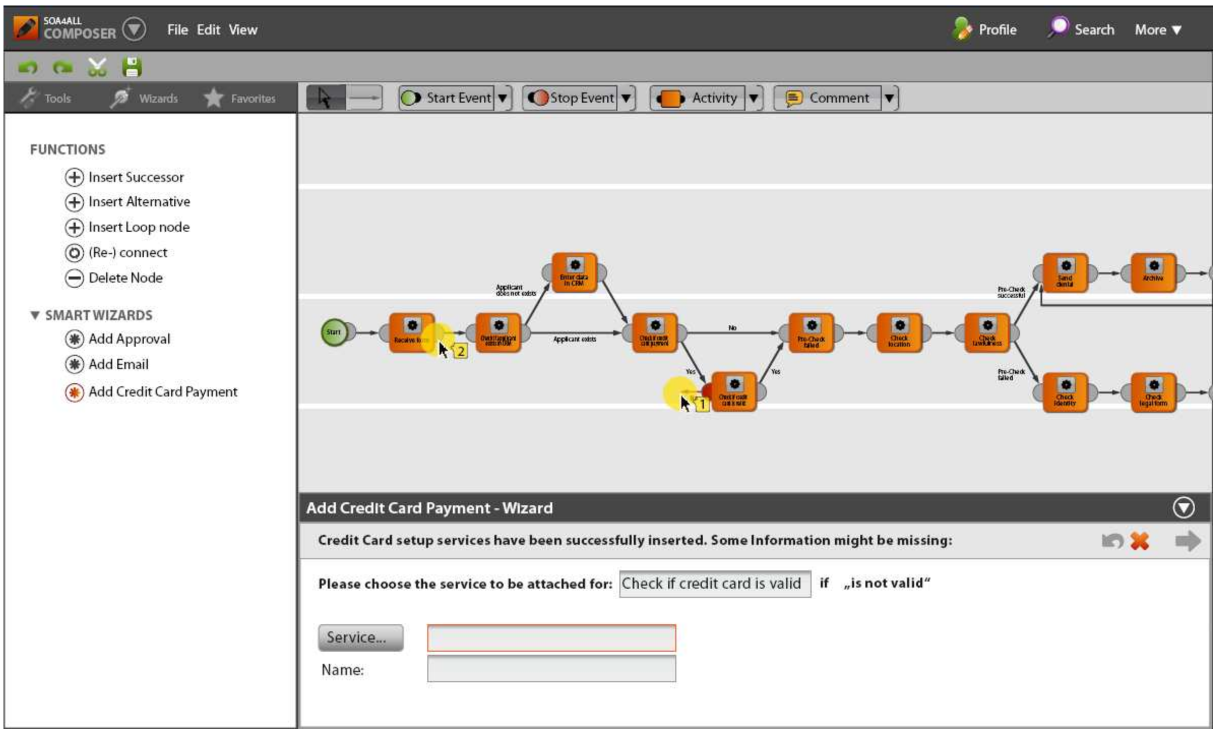
Figure 5 Attaching Activities During Modeling

4.3 Guided Modeling

The Composer will offer two modes of modeling – *free* and *guided*.

The *free* mode is intended for experienced users, who can manage all steps such as placing activities on the canvas, connecting the activities in a meaningful way, relating the activities to actual services, specifying goals etc. without the need for additional support.

The *guided* modeling mode is intended to help inexperienced users perform these tasks by offering them context-based step-wise guidance. *Guided* modeling will make use of wizards or assistants, which will provide meaningful suggestions based on context information such as currently selected activity or user-input such as goal of the process or search keywords. This mode will employ external services such as Task 2.2 Service Consumption or Task 2.7 Service Recommendation. The user will be free to change modes at any time during process modeling. Figure 6 shows an example of a mock-up smart wizard during process modeling.



Add Credit Card Payment - Wizard

Credit Card setup services have been successfully inserted. Some Information might be missing:

Please choose the service to be attached for: Check if credit card is valid if „is not valid“

Name:

Figure 6 Mock-up Smart Wizard Example

The project intends to make use of recommendation, rating, semantic and context information in the guided modeling wizards

.

4.4 Process Model Patterns and Templates

The SOA4All Composer will provide a set of predefined process patterns and templates to users. Process patterns simplify the process of designing a solution by providing commonly occurring behaviors in process modeling, which can be configured to match the concrete requirements.

On a higher level, workflow templates combine different process patterns into more complex compositions. Templates are also intended to be generic and configurable. Patterns and templates are a subject of research for D.6.3.1 and this is where an exhaustive list can be found.

