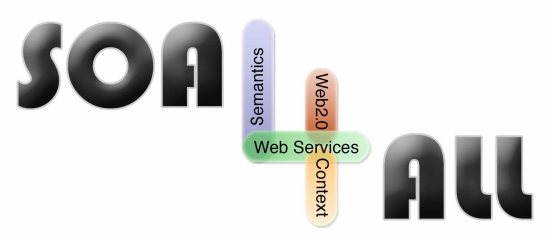


Project Number: **215219**

Project Acronym: **SOA4ALL**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

DX UI – Holistic User Interface

Activity N:	Extra Deliverable
Work Package:	Extra Deliverable
Due Date:	-
Submission Date:	26/08/2008
Start Date of Project:	01/03/2008
Duration of Project:	36 Months
Organisation Responsible of Deliverable:	TIE
Revision:	1.0
Author(s):	TIE, SIRMA, iSOCO and University of Manchester

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	18.07.2008	Kick-Off Version, integrating existing input	TIE, SIRMA, iSOCO and University of Manchester
0.2	28.07.2008	Beta 1 Version	TIE, SIRMA, iSOCO and University of Manchester
0.3	04.08.2008	Beta 2 Version	TIE, SIRMA, iSOCO and University of Manchester
0.4	05.08.2008	Beta 3 Version	TIE, SIRMA, iSOCO and University of Manchester
0.8	11.08.2008	Release Candidate 1	TIE, SIRMA, iSOCO and University of Manchester
1.0	25.08.2008	Final Version	TIE, SIRMA, iSOCO and University of Manchester

Table of Contents

Glossary of Acronyms

Acronym	Definition
UI	User Interface
AJAX	Asynchronous JavaScript and XML
GWT	Google Web Toolkit
HCI	Human Computer Interaction

Executive summary

In order to increase the usability of SOA4ALL, it is of major important to create a low entrance barrier for potential users. One way of achieving this is the provision of one holistic user interface instead of providing separate user interfaces for each work package development.

The SOA4ALL consortium therefore decided to provide a common core UI framework which forms the base for UI implementations and which will be implemented by developments in all work packages. This ensures a common look and feel among all developments and will allow easy navigation between SOA4ALL components.

This extra informal deliverable is focused on those core UI parts. It starts with describing the generic requirements for a holistic user interface in SOA4ALL including Human Computer Interaction aspects. Next the deliverable analyses possible frameworks and tools that may be used for an implementation. It focuses on frameworks for creating Rich Internet Applications (RIA). After analysing the different frameworks, the deliverable gives a concrete recommendation for a specific framework.

Based on the framework analysis and conclusions, the deliverable then provides an overview about how to develop a core UI framework. This core UI framework will provide generic functionality such as UI dialogs for managing users or preferences. It may be used by concrete work packages to perform common functionalities. In general, the core UI framework will be used for all 'common' tasks that are work package independent, while the work package specific implementations will concentrate on realizing user interface elements that are work package specific (e.g. an interface for service construction). The core UI framework will allow work packages to seamlessly integrate into the holistic SOA4ALL UI by providing a plug-in based architecture and by providing templates and stylesheets for a common look and feel.

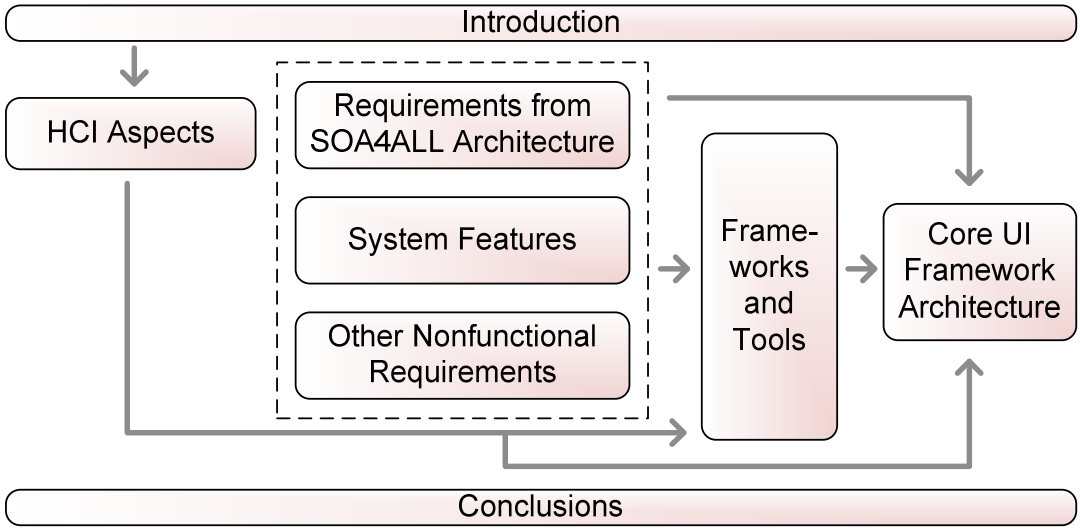
1. Introduction

The purpose of this deliverable is to give an overview about the holistic UI approach that is applied in SOA4ALL.

The document will describe:

- Human Computer Interaction (HCI) aspects describing approaches for an easy to use and comfortable UI which is required in order to make SOA4ALL accessible for everyone.
- Requirements for user interfaces, based on the IEEE Software Requirements Specification template (IEEE SRS). The SRS template is the recommended practice for software requirements specifications of the IEEE.
- Rich Internet Application frameworks that can be used as a base for creating a holistic user interface in SOA4ALL
- A core UI Framework, which is a common framework to be used by all SOA4ALL plug-ins in order to provide common functionality and a base UI look and feel

The following figure shows the overall structure of the document and how the different parts are interconnected:



Note: This deliverable will concentrate on the description of elements that belong to the holistic UI approach of SOA4ALL. It will not cover the work package specific user interfaces since those are largely independent form the core UI approach. Work package specific user interface issues are therefore described in the corresponding work package deliverables.

2. Human Computer Interaction aspects

Achieving the objectives of SOA4ALL requires the delivery of user interfaces that are tuned to the expected skills and foreseen tasks of our target users. This section will explore the need for user-centric approach to designing and evaluating these interfaces as well as outlining a palette of design and evaluation techniques. SOA4ALL will choose techniques from this palette as appropriate to the needs of specific user interface aspects and case studies.

The long-term aim of SOA4ALL is to open up construction and consumption of software services to everyone. Initially, however, the projects target users are those found in the SOA4ALL case studies (WP7-9). These case studies are still in the initial stages of definition, yet it is already clear that some of SOA4ALLs target end users (e.g. the civil service administrators from the WP7 case study) will not have a software development background. Their main interest would be to use SOA4ALL to fulfil the support requirements of their main job tasks and duties rather than program. At the same time, some of their manipulations of SOA4ALL services will cross the boundary from setting parameters to software modification and even outright development. Examples of such activities are service construction and composition. This type of involvement is known as end user development or EUD - see for example Sutcliffe and Mehandjiev (2004).

EUD research results provide an insight into the type of software interfaces and service representations likely to be effective in relation to our use cases and target end users. This section will present the generally known facts about EUD interfaces and representations, and will chart a systematic process of tuning these generic findings and requirements to the needs of our target end users.

2.1 What makes a user-friendly representation?

There is a widespread belief that graphical representations help users who are not programmers to solve the complex cognitive tasks involved in programming and software design (Shu, 1988). The saying “a picture is worth a thousand words” is often cited but experiments have demonstrated that this hypothesis only holds for cases such as process modelling and electronic circuit board design, where the structure of the problem being described is naturally mapped to a diagrammatic or spatial representation.

One advantage of graphical interfaces and representations is the possibility to use metaphors and thus draw on a large pool of knowledge and skills shared by most humans. However, systematic studies have shown that benefits of metaphor are small and mainly restricted to mnemonic assistance (Blackwell,1998). Indeed, even the optimal form of representation (graphic versus text) was found to depend on the background of the users and the task they are aiming to accomplish ((Blackwell,1998 and Nardi, 1993)).

The visual languages community is now accepting that technical or expert-focused diagrams by themselves do not help users to any significant extent, and that users would find as most useful the representation which is most familiar to them, for example because it is similar to their non-computerised drawings.

Nardi (1993) also shows that formality of representation by itself is not a barrier to use by non-programmers, as long as the representation focuses on tasks of intrinsic interest to the user. Sutcliffe *et. al.* (2003) extend the motivational dimension and see the trade-off between expected benefits and learning costs as a main determinant of uptake of a representation or a tool by its users. This has been extended to organisational context by Mehandjiev *et. al.* (2006), who identified the main risks and benefits for end users being involved with the development of software, including the construction of software services.

Apart from the motivational and background aspects, cognitive psychologies and HCI

academics have developed a number of frameworks to evaluate the general usability of languages and their supporting environments from the perspective of models of cognition and design which are believed to apply across all human beings. The most famous is Green's system of *"cognitive dimensions*" (1989). It aims to assess the degree to which a language (be it textual or diagrammatic) facilitates the cognitive strategies for design problem solving. The framework has been used for comprehensive evaluation of visual languages and environments (Green and Petre, 1996), and its use has been extended from evaluation to improvement, using an associated set of improvement strategies.

