

White Paper on
Service Front-Ends in the Future Internet of Services and Next Generation SOA
Version 0.5, July 21st 2008

Editors: J. M. Cantera, M. Reyes, J. Hierro, N. Tsouroulas. Telefónica I+D

Note from the editors

The current version of this white paper should be considered a preliminary draft. Version 1.0 of the white paper will be published by July 31st. Readers are encouraged to download and review version 1.0 as soon as it becomes available.

Introduction

Service Oriented Architectures (SOA) have attracted a great deal of interest during the last years. In fact SOA increase asset reuse, reduce integration expenses and improve businesses agility in responding to new demands.

Nonetheless, until now, the mainstream development and research around SOA have been focused mainly on middleware and scalability, service engineering and automating service composition using BPM technologies. Little or no attention has been put on service front-ends which are, in our vision, a fundamental part of SOA. As a consequence SOA remain on a technical layer hidden to the end-user.

The evolution of web-based interfaces is a testimony of the progress achieved to improve service usability. However, existing, web-based service front-ends, are far from completely meeting end-user expectations. Applications and information portals still are based on monolithic, inflexible, non-context-aware, non-customizable and unfriendly UIs. As a consequence end-users do not really benefit from the advantages promoted by service orientation in terms of modularity, flexibility and composition. In addition service front-ends are constructed in an ad-hoc manner and with the absence of formal engineering methods and tools that could accelerate the time to market.

This position paper presents our vision with respect to the next generation, user-centric service front-end technology. We present a series of guiding principles which promote the adoption of SOA front-ends that empower end-users and fully exploit Context and Knowledge. Our proposal will make end-users appreciate the benefits of SOA, fostering composition, loose coupling and reuse on the front-end layer.

We start the document with the problem statement about service front-ends. Then we present the guiding principles that drive our quest to address the limitations of existing service front-end technology. We carry on with a discussion on how to materialize the guiding principles. A novel reference model and architecture that empowers end-users and supports this vision is then proposed. This architecture will enable the creation of new ecosystems where all stakeholders will be able to collaboratively develop capabilities and innovate new operating procedures by mixing and integrating already available services. Finally, we explain the main conclusions that have been obtained from the present work.

Problem Statement

In recent years business IT systems and information portals on the Internet have increased their complexity to adapt to the demanding, and at the same time volatile, business and end-user requirements. This complexity has derived in an enormous effort to integrate, adapt and evolve applications and contents in order to satisfy user needs and expectations. The final results are severe penalizations in time to market with a subsequent big business impact.

Service Oriented Architectures (SOAs), which enable a “loose coupling” between applications, business logic and contents are starting to be used to alleviate these problems. Nonetheless, SOAs remain focused on service to service communications and automating service composition (using BPM technologies) on the backend layer.

SOA front-ends are still based on monolithic and non-customizable user interfaces and portals that invoke, when needed and in ad-hoc manner, backend services and processes. In this scenario services do not feature a “face” to human users, as they reside on a merely technical layer.

Existing approaches such as Internet mash-ups are valid in the short term, but unsustainable in the long term. The main issue is the restricted functionality of the UI which is limited to the combination of simple content rather than application functionality. High dependency on the underlying computing infrastructure is also a limiting factor. Changes in the original wrapped applications, in the back-end services, or in the portal infrastructure may cause a failure in the UI and render user service interaction unusable.

The reality is that nowadays users cannot actually benefit from the potential advantages provided by SOA in terms of composition, flexibility, customization and adaptability to the target Context. We advocate a next generation SOA front-end technology where users should be able to create their own applications by combining different pieces without any help from IT experts or deep knowledge about the underlying infrastructure. There are very good reasons to do so:

- IT systems and Internet applications nowadays are still designed and implemented taking as a reference the most common use cases and situations. Catering for more specific needs is, in many cases, commercially not viable. When it is, it almost always leads to increased complexity of the application and the interface, leaving less proficient users out.
- The traditional specify, design, implement, test and deploy lifecycle is no longer sufficient to capture the rapidly changing user needs and requirements. Users are increasingly dependent on IT systems and, at the same time, more knowledgeable about the problems they want to solve and how to tackle them. They hold a very valuable knowledge and some of them are willing to devise new solutions if they are given the means to do so. Traditional tools and methods raise the bar too high for non-IT-aware users in order to be able to contribute new solutions.

These are well recognised Web 2.0 principles that are still not adequately supported by current service front-ends and engineering tools. Providing solutions tailored to users needs and allowing them to contribute their knowledge and creativity represents a huge potential yet to be unleashed. Service front-end is the best place to do so, since it is the layer that imposes the least IT knowledge requirements on the user and is the one closest to the user and the problem to be solved.

Therefore, it is necessary to devise a new SOA front-end capable to provide an environment which empowers end-users to easily use, customize, contribute, enhance and compose services coming with different providers. Furthermore, users should be able to create their own

applications by combining different pieces without any help from IT experts or deep knowledge about the underlying infrastructure.

In addition the new generation SOA front-end should embrace the Web 2.0 principles, exploiting user's collective intelligence and domain knowledge. Service front-ends should be in charge of knowledge capture and further exploitation, as they are the architectural components that deal with user interaction. Thus, new challenges appear with respect to user's knowledge gathering, representation and integration with formal knowledge. At this respect an important research topic is the seamless integration between ontologies (formal semantics) and folksonomies (light semantics).