4.5 Drag & Drop

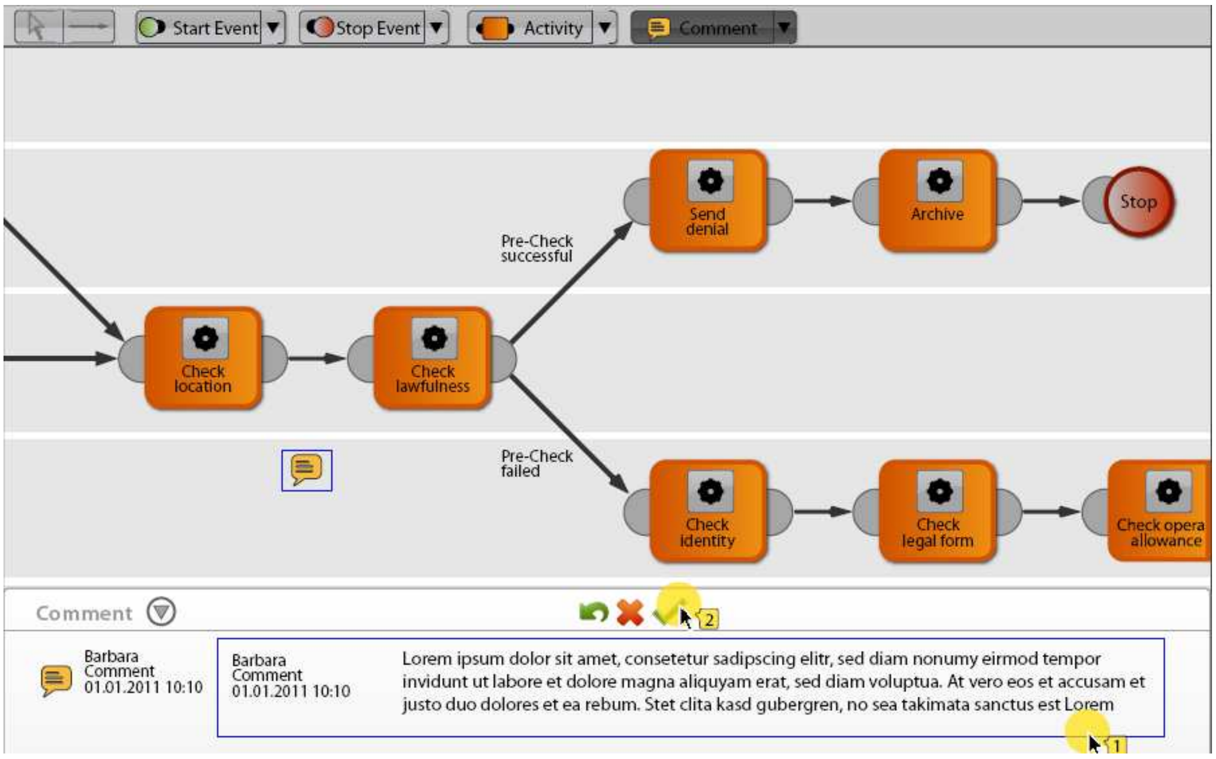
The Composer will offer a canvas widget, on which the graphical service composition occurs. Activities will be available in a separate toolbox widget and users will be able to simply Drag & Drop (DnD) them on the canvas. Creating connection will also be intuitive by first selecting the source then dragging and releasing the mouse at the destination.

Drag & Drop will also be available for services and processes regardless of whether they are returned from search or favorites catalogue browsing. DnD is planned to be provisioned by Task 2.4.

The modeling canvas will feature a Snap-To-Grid functionality, which automatically aligns activities and other elements as they are placed on the canvas by snapping them to an invisible grid. In this way, the designed process retains a certain structure and proper alignment.

4.6 Collaborate Using Comments

A collaborative commenting feature will allow users to model, discuss, and exchange ideas in a collaborative fashion. Comments will be realized as annotations attached to process elements, which add an additional descriptive value to the process model. Comments will be stored together with the process and potentially could be used for service discovery. Figure 7 presents the commenting feature of the Composer.



Barbara Comment 01.01.2011 10:10

Barbara Comment 01.01.2011 10:10

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem

Figure 7 Comments Feature in the SOA4All Composer

4.7 Explore and Organize

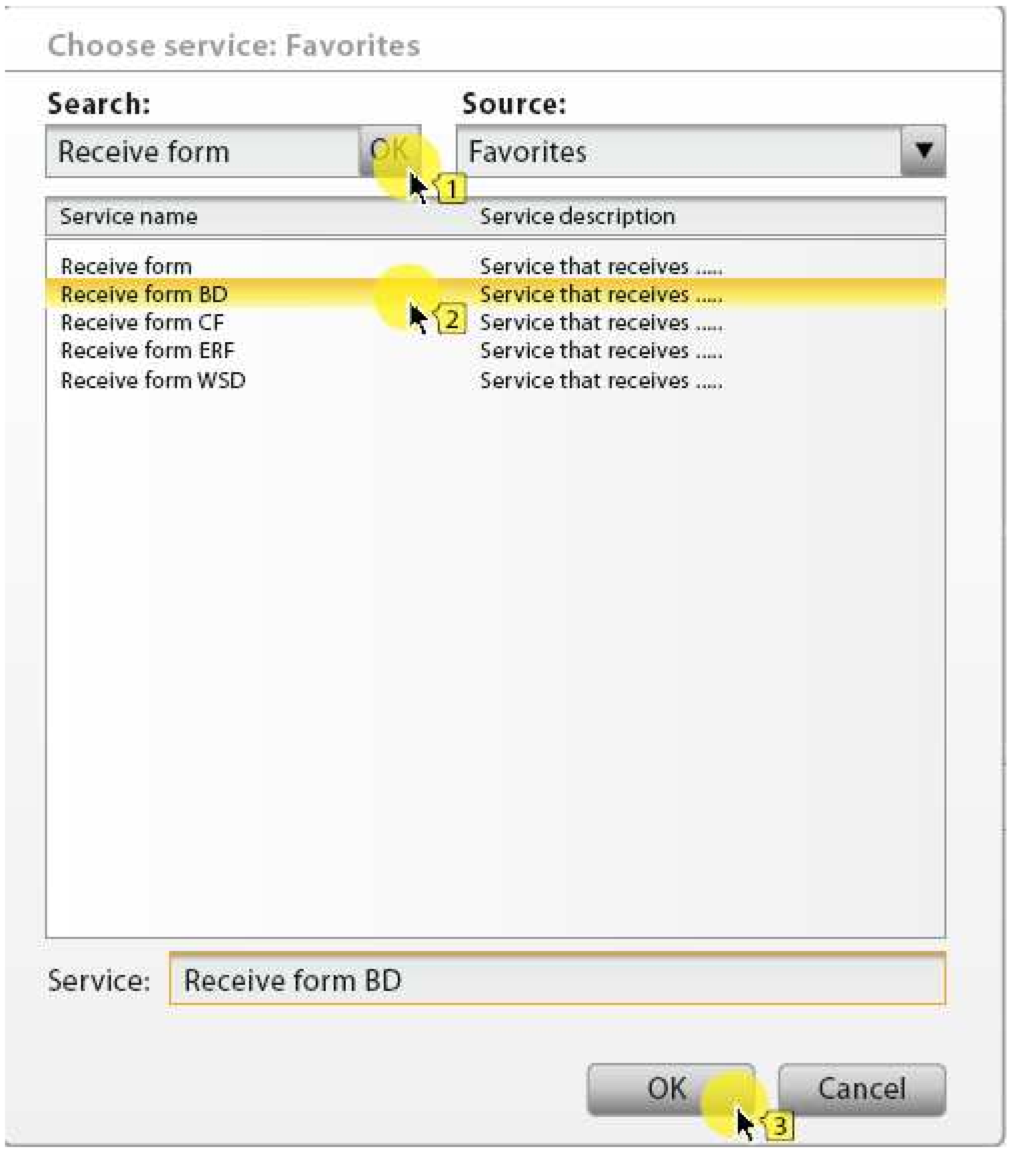
The SOA4All Composer will allow the user to explore and organize processes and services, both his/her own and those deployed by others. See Figure 8 for an example screen of favorites browsing.