In conclusion, if SOA4ALL is to create useful and effective service representations and user interfaces, it needs to integrate

(a) knowledge of the background and skills of our target end users, with

(b) HCI analysis and improvement tools such as the cognitive dimensions.

2.2 Systematic process of creating and evaluating our representations and interfaces

Described below is a systematic process which leads from requirements capture and user analysis to the development and validation of representations and interfaces tuned to the target case studies.

- The following main stages summarise the process of creating effective and usable representations and environments, which can then be used for user-driven service composition:
- Understand end users
- Understand case studies
- Understand existing tools and applicable standards
- Construct representations and interaction techniques
- Validate the effectiveness of representations and interaction techniques

These techniques are briefly described below. Note that not all techniques will be applicable or suitable for all case studies and technical aspects of SOA4ALL, and the precise combination of techniques can only be made when the case studies and underlying technology choices are finalised.

2.2.1 Understand our end users

The discovery of stakeholders and their "wish lists" of system requirements is an important step in the design process of any system. Requirements elicitation is defined as: "the process of obtaining the requirements of a system by communication with users, customers, and others who have a stake in the system development" (Sommerville and Sawyer 1997).This process can be decomposed into five stages (Christel and Kang 1992):

- Identify the relevant stakeholders which are the main sources of the requirements
- Compose the "wish list" for each group of users
- Document and refine the "wish list" for each group of users
- Integrate the "wish list" of all groups and resolve any conflicts between viewpoints
- Specify the non-functional requirements

Hence, the first step in developing requirements is the identification of the stakeholders of SOA4All. The identification of the system stakeholders can be guided by answering a set of questions adapted from the takeholder analysis, namely:

- What does the persona[1] have to achieve?
- How is success measured?
- What knowledge and skills does the persona have?
- How is SOA4All going to affect the work of the persona as compared to traditional offering?
- How often is the persona expected to use the system?

At this stage the investigators contact users who are representative of the target system users in order to gather answers for the questions above. They proceed to create abstract profiles of target users based on the narrative of the case study, and exemplify these using Personas. Personas are imaginary characters created to represent the different users of a particular system. They are usually synthesised from data collected from user interviews and captures goals, skills, attitudes, behaviour patterns, and environment. The creation of personas can enable SOA4ALL partners to construct a consistent and common understanding of the possible end user groups, and guide them in examining how well their solutions meet the needs of individual user personas. Additionally, the provision of a hypothetical persona enables the designers to better infer the needs of a real user (Alan Cooper, 1999, The Inmates are Running the Asylum), (Cooper and Saffo 1999).

Once the end users are identified, preliminary studies need to be carried out in order to provide a complete understanding of the users' needs. To ensure that all information concerning the customers' needs is collected and understandable several elicitation techniques can be employed:

The stage of personal approach may involve any of the following activities:

- Interviews and Observation (Initial part)
- Questionnaires and Brainstorming sessions (Middle part)
- Use cases, Scenarios and Storyboarding, and Prototyping (Late part)

These techniques can be incorporated within workshops which are facilitated working sessions in which all project stakeholders (system developers and end users) are gathered together to discuss the project needs and goals.

Use cases describe the interactions between a user and a system represented as a sequence of simple events. Use cases capture who does what with the system, for what *purpose* without the need to know about the system components. Use cases are used as a means of communication between the end users and software developers so bridging the gap of these different worlds. Use cases can be translated into scenarios which can be understandable by the target user. They are usually described using the end user language avoiding any technical jargon. One or more instances of scenarios representing a possible flow can be generated from a use case. (Ref: Booch, 1999)

Storyboarding is also characterised as an effective and low fidelity prototyping mechanism for requirements elicitation as it captures early "Yes, But" reactions from the target users. Storyboards are a short graphical description of a specific scenario. In addition to the details of the interface, storyboards identify the users; explain what happens to them, how it happens, and the interaction between the users and the system. (Ref: Storyboarding: An empirical determination of best practices and effective guidelines, Khai N Truong, Gillian R. Hayes & Gregory D. Abowd). Storyboarding perfectly fits projects with innovative content such as SOA4All.

Prototyping involves the usage of software prototypes to visualise the interface of the system

[1] Please note that we refer to a single stakeholder when using the word "persona" here.

but also the interaction between the user and the system. As such, it provides a higher fidelity projection of the future system, and can elicit feedback with finer granularity but at the expense of more in-depth development.

The following table summarises the main techniques for eliciting requirements and their corresponding advantages and disadvantages:

Table 1 1 Requirements elicitation techniques

STAGE	TECHNIQUE	ADVANTAGE	DISADVANTAGE
Initial part	Interviews	Rich collection of information Getting to know the domain and its problems	Large amount of qualitative data. Hard to see the goals and critical issues, subjective Hard to compare different respondents. Time consuming
	Observation	Map current work and practices High level of validity	Critical and usability issues hardly captured Time consuming
Middle part	Questionnaires	Gather information from many users Statistical indications, views Collect information quickly Economical	Questions often interpreted differently Hard to classify answers in open questions Hard to capture real needs.
	Brainstorming	Many ideas Natural interaction between people. Gauge reactions to stimulus materials.	Hard to create a good atmosphere and get everyone involved Time consuming Danger of “Groupthink” Requires a highly trained facilitator
Late part	Use cases, scenarios	Concentrate on a specific case. More accuracy	Solution oriented rather than problem oriented.
	Prototyping	Partial implementation of the system Addresses the “Yes, But” problem Usability centred	Premature design

Observations constitute a powerful mechanism for discovering the way of working for our target users. Myers has pioneered a method of developing "natural" representations for development (including service construction), where users are asked to solve a typical problem using only pen and paper and observed (*e.g.* (Myers *et al* 2004)). Alternatively, users are observed whilst explaining such a solution to a friend with their background.

Some of the findings reported on these studies are universally applicable (*i.e.* users prefer array-style formulas rather than enumerating through all the elements of the array using a loop), and these will be taken into account when designing the SOA4ALL representations from the very beginning. Others will no doubt be specific to the target domains of our case studies, and these can be determined after similar observations of our end users.

2.2.2 Understand the case studies

Whilst the user profiles will be used to determine details of the representations and user interface of the SOA4ALL deliverables, the scenarios found within WP7-9 can be used to determine functional requirements to the SOA4ALL system in general. Scenarios provide "Informal narrative descriptions" of the user interacting with the system. There are flexible iterative tools using scenarios throughout the entire design process, including testing and validation.

In formulating the scenarios, HCI specialists would concentrate on goals, expectations, motivations, and activities i.e. what the user is trying to achieve, and why? They would also pay attention to the frequency of expected interactions with the system, and the expected benefits for the users from learning the advanced features of the SOA4ALL system. This can help the project partners tune the functionality of the system to produce maximum value for its users, thus motivating them to use the system.

2.2.3 Understand existing tools and applicable standards

SOA4ALL brings together centres of expertise in semantics and software services, and partners from those centres bring to the project a portfolio of mature tools and modelling environments. These tools and also their underlying standard descriptions (e.g. OWL[2], etc.) are mainly focused on IT specialists so they would not be suitable for our target users "out of the box". However, valuable information may be gleaned from these environments and representations, by applying expert-based analysis using cognitive walk-throughs and the cognitive dimensions framework in its "appraisal and improvement" mode (Green and Petre,1996).

2.2.4 Construct representations and interaction techniques

Using the knowledge generated by the three previous stages investigators then proceed to create user-friendly "lightweight" service representations and environments for service creation, composition, consumption and monitoring.

2.2.5 Validate the effectiveness of representations and interaction techniques

An effective process to design user-centric representations and interaction techniques will continuously evaluate the effectiveness and usability of the representations and interaction techniques developed under the previous heading, using observational studies of user groups which approximate our target end users from the case studies. This is generally an interactive process, where the results of these evaluations are used to improve the representations designed.

[2] Web Ontology Language, http://www.w3.org/2004/OWL

3. SOA4ALL Interface Requirements

3.1 Software Interfaces

This section addresses the relations between the holistic GUI (that are covered by this deliverable) and the connected architectural components. Highlighted are the relations with the Semantic Spaces addressed in WP1, and with the Context Adaptation Framework of WP4.

3.1.1 Semantic Spaces

The holistic GUI described in this document enables the interaction of end users with a wide variety of services, which need to be stored somewhere. The tools and platforms covered by the GUI will rely on the underlying technology developed within the project. Particularly, in terms of storage and communication, it relies on the Semantic Spaces addressed in Deliverable 1.3.1, which have been envisioned for interoperation and coordination for SOA4ALL services.

Semantic Spaces define a unified model for storage and communication of data, service descriptions and ontologies. They will be composed of several kernels deployed in different machines located at different geographical location and interconnected with each other over Internet forming an unstructured P2P network. They will be used to store semantic descriptions of Web Services, Monitoring data for semantic service execution, contextual information related to services and users, and storage of compositional information.