Explore:

- Search: By accessing search functionality of the SOA4All Studio, more specifically Task 2.2
- Browse: By browsing through categories and tag-clouds.
- Favorites: By navigating through processes that have been added to the favorites.

Organize:

- By adding processes to favorites catalogue and organizing them in a folder structure



Choose service: Favorites

Search: Receive form

Source: Favorites

Service name	Service description
Receive form	Service that receives .....
Receive form BD	Service that receives .....
Receive form CF	Service that receives .....
Receive form ERF	Service that receives .....
Receive form WSD	Service that receives .....

Service: Receive form BD

Figure 8 Search and Browse

As part of the exploration possibilities, the user will be able to examine services in more detail by reading their detailed descriptions such as WSDL documents, WSMO descriptions, mark-up and plain text descriptions or even associated images. These will be visualized together with additional information coming from other SOA4All components such as rating, monitoring and analysis information etc. The actual content is provided externally, the SOA4All Composer supplies the necessary component to support a large set of representations.

4.8 Undo/Redo

A desirable feature for an interactive and user-friendly modeling application is the ability for a user to make recent changes undone. The possibility of undoing and redoing user commands gives the user a feeling of security. This is especially important for inexperienced users in the early learning phase but also contributes to the effectiveness of advanced users as well. The presence of undo/redo support is a good indicator of the maturity of an application.

The SOA4All Composer will feature an Undo functionality, which would support reverting major modeling steps – e.g., adding an activity, removal of activity etc. The provisioning of undo functionality for non-critical steps such as text editing is still under consideration.

4.9 Auto-save

The SOA4All Composer will include Auto-save in predetermined regular intervals in order to reduce the risk and impact of data loss in case of crash or freeze. Auto-save backups will be stored in the externally provided, semantically-enriched Semantic Spaces storage. Intermediate saves will be purged whenever the user finishes their work.

4.10 Communication with External Components

The SOA4All Composer is a part of the SOA4All Studio and is related to many different components in the complete SOA4All ecosystem. As such it will provide means to communicate with all related components as needed.

5. Architecture and Interface Specification

The SOA4All Composer will be implemented as a web-based desktop-like modeling application. It will be based on the Google Web Toolkit (GWT) [GWT] and an extension library ExtGWT [ExtGWT]. The Composer will be registered in the SOA4All Studio as a plug-in, respecting the specification elaborated in D2.4.1. For functionality not offered by these two frameworks custom native JavaScript through JSNI (JavaScript Native Interface) will be used.

Rich Internet Applications tend to have an ever-increasing size and complexity and large applications are difficult to manage. This is where the Google Web Toolkit (GWT) shines - it brings the manageability of Java to RIAs. The same code used to perform business logic can be executed on client-side, server-side or both depending on the application requirements. In this way, so-called progressive enhancement can be conducted – development begins with a standard approach – working markup/basic client-side scripting and server-side logic, and then iteratively enhance it in layers to add functionality for the browsers that can handle it, so that it always works, even if a layer is missing or broken.

5.1 Composer UI Design

The Composer will make use of the UI components developed in Task 2.4 (see Figure 9) and will respect the provided stylesheets and common look & feel suggestions.

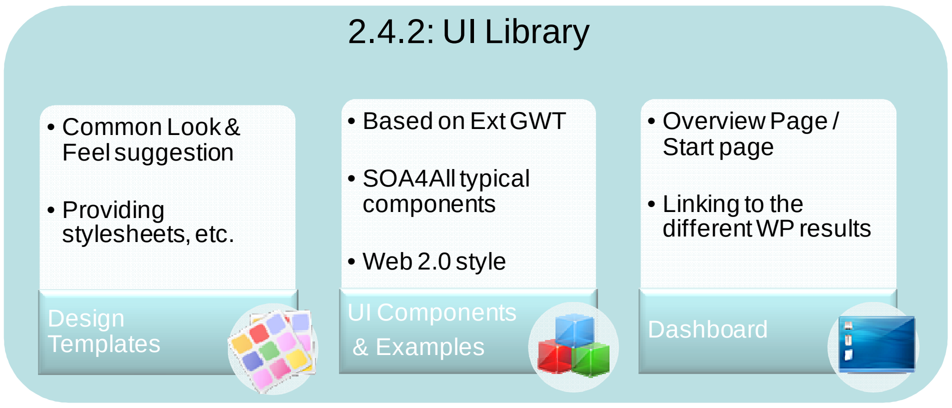

2.4.2: UI Library
• Common Look & Feel suggestion
• Providing stylesheets, etc.
Design Templates
• Based on Ext GWT
• SOA4All typical components
• Web 2.0 style
UI Components & Examples
• Overview Page / Start page
• Linking to the different WP results
Dashboard

Figure 9 Design Templates, UI Components & Examples and Dashboard from Task 2.4

Among all components to be delivered by Task 2.4, the most relevant for the SOA4All Composer are:

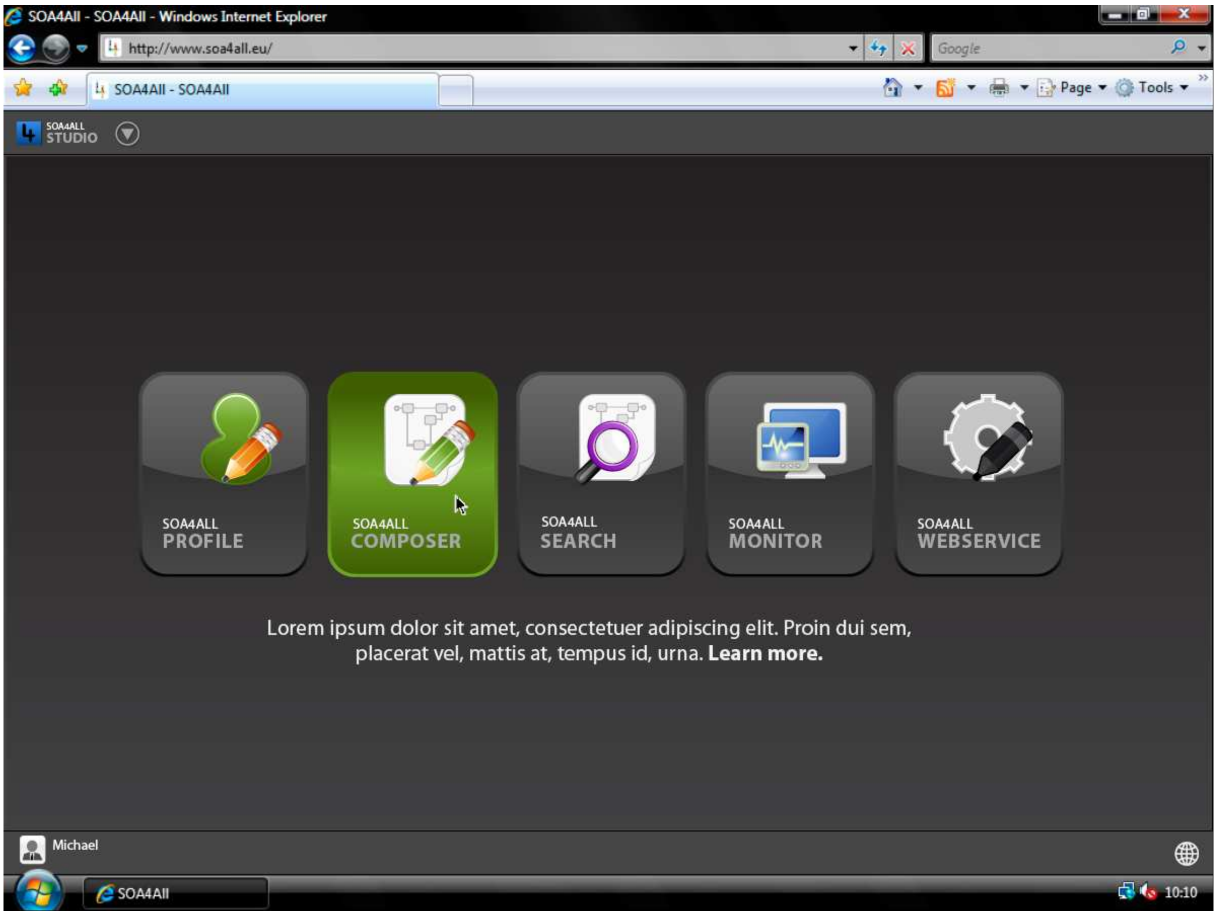
- A Graph Visualization Widget
- A Drawing API
- Integration into the SOA4All Dashboard including a flexible menu toolbar
- List components for search results and list entries
- A tag cloud widget

However, Task 2.4 provides only a set of generic UI widgets and will hence not provide all

Composer specific components. The generic widgets will be customized in order to fit the Composer requirements. In addition, a number of custom GWT widgets will be developed in 2.6 in order to complete the requirements of the SOA4All Composer:

- Modeling Canvas supporting Drag & Drop, internally using the Graph Visualization Widget from Task 2.4
- Modeling Tools Widgets
- Pluggable Wizard Widget that can be instantiated as different specialized Wizards

As part of the SOA4All Studio, the entry point to the SOA4All Composer will be from the Dashboard View, as illustrated in Figure 10.



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin dui sem, placerat vel, mattis at, tempus id, urna. **Learn more.**

Figure 10 Dashboard Entry Point for the SOA4All Composer

After entering the Composer the user has the choice to use different ways to build service compositions – e.g search, browse favorites, start modeling wizards, use patterns and templates etc. Figure 11 shows a screenshot of the SOA4All Composer UI during modeling - with opened Modeling Canvas and Tools components.

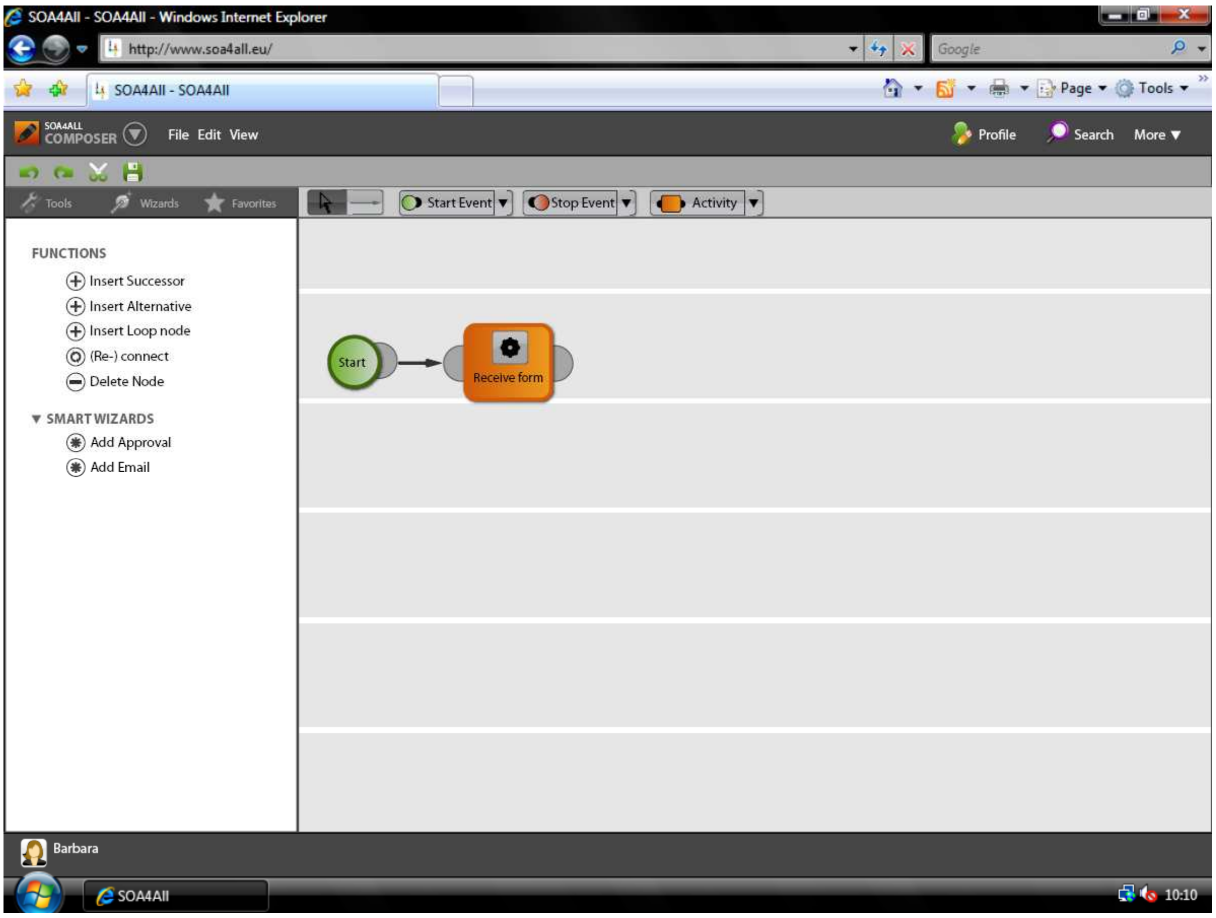


Figure 11 SOA4All Modeling Canvas

The lightweight modeling language of Task 6.3 defines a core set of constructs for modeling service compositions. According to D6.3.1 the limited set of components at this moment is intentional to make sure that the language remains light-weight and usable by a broad spectrum of users. D6.3.1 cites that a study has revealed that the average subset of BPMN used in most of the existing process models consists of just nine different symbols. The SOA4All Composer will try to minimize the modeling effort by explicitly omitting modeling elements from the language such as gateways as separate visual elements as they are believed to be hard to understand for non-technical users. They will be modeled implicitly by allowing multiple outgoing and incoming connections from activities. The input and output endpoints of activities will be configurable to mimic gateway functionality.

The set of visual components currently consists of the elements listed in Table 1. This table also presents the graphical representation of these constructs in the SOA4All Composer.

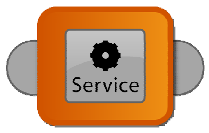
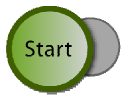
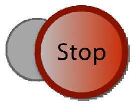
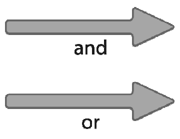
Name	Meaning	Symbol
Activity	An activity is a unit of work, the job to be performed. It can be either concrete (service) or abstract (goal).	
Start	Starts a process flow	
End	Ends a process flow	
Connection	Connections define the execution arrangement of activities.	

Table 1 Visual Representation of the Language Elements as Defined in D6.3.1

5.2 Composer Architecture

The SOA4All Composer will exhibit a Rich Internet Application (RIA) architecture consisting of server-side core business logic integrated in the SOA4All Studio and a light-weight client-side user interface.

The choice of GWT as underlying framework gives a number of implementation choices. The custom GWT-RPC mechanism makes it very easy to transfer data between the client and the server sides through out-of-the-box object serialization into opaque JavaScript Object Notation (JSON) [JSON] objects. While this approach is straightforward, it ties both sides together and the server-side is difficult to access outside the context of the respective GWT client. In the context of SOA4All, which promises integration of services at a world-wide scale in a Web 2.0 manner this approach appears unfit.

The alternative option is to design the server-side in a REST-ful manner, exposing functionalities as resources [RoyFielding]. The benefits are numerous:

- Uniform interface – HTTP methods (GET, POST, PUT, DELETE)
- Named resources - the system is comprised of resources which are named using a URL
- Interconnected resource representations - the representations of the resources are interconnected using URLs, thus allowing a client to progress from one state to another
- Open data-exchange format (JSON)

- Accessible through different client implementations

The drawback in respect to GWT-RPC is that the process of converting from and to the data-exchange format JSON (also known as marshaling and unmarshaling) on the client and server side is not automatic, thereby incurring a greater one-time implementation effort.

The project is planning to support both approaches by abstracting protocol considerations from business logic as much as possible. It will start with the less laborious approach of GWT-RPC as a proof-of-concept and will open the interfaces as development progresses.