Additionally, Semantic Spaces will be used as communication and coordination paradigm for SOA4ALL. It includes building Semantic-based event-driven and publish-subscribe mechanism for SOA4ALL Distributed Service Bus, Semantic Space based coordination for SOA4ALL platform services as well as business (external) services.

3.1.2 Context Adaptation Framework

Context is one of the four pillars that support SOA4All together with Web 2.0, Web Services and Semantics, because its impact is crucial as a way to facilitate the customization of existing services for the need and expectation of users. Contextual factors range from immediate concerns of location and language to legal issues and financial regulations.

In addition to this, the Context Adaptation Framework described in WP4 will be important for the holistic GUI of the project in order to modify the GUI and its behaviour and present an adapted version of the existing services during the composition and execution phases according to those user needs, preferences, and user context.

Deliverable 4.1.1 addresses the design characteristics of the Context Adaptation Framework and its components.

3.2 Communications Interfaces

One of the choices that have been taken in the course of the research activities that have been performed for this deliverable is the idea that the holistic GUI should be Web-based. A detailed justification for this decision will be given in section 5. The choice of using a web based UI leads to specific constrains in terms of communications interfaces. Particularly, the use of a web browser and an Internet connection are implied.

There are no specific requirements on what web browser should be used to access the

holistic GUI. In fact, following the objective of SOA4All of enabling a world where ubiquitous access to services is envisaged, it will be key that the project GUI is accessible through all major web browsers. Hence, relying on an appropriate web technology that supports general browser access is of major importance.

4. System Features

This part of the deliverable addresses the general design characteristics of the holistic GUI for SOA4All. However, the specific functionalities related to the different architectural components of each Work Package are addressed in the different deliverables.

The following subsections review the generic requirements that need to be fulfilled by the architecture and the technology that will be chosen for the realization of the GUI.

4.1 Web-based user interface

Providing a holistic interface for users might be performed by either providing a Desktop Application or by adopting a Web Application approach. Since WP2 and WP6 have a strong interaction with services being available on the web and since a wide exchange with other web technologies is planned, a web-based application is the natural candidate for a user interface. Besides, a web-based solution will mean that the tools are available in the Web, so this will significantly facilitate the provisioning of new services, hence contributing to the possibility of the web of billion of services.

4.2 Separation of concerns

In order to provide a user interface extendable by all partners, it is required to allow different SOA4All partners to contribute specific parts to the GUI in a distributed development environment. This makes it important to be able to separate specific aspects of the GUI so that they may be developed in parallel.

4.3 Structured programming model with Debugging functionality

SOA4All includes many complex elements and uses cutting-edge technologies that are sometimes not well documented or even yet 100% mature. This is due to the nature of the research project. However, in order to cope with this complexity, it is required to provide a programming model for the UI that allows developers to develop UI components in a structured way and to provide in depth debugging functionalities.

4.4 Usability and easiness of use

SOA4All is promoting a service world where many different kinds of users will access our tools to deal with services in several ways. The holistic GUI has to stress its easiness of use, thus permitting all kinds of users, both experts and non-experts, interact with it. Having a powerful set of tools covered by a complicated GUI would prevent many users from accessing SOA4All functionalities, and that would be a serious drawback for the envisaged service world paradigm.

SOA4ALL will follow important usability principles, such as Jakob Nielsen's "Ten Usability Heuristics" [Nielsen, 2005]:

- **Visibility of system status**: The GUI will inform users of what is going on
- **Match between system and the real world**: Language used will be understandable by both expert and non-expert users, rather than full of system-oriented technical terms
- **User control and freedom**: The GUI will provide "emergency exits" to leave unwanted

states caused by mistakes. It will also support undo and redo.

- **Consistency and standards**: Generic platform conventions will be used so the user will know when different words, situations or actions mean the same thing
- **Error prevention**: Error-prone conditions will be eliminated, and users will be presented with a confirmation option before they commit to potentially dangerous actions
- **Recognition rather than recall**: Objects, actions, and options will be visible enough, in order to avoid the user the burden of remembering a lot of information
- **Flexibility and efficiency of use**: Whilst the GUI will allow non-expert users, the use of "accelerators" by expert users will satisfy their needs.
- **Aesthetic and minimalist design**: Dialogues will be made bearing in mind that they should not contain information which is irrelevant or rarely needed
- **Help users recognize, diagnose, and recover from errors**: Error messages will be expressed in plain language, instead of technical codes, constructively suggesting a solution when possible
- **Help and documentation**: Concise and well documented help will be available in the GUI.

4.5 Community-oriented interface

The holistic GUI is going to be used by each SOA4ALL user individually. However, the functionalities that it is going to support will have a strong focus on collaboration. The GUI will have to reflect this idea and stress it conveniently. The UI will therefore have to support:

- An easy way of exchanging information between two partners and also between a whole community of users
- Possibilities of rating content and of viewing ratings and judgements of other users
- Possibilities of commenting and tagging elements
- Easy ways of making elements that have been created with SOA4ALL accessible for other users by defining access rights

4.6 Context-awareness

As was addressed in Section 3.1.2, Context is a key factor in SOA4All and the idea of adaptation based in contextual factors is everywhere within the project. It is therefore expected that the Context Adaptation Framework will be used to adapt the GUI, offering the users a personalised interface based in factors such as their preferences or community shared knowledge, etc.

4.7 Design Principles

SOA4ALL will follow other important design principles in the design of the GUI adopted from [Yusof, et al. 2004]:

- **Simplicity**: Don't compromise usability for function
- **Support**: User is in control with proactive assistance
- **Familiarity**: Build on users' prior knowledge
- **Obviousness**: Make objects and their controls visible and intuitive
- **Encouragement**: Make actions predictable and reversible

- **Satisfaction**: Create a feeling of progress and achievement
- **Accessibility**: Make all objects accessible at all times
- **Safety**: Keep the user out of trouble
- **Versatility**: Support alternate interaction techniques
- **Personalization**: Allow users to customize
- **Affinity**: Bring objects to life through good visual design

5. Other Non-functional Requirements

The holistic GUI addressed in this deliverable will also tackle other non-functional generic requirements, in terms of performance, safety, security, etc.

Firstly, from the performance point of view, as SOA4All is proposing a paradigm where billions of services are deployed and consumed by an extremely large number of users, the holistic GUI has to be scalable and reliable, regarding the large number of simultaneous connections that it might need to handle.

Particularly, fast reaction time as a specific requirement for the holistic GUI needs to be highlighted. SOA4All users will need to use the UI to perform various activities such as creating or composing Web Services, and this will require a lot of UI interaction. In order to provide a usable UI it is therefore necessary to provide a fast and desktop-like reaction of the web interface.

In terms of safety, such a large number of users and services will imply the necessity of defining safeguards for unplanned problems such as the loss of data. Due to the distributed nature of SOA4All and the underlying technologies below the GUI, there will be enough safety mechanisms regarding services, but in addition, versioning and backup mechanisms will have to be taken into account for the GUI.

Regarding security and privacy concerns; the GUI will be designed so that the proper user identity authentication mechanisms are used, in order to avoid unexpected identity problems. Also mechanisms for handling public and password-protected areas will be necessary.

6. Frameworks and Tools

6.1 Web applications

A web application runs entirely from within a browser, such as Mozilla Firefox or Microsoft Internet Explorer. Web applications are becoming increasing more popular because they offer many advantages over traditional desktop applications as assessed below.

- Advantages of Web Applications:
  - No special configuration or changes are need on users PCs
  - Lower costs
  - Centralised data is secure and easy to backup
  - Updates can be made quickly and easily
  - Information is accessible to a wide audience anywhere in the world
  - Access time - 24 hours a day, 7 days a week
  - Everybody has a browser
  - On-line training can be completed at user's own time and place
  - Always up-to-date
- Disadvantages of Web Applications:
  - Slower as run over internet
  - Interfaces not as sophisticated
  - Can take longer to develop as more complex
  - Security risks

Web applications are especially suitable for environment with millions of users with vast heterogneity of clients and platforms.

6.2 Rich Internet Applications (RIAs)

Rich Internet Applications (RIAs) are web applications that have the features and functionality of traditional desktop applications combined with advantages of web applications. RIA frameworks, such as Google's Web Toolkit (GWT), solve most of the disadvantages of web applications. They have sophisticated interface, run fast and responsive and provide tools for rapid development. RIA's typically transfer the processing necessary for the user interface to the web client but keep the bulk of the data (i.e., maintaining the state of the program, the data, etc.) back on the application server.

RIAs typically allow the following:

- Run in a web browser, or do not require software installation and browser add-ons
- Run locally in a secure environment called a sandbox.

Based on the fact that RIAs combine the advantages of desktop and web applications it can be said that the only chance of SOA4ALL to fulfill requirements defined in the preveous chapters is using a RIA technology.

An analysis of further RIAs is detailed below.

6.3 Overview about current RIA frameworks

6.3.1 Google Web Toolkit 1.5 (GWT) & EXT GWT [GWT]

Google released the 'Google Web Toolkit' or GWT in 2006 which allows the development and testing of JavaScript-based AJAX RIAs using the Java language. The GWT programming paradigm centres around coding user interface logic in Java (similar to the Swing/AWT model), and then executing the GWT compiler to translate this logic into cross-browser-compatible JavaScript. Designed specifically for Java developers, GWT enables Java programming, refactoring, debugging and unit testing of RIAs using existing tools (e.g. Eclipse), without requiring knowledge of JavaScript or specific browser DOM irregularities (although hand-written JavaScript can still be used with GWT if desired).

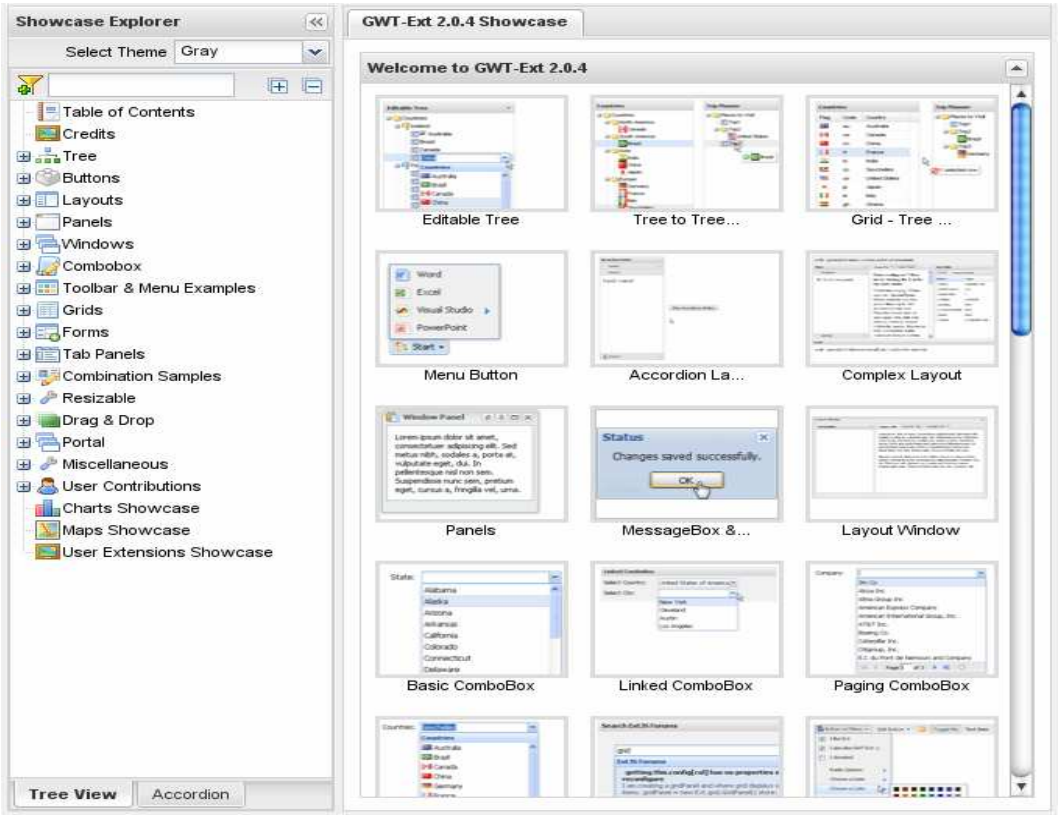


Figure 1: GWT-EXT Showcase

Pros:

- License - Apache License, v2.0 (GWT) & Dual License Model with GPL v3.0 or Commercial Licenses (EXT GWT)
- Browser Support IE, Firefox, Safari, Opera
- Do not need browser add-ons
- API – Java
- Very Strong Community, active and fast maintenance
- Very good documentation
- Excellent and free development tools
- Easy to write and debug

- Easy to extend
- Production mature

Cons:

- Relatively large library, which might result in a longer loading time for non-broadband users when visiting the RIA for the first time.

GWT is an excellent web toolkit which has made huge progress within avery short time since it started in 2006. It is open source and supports all popular browsers. There are many extensions to GWT that makes it richer and a strong working group within Google that intent to make GWT better. It can be run both on a browser and on a desktop using "Google Gears".

All these advantages makes GWT an excellent candidate for implementing large, rich and scalable user interfaces without need to by licenses and for free.

6.3.2 Adobe Flex 3.0 [Flex]

Flex is a free, open source framework for building highly interactive, expressive web applications that deploy consistently on all major browsers, desktops, and operating systems. It provides a modern, standards-based language and programming model that supports common design patterns. MXML, a declarative XML-based language, is used to describe UI layout and behaviours, and ActionScript 3, a powerful object-oriented programming language, is used to create client logic. Flex also includes a rich component library with more than 100 proven, extensible UI components for creating rich Internet applications (RIAs), as well as an interactive Flex application debugger.

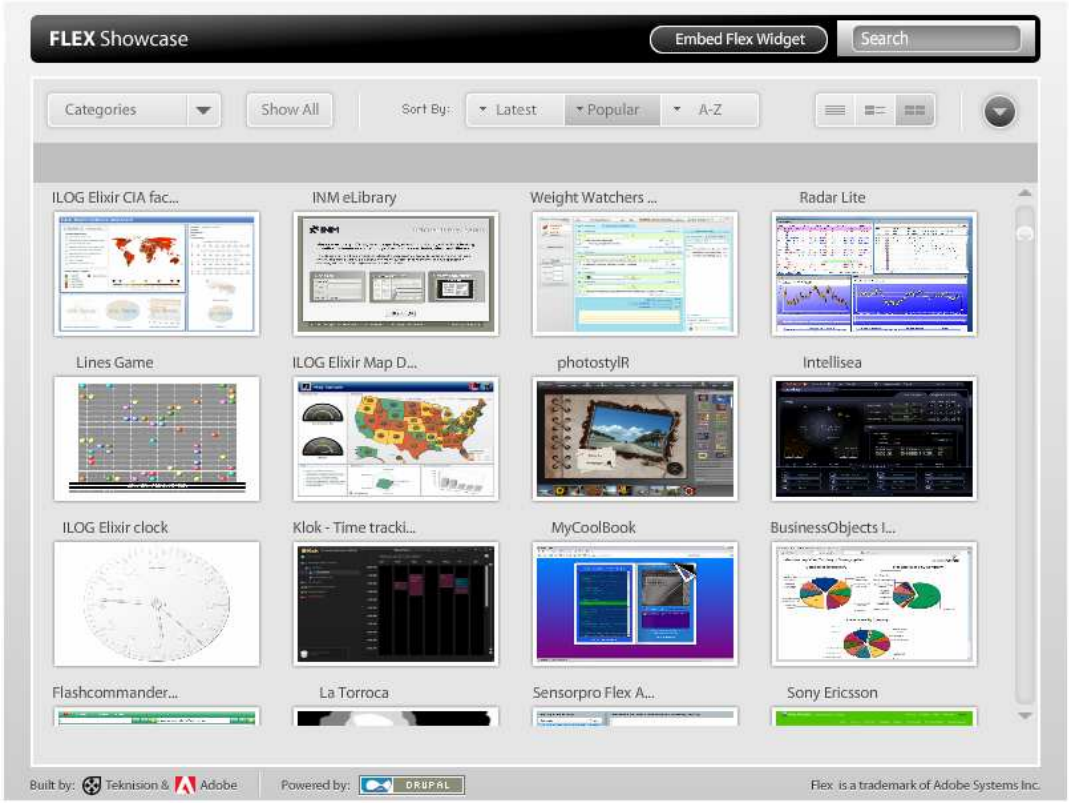


Figure 2: Flex Showcase

RIAs created with Flex can run in the browser using Adobe Flash® Player software or on the desktop on Adobe AIR™, the cross-operating system runtime. This enables Flex applications to run consistently across all major browsers and on the desktop.

Pros:

- API – XML
- Browser Support IE, Firefox, Safari, Opera, others
- Very Strong community
- Very good documentation
- Production mature and proved
- Easy to extend

Cons:

- License – Proprietary
- Need to install browser add on – flash player
- Commercial development tools (Flex Designer)

Adobe Flex is a presentation technology which runs both on a web browser (with installed Adobe Flash Player) or on a desktop through “Adobe Air”. It has a proved track record of perhaps thousands of successful web sites with rich user experience. Flex can run on any browser with installed Adobe Flash Player on it on any platform. There are many third party components developed from the huge and strong community of Adobe. Its production maturity has been proven in lots of heavy commercial sites. Unfortunately Flex is not open source and only Adobe Flash Player is free to use. All other tools and products are commercial.

Flex is maybe the best and proved technology for rich internet applications so far. However proprietary license and commercial tools and bad learning curve make it not very appropriate for non commercial applications.

6.3.3 Yahoo! User Interface Library 2.5.x (YUI) [YUI]

The Yahoo! User Interface (YUI) Library is a set of utilities and controls, written in JavaScript, for building richly interactive web applications using techniques such as DOM scripting, DHTML and AJAX.