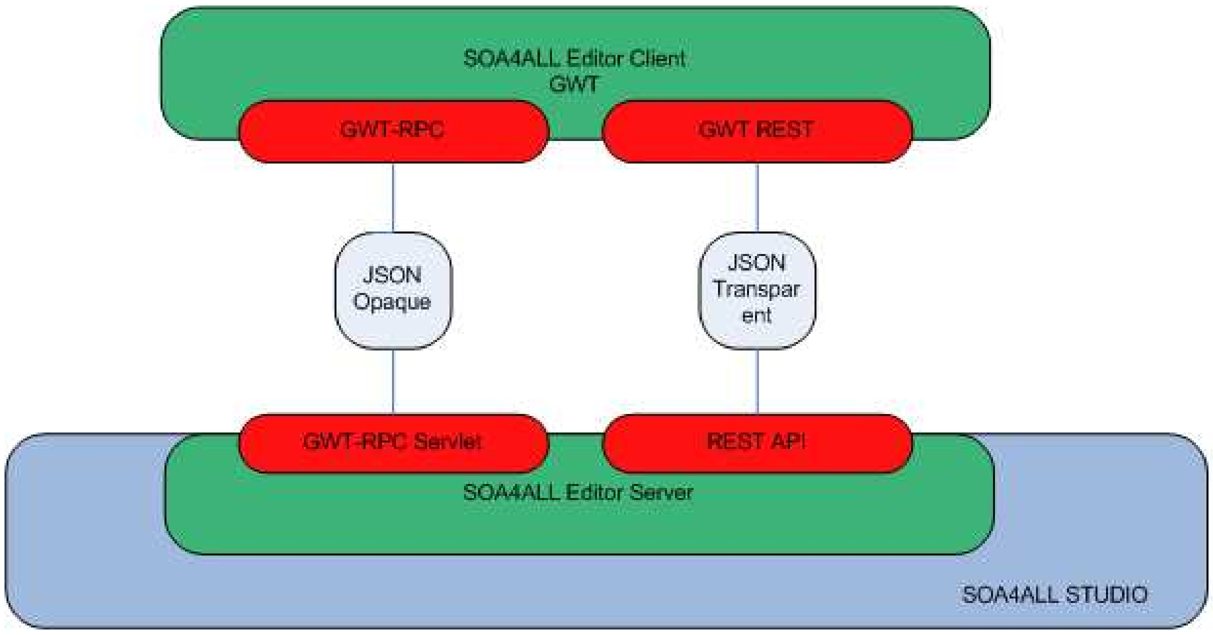


Figure 12 High-level SOA4All Composer Architecture with emphasis on the client-server communication

RIA applications typically involve rather complex client-side logic required by the desktop-like look & feel. Moving some functionality to clients also makes the application more scalable. This fact suggests a careful design of the client-side logic and the application of useful design patterns for modularity reasons.

The SOA4All Composer client-side will include an adapted Model-View-Controller (MVC) architectural pattern, which takes into consideration the asynchronous nature of the application. The controller communicates with the server as required – e.g., to synchronize the client-side model, call an external service or to fetch data. The external services in the context of the SOA4All Composer are the ones developed in Task 2.4. Furthermore, the server-side logic implements or dispatches all critical functionality such as storage, authentication etc. The client side is always considered untrustworthy; therefore some elements are kept server-side only.

Figure 13 gives a high-level overview of the proposed architecture. For clarity the different communication approaches have been unified.

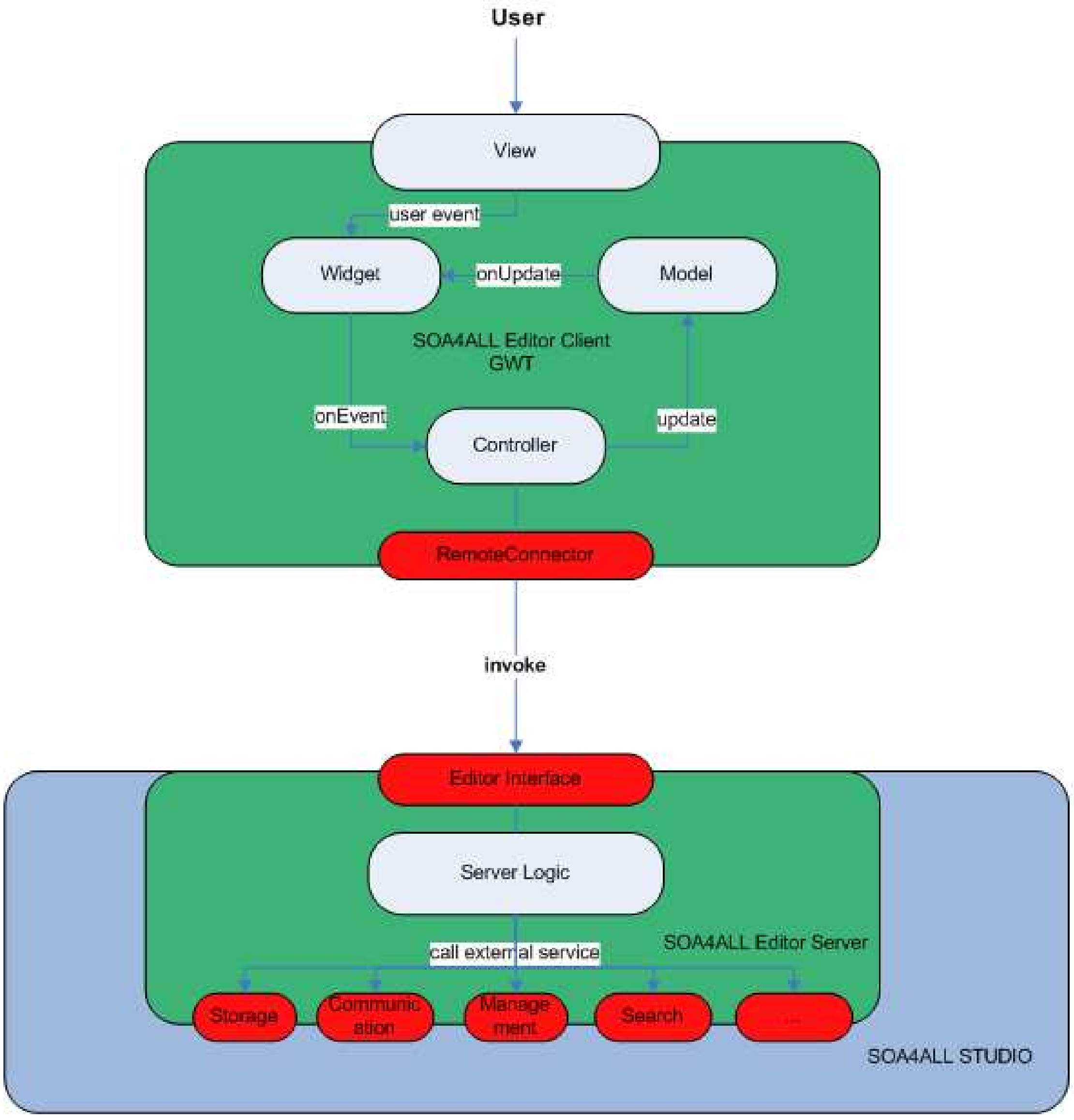


Figure 13 High-level SOA4All Composer Architecture

A detailed UML design of the intended technical specification of the SOA4All Composer Client is given in Figure 14. The emphasis is on the separation on concerns in order to allow for component change with minimal effort. The architecture is open for extension as requirements may change during the course of the project.