Pros:

- License – BSD
- Browser Support IE , Firefox, Safari, Opera
- Do not need browser add-ons
- API – JavaScript
- Strong Community
- Very Good documentation
- Production Mature

Cons:

- Lack of good development tools
- Not easy to debug

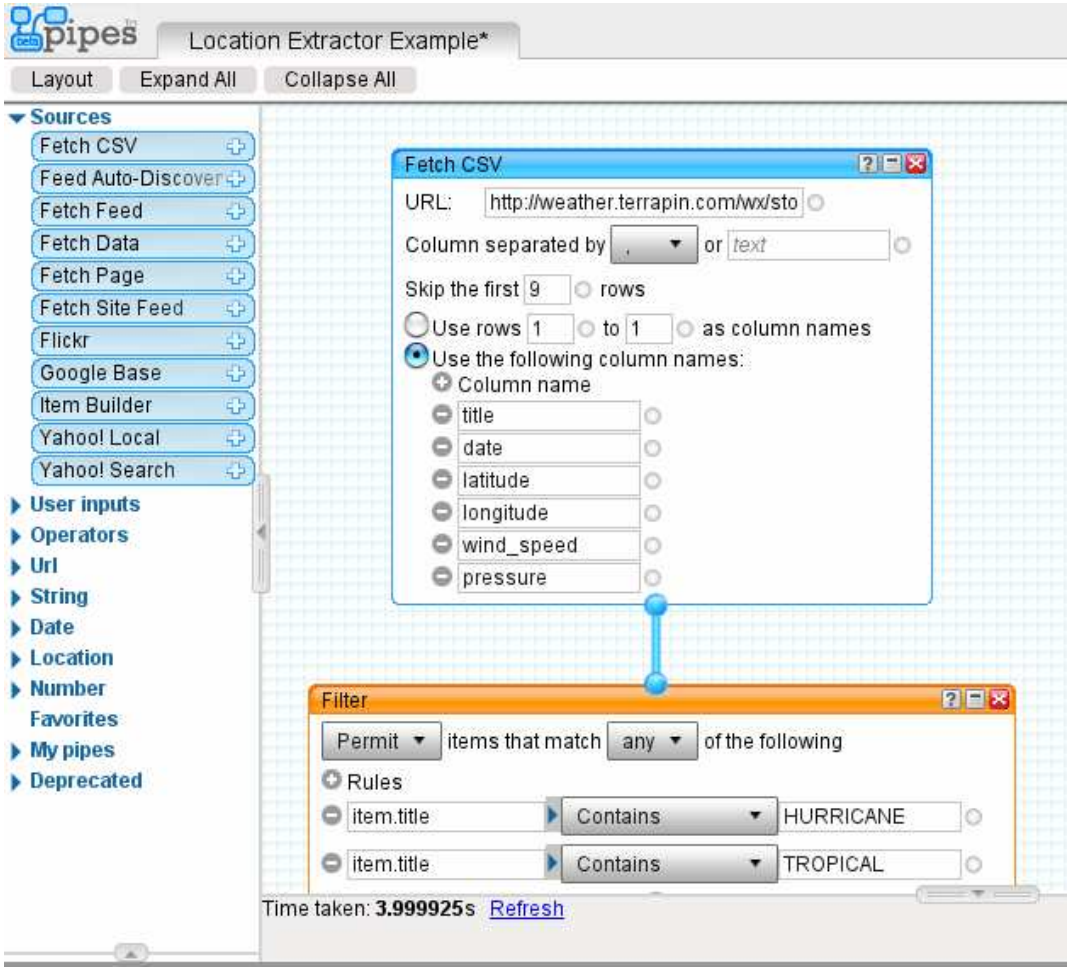


Figure 3: Yahoo! Pipes Editor

Yahoo User Interface library is young and good for applications that are already running on HTML to enrich them with better user experience. It is open source and has strong community and good documentation. It supports all major browsers and is cross platform. There are many of production application running using YUI. YUI has one major disadvantage – it uses a JavaScript API and the tools for development and debugging of javascript are not very good.

6.3.4 SUN JavaFX [JavaFX]

JavaFX is a family of products for creating RIAs with immersive media and content across the multiple screens of your life. It includes a runtime and a tools suite that web scripters, designers, and developers can use to quickly build and deliver the next generation of rich Internet applications for desktop, mobile devices, TV, and other platforms. JavaFX tools suite offers developer and authoring tools that bridge the gap between user experience design and development logic, giving designers and developers good collaboration opportunities.

Figure 4: SUN JavaFX Showcase

Pros:

- License - GPL v2

Cons:

- API - JavaFX Script
- Need to install browser add on – JavaFX runtime
- Not released yet, not mature, experimental, currently in development

JavaFX is a very young technology that is not yet production mature (and will not be soon). It requires sandbox to run on browsers and currently there is not a good community and documentation.

6.3.5 Microsoft Silverlight 2.0 [Silverlight]

Microsoft Silverlight is a cross-browser, cross-platform, and cross-device plug-in for delivering the next generation of .NET based media experiences and rich interactive applications for the Web.

Figure 5: Microsoft Silverlight Showcase

Pros:

- API - C#, VB, C++, IronPython, IronRubi ...
- Good documentation
- Strong community
- Production mature

Cons:

- Browser Support - Only Windows and OSX – IE, Firefox, Mozilla, Safari, Opera
- License – Proprietary
- Need to install browser add on – Silverlight runtime
- Commercial development tools (Microsoft Visual Studio)

Like Flex and JavaFX Silverlight runs on a browser sandbox which should be installed as a browser add on. Silverlight is a very expressive and rich presentation framework mainly oriented to Microsoft developers used to Microsoft development tools like Visual Studio. There are many internet applications running on Silverlight and it is production mature. Unfortunately Microsoft only support browsers on Windows and OSX so far and do not support browsers under Linux platforms. There is open source implementation of Silverlight – Moonlight from Novel that can run on Linux. An advantage of Silverlight is that it supports many different languages for development. There are many tools coming from both, free and commercial domains.
Although not as popular as Adobe Flex in combination with Flash, Siverlight is an excellent and promising RIA platform supported by Microsoft. Unfortunately although it is claimed that Silverlight is a cross platform technology, it is mainly focused on the Windows world and still not implemented for some of the major platforms, which is a major drawback.

6.3.6 Openlaszlo 4.0.12 [Laszlo]

OpenLaszlo is an open source platform for the development and delivery of rich Internet applications. It is released under the Open Source Initiative-certified Common Public License. The OpenLaszlo platform consists of the LZX programming language and the OpenLaszlo Server:

LZX is an XML and JavaScript description language similar in spirit to XUL, MXML, and XAML. LZX enables a declarative, text-based development process that supports rapid prototyping and software development best practices. It is designed to be familiar to traditional web application developers who are familiar with HTML and Javascript.

The OpenLaszlo Server is a Java servlet that compiles LZX applications into executable binaries for targeted run-time environments.

OpenLaszlo is the only rich Internet application platform examined which is capable of compiling into two different runtimes from the same code base (although GWT will compile into browser-specific java script from Java).

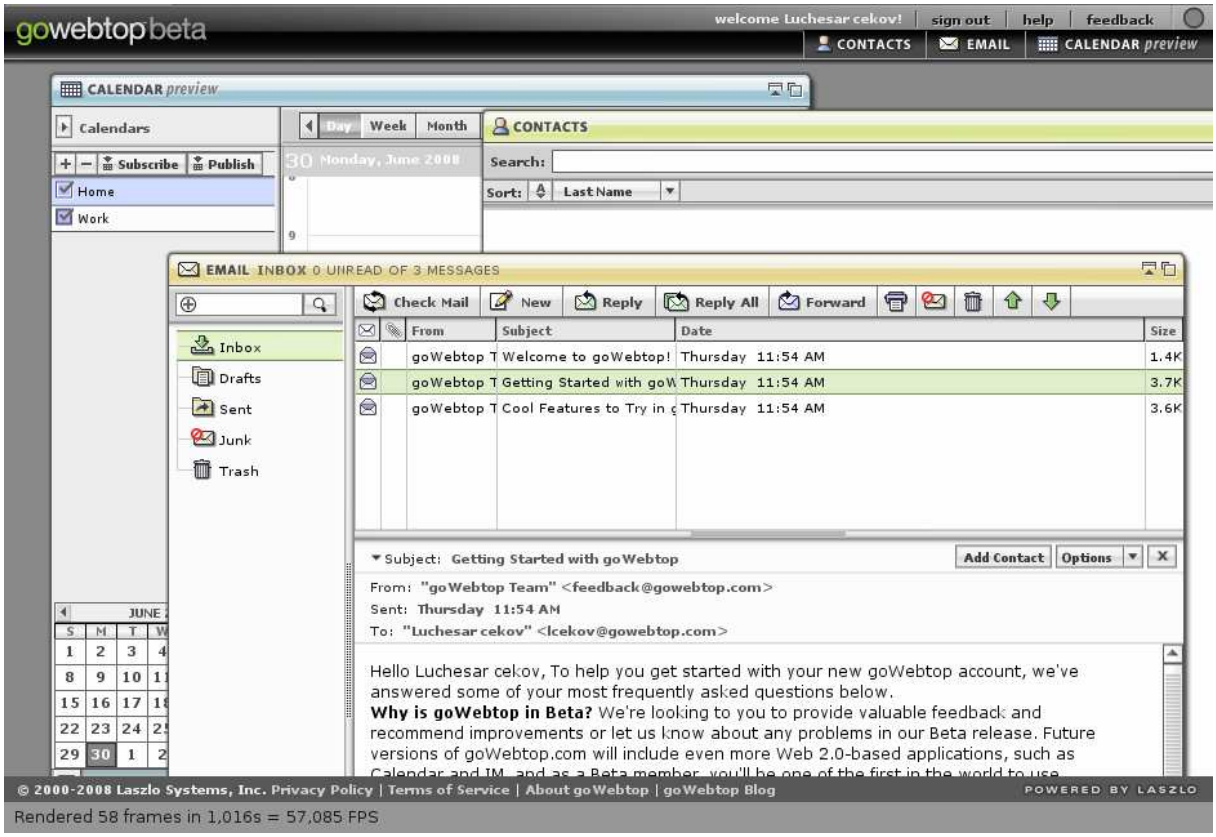


Figure 6: Open Laszlo Webtop

Pros:

- License – CPL 1.0
- Browser Support IE , Firefox, Safari, Opera
- Do not need browser add-ons
- API – XML
- The code can be compiled either to AJAX or Flash

- Good Community
- Very Good documentation
- Production Mature

Cons:

- No development tools
- Not so easy to debug
- Not easy to extend

OpenLaszlo has many pros except for good (and free) development tools. It is the only RIA technology that can compile to both Flex and AJAX - DHTML. It supports all major browsers for the DHTML version and all browsers with flash support. There is a large community and excellent documentation. There are many RIAs that uses OpenLaszlo in production environments. There are many OpenLazslo third party libraries and extensions. As OpenLaszlo is Java based it can run on any platform.

6.3.7 Eclipse Rich Ajax Platform (RAP) [RAP]

The RAP project enables developers to build rich, Ajax-enabled Web applications by using the Eclipse development model, plug-ins with the well known Eclipse workbench extension points, JFace, and a widget toolkit with SWT API using a technology called “qooxdoo” (see [qooxdoo]) for the client-side presentation.

The underlying Ajax toolkit - qooxdoo is a comprehensive and innovative Ajax application framework. Leveraging object-oriented JavaScript allows developers to build cross-browser applications. No HTML, CSS nor DOM knowledge is needed.

It includes a platform-independent development tool chain, a state-of-the-art GUI toolkit and an advanced client-server communication layer. It is open source under an LGPL/EPL dual license.

Pros:

- License – EPL
- Browser Support IE , Firefox, Safari, Opera
- Do not need browser add-ons
- API – Java + SWT + Jfaces
- Excellent tools
- Easy to debug
- Easy to extend

Cons:

- Bad Documentation
- Weak Community
- Not production mature

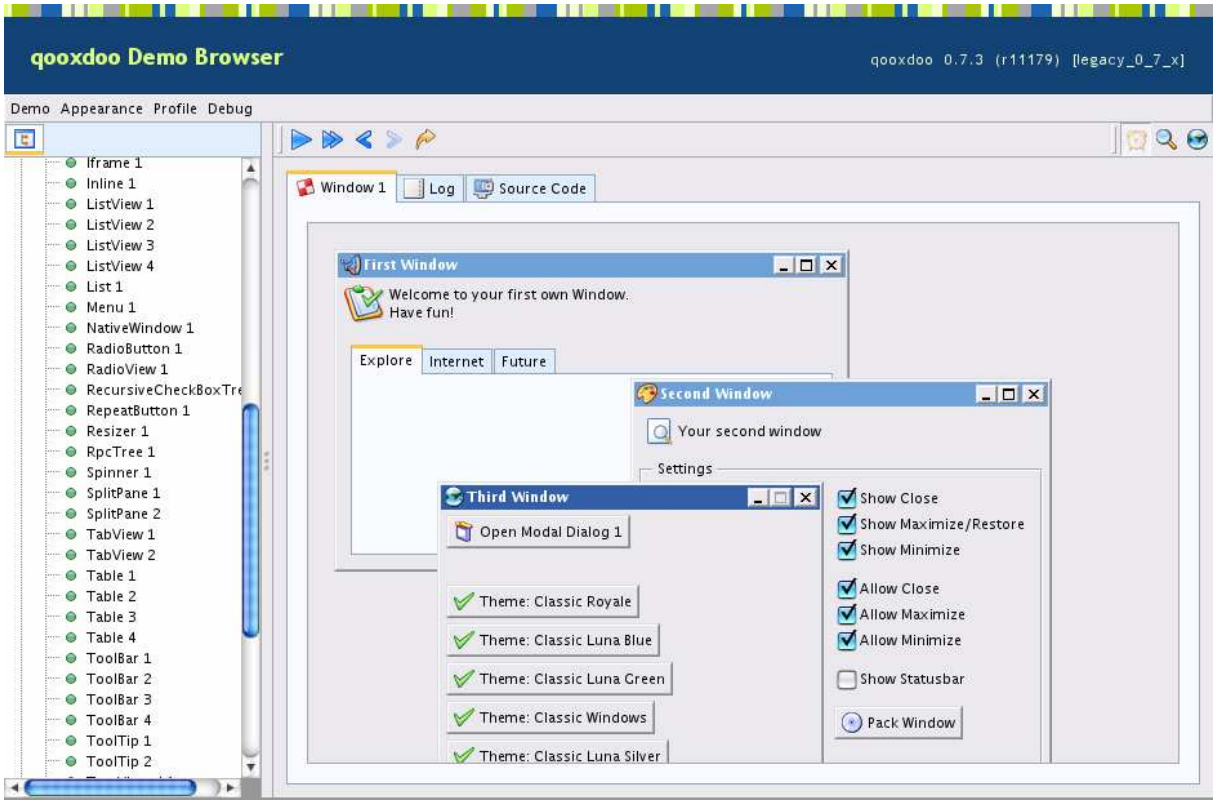


Figure 7: Qooxdoo Demo Browser

RAP has just bean introduced by Eclipse Foundation and although very promising is not mature and is definitely not production ready. The documentation that covers several examples and is yet to be developed. There is no good community.

Being one of the best development tools Eclipse + SWT + JFaces is proven UI paradigm and it is hoped RAP will mature quickly

6.3.8 Bindows

Bindows (abbreviation of “Browser Windows”) is a Graphical User Interface (GUI) software framework for development of rich AJAX and Web 2.0 enterprise web applications. The most promising feature is that it can generate the exact look-and-feel of a Windows GUI on web pages. Bindows object-oriented approach and expandability scale well to very large development teams and projects.

This framework is integrated with IntelliJ Idea IDE, through a specific plug-in.

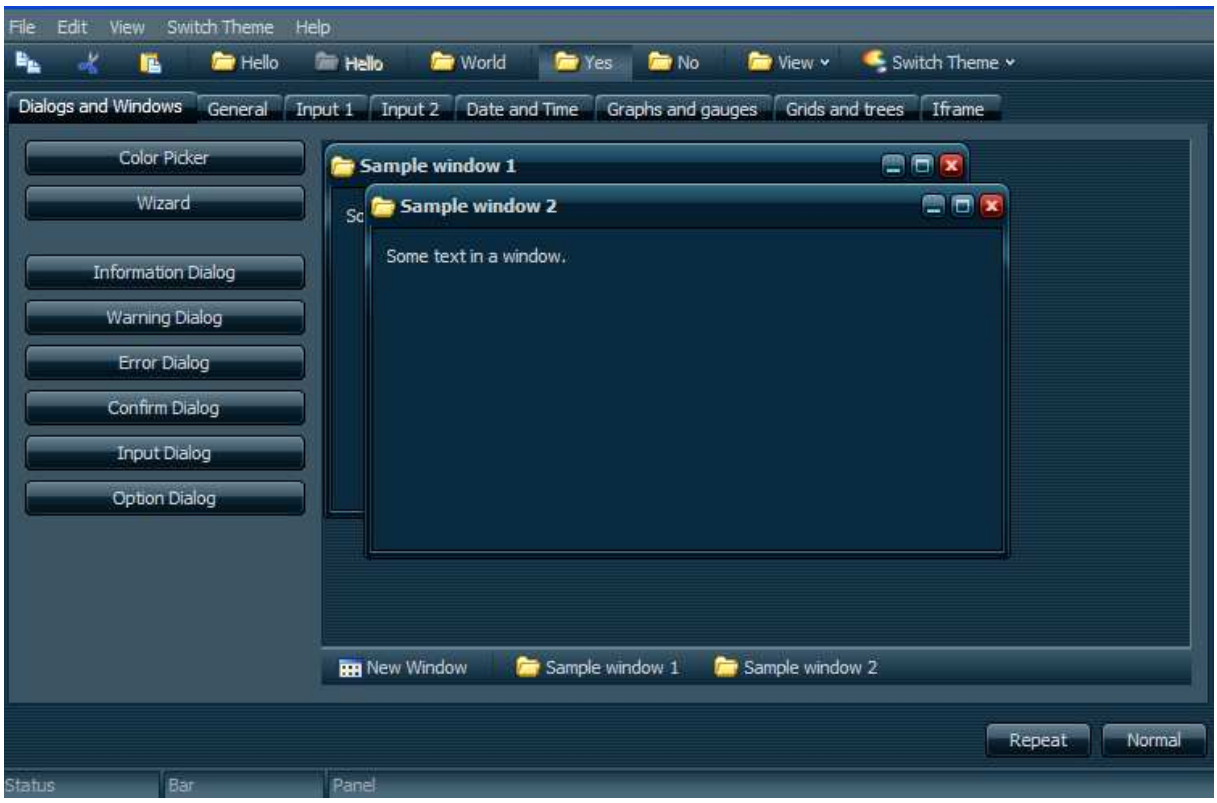


Figure 8 – Sample Bindows web application

The new Bindows 4.0 (beta 0806) includes four major additions:

- Enable development of complete Ajax applications without writing a single line of JavaScript
- A completely redesigned menu system
- Brand new looks - 6 new themes and a powerful theme engine (so you can create your own)
- Initial support for new browsers (IE8, FF3, Safari)
- BindowsFaces™ - Enabling JSF to produce full Bindows applications

Pros:

- Using the Bindows technology it is possible to migrate existing web applications and web pages to be 508-compatible (even Ajax applications with very rich GUI).
- Fully Object Oriented coding (javascript)
- API – Javascript + XML
- IntelliJ Idea plug-in
- Browser Support IE , Firefox, Safari
- JSF module
- Use of themes
- All Bindows components are designed for extension

- Do not require software installation on the client side

Cons:

- Commercial development using Bindows requires a purchase of the Bindows Framework Developer's Kit.
- Commercial and Government use of a Bindows runtime software requires a Bindows Deployment License

The Graphical User Interface generated by Bindows is astonishing and looks truly like a common-use desktop application, the question is if such a potential complexity is something useful to the end-client.

6.3.9 Others

There are several additional web toolkits, which could also be considered. Toolkits that can be mentioned included:

- **ZK -** http://www.zkoss.org/
- **Dojo -** http://dojotoolkit.org/
- **Rico -** http://openrico.org/rico/home.page
- **Echo -** http://echo.nextapp.com/site/
- **Mochikit -** http://www.mochikit.com/
- **Rialto -** http://rialto.improve-technologies.com/wiki/

Since space and time is limited, the SOA4ALL team focused on analyzing the most promising technologies in depth instead of giving a superficial overview about all toolkits. We therefore did focus on the toolkits described in the last sections.

6.4 Comparison and selection

Framework	Documentation	Community	Write/Debug	Tools	Comments
GWT & EXT GWT	+++	+++	++	+++	
Adobe Flex	+++	+++	+	-	Proprietary License
YUI	+++	++	---	---	
JavaFX	--	--	---	---	Needs Browser Plugin
Silverlight	+++	+++	++	+/-	Platform Dependencies

Openlazlo	+++	++	---	---	
RAP	---	--	+++	+++	
Bindows	--	++	--	++	Proprietary License

There is plenty of good Web 2.0, RIA technologies in the market.

The best sandbox based approachesare Adobe Flex and Microsoft Silverlight. Both Adobe and Microsoft will try to gain as much market share as possible and make their products better to attract more developers. Both have large communities and third party libraries.

The best AJAX and DHTML based RIAs are GWT and OpenLaszlo. Both are mature enough and popular to server needs of enterprise applications and even on-line games. GWT is more oriented to ease of development and debugging then Laszlo.

At the end the decisions of which RIA technology to be used in SOA4ALL won't be so trivial because there are so many options.

Based on the analysis above it seems GWT is a good choice. The main advantages are the easy development and debug of AJAX based web 2.0 applications. There are many libraries that can serve any type of user interface and the best is GWT seams to get better very fast.

7. Core UI Framework Architecture

7.1 Approach

SOA4ALL is realized by a group of partners collaborating in a physically distributed environment. All sub-parts of the various work packages will focus on a slightly different aspect of the SOA4ALL vision. This means that it is of major importance for the UI to provide a possibility of integrating the effort and libraries of different teams into a holistic user interface. Such a holistic user interface has many advantages:

- **User perspective:** It allows users to use SOA4ALL with one common look and feel allowing them to seamlessly use the different parts of SOA4ALL without having to learn different user interfaces at the same time. Instead of this, users will be able to control all parts of SOA4ALL from one common UI, which integrates all work package environments.
- **Developer perspective:** It allows developers to focus on the actual core part of their work instead of having to implement core functionalities such as user management for each and every work package separately.
- **Management perspective:** It provides a single point of entrance to the SOA4ALL world allowing people to easily locate all parts of the project by using one central user interface that links or integrates all other parts. However, the different parts might still run on different physical machines as necessary/desired but they will be accessible via one user interface.

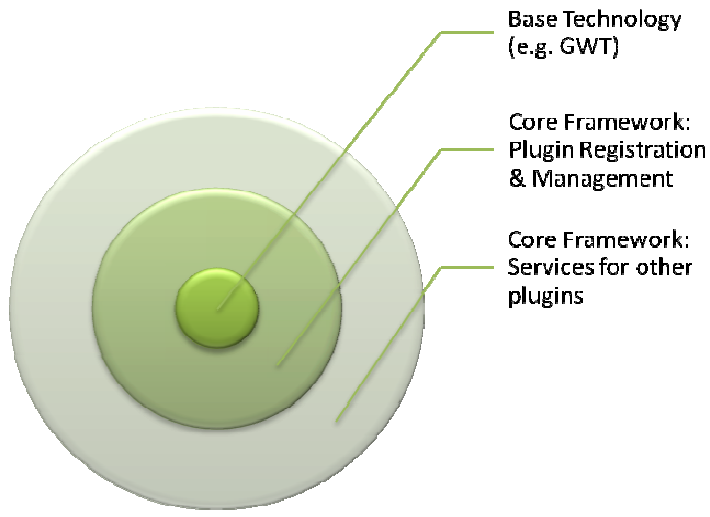


Figure 9: Core UI Framework

In SOA4ALL, such a holistic user interface will consist of three “levels” each based on each other:

- **Base Technology**
  - This provides the base functionality used by SOA4ALL. This base functionality is provided by using an existing Rich Internet Application (RIA) framework as discussed in section 6.3. As the result of 7.2, an Ajax based solution (GWT) will be used to provide browser independent web interface functionalities.
  - Examples: Typical elements that will be provided by this layer are core elements for creating checkboxes, menus and list items.

- **Core UI Framework**
  - Plugin Registration & Management

    SOA4ALL will implement a core UI framework that is based on the RIA framework that has been selected. This core UI framework will be extended by plugins as described in the next section.

    Examples: This layer will provide functionalities for adding plugins for e.g. service composition or service monitoring.

  - Service Layer for other Plugins

    The core UI framework will provide a rich set of common functionalities needed by most SOA4ALL work packages. Those functionalities will be described in detail in section 7.3.

    Examples: This layer will provide functionalities such as user management and error handling.

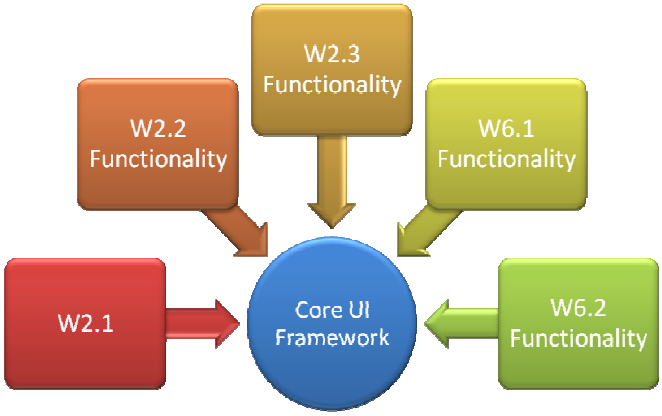
- **Specific User Interfaces**

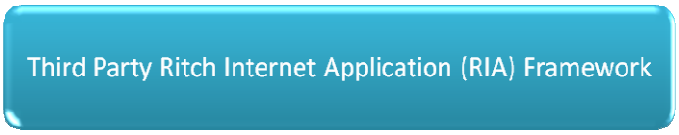
  Based on the core UI framework of SOA4ALL, work packages can create their work package specific user interfaces. Those user interfaces will be used for allowing users to perform the actual features provided by the work packages.

  Examples: For example, those plugins will provide a specific UI for service composition or service monitoring.