The **View** consists of the SOA4All Composer widgets presented above, glued together but each of them serving a different purpose. The most important widget is the modeling canvas, which visualizes the modeled service composition. Widgets capture user interactions and dispatch them to the controller for proper handling.

The **Controller** contains the client-side logic and translates the user events into model changes or invokes a remote service interface. The controller is also responsible for submitting the client-model to be persisted on the server-side.

The **Model** represents the service composition, which a user defines through the provided modeling tools with help from context-aware wizards, recommendations, ratings etc. The model is unaware of the existence of the controller thus decoupling logic from data.

The best way to understand the components relationships is through examples. Let's consider two common cases - the user adds a new activity and the user saves a process model. In terms of events they translate to:

Add new activity:

- Modeling canvas widget registers a drag & drop event, captures its context such as coordinates, element dropped etc
- Canvas passes the information to the controller
- Controller interprets the information and updates the model (it can potentially trigger other functions such as recommendation on the basis of context)
- Model updates the view and the element is visualized on screen

Save service composition:

- View registers a save event through a button click
- Controller converts the current model in the respective wire-format (JSON) and sends it to the server side
- Response from the server comes asynchronously and is presented to the user – e.g service composition stored successfully

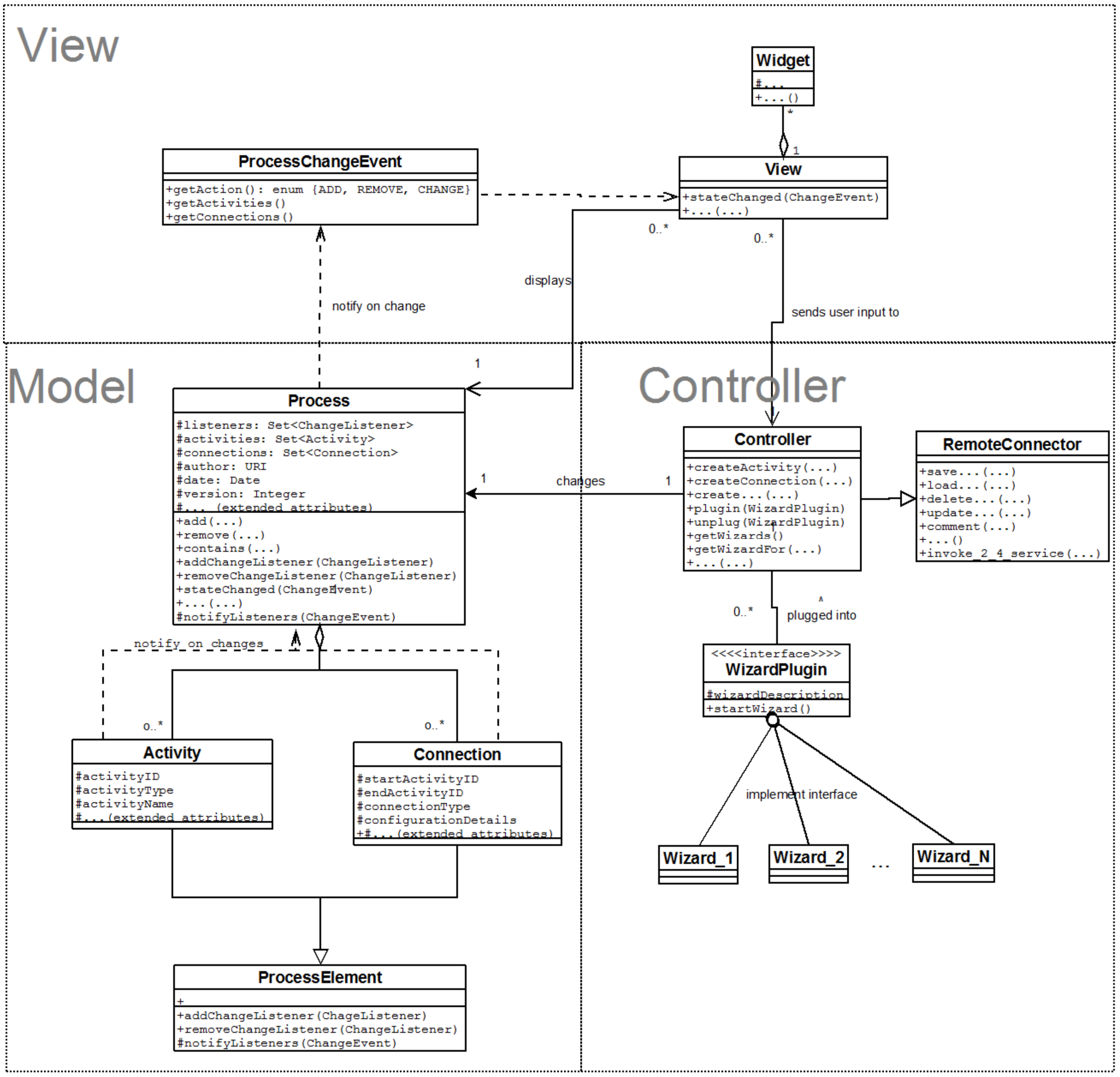


Figure 14 Simplified UML Design of the SOA4All Composer Client-side

5.3 Composer Server-side Interfaces and Data-formats

There are a number of functionalities, which are better delegated to the server side of the SOA4All Composer. They include persistence and retrieval of service compositions, communication with other SOA4All components – search, recommendation etc.