7.2 Integration of Work Package Specific Functionalities

The user interface will consist of an SOA4ALL core UI framework that provides an interface allowing other UI elements to plug themselves into the SOA4ALL UI in an easy way. This plugin concept allows SOA4ALL to easily add new functionalities and to keep the user interface flexible and open for new functionalities. The following diagram shows the main idea of this architecture:





UI components are registered at the core framework on design time and will be able to use common services provided by the core UI framework such menu and access rights functionalities. For this purpose, the core UI framework will provide a plugin registration interface for new plugins that want to add themselves to the user interface (registration) or that wish to remove themselves.

Once this registration process has been performed, the core UI framework will allow plugins to use a wide set of functionalities that is provided by for achieving common tasks such as user management and access rights definition. Plugins are free to use those functionalities although they may also extend from the classes that are provided or they may choose to use their own implementation if necessary. A detailed set of functionalities provided by the core UI framework is listed in the upcoming subsection.

The SOA4ALL core UI will allow plugins to define dependencies. For example, a plugin A might require another plugin B in order to work properly. Those dependencies can be described in a specific format and will be checked by the core UI framework on plugin registration time.

In addition to this, the SOA4ALL core UI framework will allow plugins to locate other plugins that have been registered already and it will allow them to exchange information either directly or via a simple message bus that is provided by the core UI framework.

7.3 Core Functionalities

The following sections describes the general services and functionalities that will be provided by the Core UI framework. As described in the last subsection, those functionalities are not mandatory meaning that plugins are free to extend them or to even replace some of them with own functionalities.

7.3.1 Structured Programming Model

The Core UI framework will provide a specific set of interfaces that can be accessed by plugins. It will be developed in a high-level programming language (Java) and will use Java specific functionalities for events and data types. This process is strongly influenced by the RIA framework that has been selected in section 6.3 (GWT). The programming model will prevent the necessary for writing “unclean” code and it will reduce the necessary for writing HTML or Javascript parts to a minimum.

7.3.2 Plugin Management and Extendibility

As described in the last subsection, an interface will be provided that allows work packages to add their functionalities as plugins. Those plugins will be able to register themselves and to describe dependencies to other plugins.

7.3.3 Accessibility

The Core UI and all plugins based on it will be realized as a browser based application. This means that the SOA4ALL components will be publically accessible via the web using a standard internet browser such as Firefox or Internet Explorer. - What might be required though is a relatively fast internet connection depending on the graphical elements or the complexity of the different plugins.

7.3.4 Dashboard and Menu Functionalities

The SOA4ALL Core UI will provide a dashboard. This dashboard acts as an overview page

showing the overall system status and the registered plugins. It also shows a summary of messages for users and a list of links to be used for entering the different work package specific areas.

7.3.5 Look and Feel Guidelines

In order to provide a consistent user interface experience to SOA4ALL users, a set of user interface guidelines and templates will be provided as well as a set of examples showing how to use them. A joint stylesheet (CSS) will also be suggested in this context. The goal is to allow work packages to create a similar look and feel on all areas of the project.

7.3.6 Holistic User Management

User Management is a core functionality that is needed by almost all work packages. The core UI framework will provide a simple user management allowing the definition of users and the management of user data (metadata).

7.3.7 Security, Access Rights, Trust

The core UI framework will provide functionalities for defining public and password protected areas. It will also allow plugins to define access rights and to check if the current user has one of those access rights. In addition to this, a simple trust model will be applied by allowing plugins to rate specific entries and to receive the ratings that have been performed.

7.3.8 High-Level Monitoring and Notification

Plugins that have been installed will need to implement a small ping method. This ping method is used to continuously monitor the status of all plugins. In case of a plugin failure the plugin administrator will be automatically notified via email.

7.3.9 Auditing

The core UI framework will allow plugins to log specific activities. Those log messages may be viewed in the core UI part.

7.3.10 Preferences and Settings-Management

Plugins may have to store preferences about various things such as URLs for web services or the plugin-specific configurations. The core UI framework will provide a functionality of setting and reading plugin specific preferences.

7.3.11 Information exchange

A simple message bus component will be provided to exchange messages between different plugins. This will be limited for exchanging text (strings) but it may also be used to receive information about the location of a specific plugin (in order to allow a direct plugin-to-plugin communication).

7.3.12 Messaging

A simple way of storing messages for users will be provided as well. This will allow plugins to send messages to SOA4ALL users. The framework will display those messages on login time and it will provide a functionality for users to be notified if new messages arrive.

7.3.13 Versioning and Archiving

The core framework will provide a simple way of storing old data in an archive. This archive may be used by plugins to store multiple versions of files in order to keep a backup of old entries.

7.3.14 Low-Level Error Handling

The core UI framework will provide low-level error handling. This means that program exceptions and errors will be caught by the framework. Users will see an appropriate error message and will be given the chance to inform the plugin administrator about this error.

8. Conclusions

This deliverable has given an overall summary of the current UI concept that is to be applied in SOA4ALL. It has been created as an extra deliverable in order to present all content as a holistic concept.

It has been decided to create such a holistic user interface approach in order to be able to present a common look & feel to SOA4ALL users. This is of major importance for the usability of SOA4ALL and it certainly the base for really opening the project to the masses by lowering the entrance barrier with a seamless user concept.

The document has been able to specify main requirements that are necessary for a holistic UI concept. This includes Human Computer Interaction, External Interface Specifications, System Features and Non-functional Requirements.

In addition to this, tools and frameworks have been defined that will be used for the implementation. A specific solution has been highlighted and recommended to form the base for the UI implementation.

The core UI specification has defined the services that will be provided to work package specific plugins. This allows work packages to concentrate on the core development of their parts in SOA4ALL without having to invest a major amount of time for base functionality such as user management or access rights.

9. References

- Booch, G., I. Jacobson and J. Rumbaugh, *The Unified Modeling Language User Guide.* Addison-Wesley, 1999
- Sutcliffe, A. and Mehandjiev, N. 2004. Introduction: Special Issue on End User Development. *The Communications of ACM* 47, 9 (Sep. 2004), 31-32.
- N. Mehandjiev and L. Bottaci. User-enhanceability for organizational information systems through visual programming. In P.Constantopoulos et al editors, Advanced Information Systems Engineering: 8th International Conference, CAiSE'96, Lecture Notes in Computer Science No 1080, pages 432-456. Springer-Verlag, 1996.
- N.C.Shu. Visual Programming.Van Nostrand Reinhold Company Inc. New York 1988
- Alan F. Blackwell. Metaphor in Diagrams. PhD Thesis. University of Cambridge. September 1998.
- Nardi, B.A.. A small matter of programming: Perspectives on end user computing. Cambridge, MA. MIT Press. 1993
- Sutcliffe, A., Lee, D., Mehandjiev, N. Contributions, Costs and Prospects for End-User Development, Proceedings of HCI International 2003, Lawrence Erlbaum Associates, Inc. New Jersey, USA
- N Mehandjiev, A Sutcliffe and D Lee, Organisational View Of End-User Development, in H Lieberman, F Paterno, and V Wulf, eds, End User Development, Human-Computer Interaction Series , Vol. 9 2006, XVI, 492 p., Hardcover ISBN: 1-4020-4220-5
- Green, T. R. G. (1989) Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay (Eds.) People and Computers V. Cambridge, UK: Cambridge University Press, pp 443-460.
- Green, T. R. G. & Petre, M. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. in J. Visual Languages and Computing, 7, 131-174. 1996
- Myers, B. A., Pane, J. F. and Ko, A. (2004). Natural Programming Languages and Environments. in Sutcliffe and Mehandjiev, eds. Communications of the ACM, special issue on End-User Development, September, 47, 9, 47-52.
- Johannes Maria Zaha, Alistair P. Barros, Marlon Dumas, Arthur H. M. ter Hofstede: Let's Dance: A Language for Service Behavior Modeling. OTM Conferences (1) 2006: 145-162
- Martinez, A., Patino-Martinez, M., Jimenez-Peris, R., and Perez-Sorrosal, F. 2005. ZenFlow: A Visual Web Service Composition Tool for BPEL4WS. In Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (September 20 - 24, 2005). VLHCC. IEEE Computer Society, Washington, DC, 181-188. DOI= http://dx.doi.org/10.1109/VLHCC.2005.74
- Nielsen, J.: Ten Usability Heuristics, http://www.dsoergel.com/794/NielsenUsability.pdf, ISSN 1548-5552, 2005
- Yusof, R.J.R. Amin, R. Zainudin, R. Baker, O.F: Human computer interaction (HCI) - aspect in developing information access system, In: Proceedings of the TENCON 2004. 2004 IEEE Region 10 Conference, 2004

10. Weblinks

[Flex] http://flex.org/

[GWT] http://code.google.com/webtoolkit/

[JavaFX] http://www.javafx.com/

[Laszlo] http://www.openlaszlo.org/

[qooxdoo] http://qooxdoo.org/

[RAP] http://www.eclipse.org/rap/

[Silverlight] http://silverlight.net/

[YUI] http://developer.yahoo.com/yui/