The server-side will be open for alternative uses and reuse by exposing its functionality in a client- and platform-independent manner. Since the Web is the compelling example for interoperability and ease of use the project will employ the web principles on the SOA4All Composer server-side. Functionality will be exposed as addressable resources in a REST-ful manner and interfaces will follow the HTTP convention:

- GET - to obtain the current state of a resource. GET is idempotent and has no side effects
- POST - to update a resource state. The payload of a POST request carries the required update information
- PUT - to create a new resource if not existing. Since some servers and browsers may not support PUT requests, we will provide an alternative POST-based implementation in which the URI specifies that the request should be interpreted as PUT. E.g., http://SOA4All.com/studio/composer/processID/put
- DELETE - to remove an existing resource. Since some servers and browsers may not support DELETE requests, we will provide an alternative POST-based implementation in which the URI specifies that the request should be interpreted as DELETE. E.g. http://SOA4All.com/studio/composer/processID/delete

The resources that will be exposed by the server are naturally the service compositions and their related elements such as individual activities, connections, ratings, comments etc. A proposed addressing structure is the following:

URI	GET	POST	PUT	DELETE
/composer/processID	Retrieve full process representation	Update process state	Create new process	Remove process
/composer/processID/comments	Get all comments for process	Post new comment	Add first comment	Remove comments
/composer/processID/rating	Get process rating	Update process rating	Add first rating	Remove rating
/composer/processID/{other}	Get element representation	Update element	Add element	Remove element
/composer/processID/activity/ID	Get activity representation	Update activity	Add activity	Remove activity
/composer/processID/connection/ID	Get connection representation	Update connection	Add connection	Remove connection
/composer/{wp2.4service}	Get data	Update data	Create data	Remove data

Table 9 Proposed Addressing Structure of the Process Composer Server

The interface can change to reflect requirements changes but it gives an impression of how the SOA4All Composer server-side is open for unintended uses and extensions. External services provided by Task 2.4 components will be accessible as well.

The data-exchange format is intended to be JSON. The JSON process representations submitted to the server-side will be transformed to an extended XPDL format according to D6.3.1 and will be persisted in the Semantic Spaces storage.

This document will not present an exhaustive specification but rather a simplified draft for proof-of-concept purposes. The service composition representation, which is the pivotal point of the application, will be represented in the following way:

```
var activity = [
        {
           "id": "Identifier",
           "name" : "Activity name",
            "type" : "Activity type",
           "outgoing" : ["connectionA_ID", "connectionB_ID" /*..*/]
           "incoming" : ["connectionC_ID", /*..*/]
           /* other attributes */
        }
];

var connection = [
        {
           "id": "Identifier",
           "name" : "Optional connection name",
           "type" : "Connection type", /* e,g OR, AND */
           "from" : activityA_ID,
           "to" : activityB_ID,
           /* other attributes */
        }
];

var process = [
   {
        "id": "Identifier",
       "name" : "Process name",
       "rating" : "Process rating",
        "comments" : ["comment1", "comment2", /*..*/],
        "activities" : [
           /* activities in JSON form listed here, see above*/
        ]
        "connections" : [
           /* connections in JSON form listed here, see above*/
        ],
        /* other attributes */
  }
];
```

This design of the Composer's server-side leaves room for unintended use by allowing different clients store and retrieve process models via its open interfaces. This approach is in the spirit of the openness of the SOA4All project and encourages reuse and sharing.

6. Conclusions and Next Steps

This deliverable clarifies the requirements, defines the functional specification and provides a detailed technical specification for Task 2.6 *"SOA4All Process Editor".*

The document takes into account the results from the analysis of the current state-of-the-art in the process modeling area and identifies its relationships to the SOA4All Studio and the SOA4All infrastructure as a whole. This deliverable will be used as a guideline base by the 2.6 team when implementing the SOA4All Composer in the next implementation phase of the task.

Next steps involve close collaboration with Task 2.4 for aligning the requirements for UI components and functionalities to be delivered by 2.4 as well as other WP2 components. Of very significant importance is Task 6.3, which will define the light-weight modeling language for describing service compositions in an executable manner. Findings and results of Task 6.3 will be iteratively included in the SOA4All Composer in order to offer the complete functionality and expressive power of the language at any stage of the project progress.

As specified in the "Description of Work", the first version of the prototype will be delivered in M24 as the output of D2.6.2. The next delivery iteration is planned for M30 as the outcome of D2.6.3.

References

1. Fischer, G, Giaccardi, E, Ye, Y, Sutcliffe, A. & Mehandjiev, N. 2004, 'Meta-design: a manifesto for end-user development.', *Communications of ACM*, vol. 47(9), pp. 33-37.
2. N. Mehandjiev and L. Bottaci. User-enhanceability for organizational information systems through visual programming. In P.Constantopoulos et al Composers, *Advanced Information Systems Engineering*, LNCS No 1080, pages 432-456. Springer-Verlag, 1996, ISBN 978-3-540-61292-
3. J. Johnson, Ed. 2000 *GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers*. Morgan Kaufmann Publishers Inc.
4. [MVC] http://java.sun.com/blueprints/patterns/MVC.html
5. [RoyFielding] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
6. [OReilly] http://radar.oreilly.com/archives/2005/10/web-20-compact-definition.html
7. [Oryx] http://bpt.hpi.uni-potsdam.de/Oryx
8. [Appian] http://www.appian.com/product/anywhere.jsp
9. [Soyatec] http://www.soyatec.com/ebpmn/features.html
10. [EclipseBPEL] http://www.eclipse.org/bpel/
11. [OracleBPEL] http://www.oracle.com/technology/bpel
12. [SAPNetweaver] http://www.sap.com/platform/netweaver/components/sapnetweaverbpm/index.epx
13. [GWT] http://code.google.com/webtoolkit/
14. [ExtGWT] http://extjs.com/products/gxt/
15. [JSON] http://www.json.org/