Once the user knowledge has been properly extracted it should be exploited to enrich context-awareness, to personalize the user-service interaction and to improve existing processes or derive new ones.

From the engineering point of view, the development of context-aware service front-ends is insufficiently supported by existing methodologies and tools. Instead, user interfaces are designed manually for service interfaces and business processes. This fact calls for the need to develop new technologies and tools that simplify and accelerate the development of context-aware service front-ends. Such front-ends will adapt seamlessly to the target environment, providing a harmonized user experience in every device, network or situation.

From the business and enterprise applications perspective, the adoption of new generation service front-ends is also challenging. Traditional monolithic user interfaces will be phased out by advanced environments based on services front-end composition and customization. In addition employees will start to be seen as an essential piece of systems. Tools and technologies for exploiting collective knowledge will start to appear. Last but not least, engineering departments will need to incorporate new paradigms supporting the development of applications that adapt seamlessly to each environment and situation.

Guiding Principles

From our point of view, the challenges explained in the previous section call for the need to user-centric SOAs based on a new generation of service front-end technologies. Such technologies will enable the massive deployment of services on the Internet, driven by the following guiding principles:

- **End-Users Empowerment**, enhancing traditional user-service interaction by facilitating the selection, creation, composition, customization, reuse and sharing of applications in a personalized operating environment.
- **Seamless Context-Aware User–Service Interaction**. New generation service front-ends should have the capability to detect contextual information, to represent it, to manipulate it, and to use it to adapt seamlessly to each situation, supporting human users in a more effective, personalized and consistent way. Novel engineering tools and methods should be devised in order to support context-aware service front-ends.
- **End-Users Knowledge Exploitation** This principle aims to exploit users' domain knowledge and collective intelligence to improve service front-ends. End-Users knowledge can be used to tag resources using light semantics, to assist while interacting with services, to enrich contextual information (for example by means of automatic user profiling) and to infer new candidate processes to be later automated (on the backend).
- **Universal Collaborative Business Ecosystems**. Enterprise systems should incorporate advanced user-centric, context-aware front-ends to enable their employees and other stakeholders to exploit, and share their extensive domain expertise, and their thorough business knowledge. Employees, customers, developers and providers will collaborate

to create and improve enterprise applications, sharing, reusing, changing and combining existing context-aware components (services, contents, things...).

Materializing the Guiding Principles

Enabling users to author and share their operating workspace and applications

The “End-User Empowerment” principle gives an active role to end-users letting them author, enhance, share and customize their own applications and operating environment (workspace), thus going beyond traditional, monolithic user-service interaction. Figure 1 depicts our technical proposal to materialize such principle. The main component of the underlying platform is a resource catalogue containing gadgets (which will implement be the front-end for one or more services) and possibly other application or content delivery services. This resource catalogue plays a similar role than backend service and process registries.

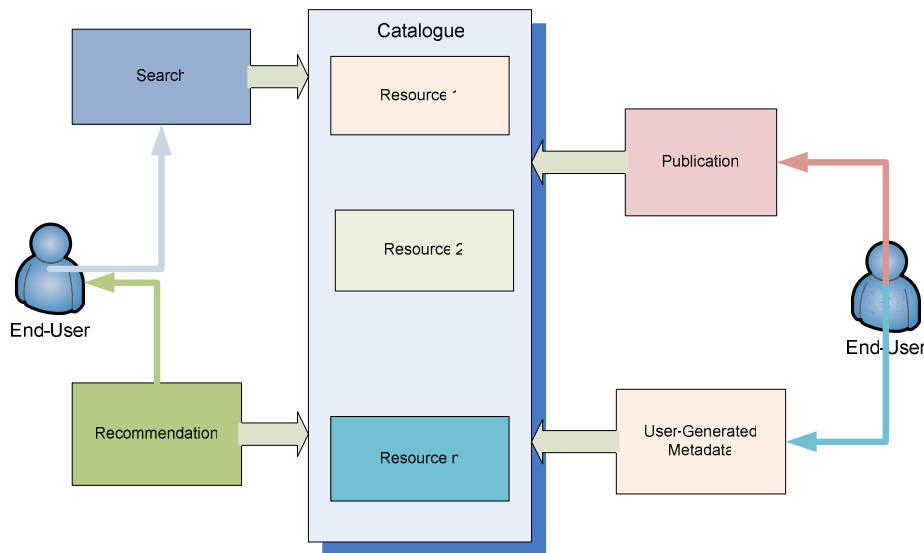


Fig. 1. Resource Catalogue

Our proposed catalogue will be responsible for maintaining the information about each front-end resource (gadget) and associated metadata (template descriptors, deployment information, author, icons, versions, formal-semantics-based annotations, user-assigned tags and punctuation...). In addition the following software modules will be devoted to user-catalogue interaction:

- An upload module that will allow service front-ends creators to add new resources to the catalogue, making them available to the community. To upload a new resource it will be necessary to provide a gadget descriptor (*template*) with all the metadata and deployment information.
- A search and recommendation module that will make it possible to locate and discover those gadgets that satisfy the necessities of end-users at a given moment. This search application will be intelligent, recommending the most suitable resources in the given Context. For example, the set of pieces found in a catalogue should be different when the end-user is at work than when is at home or on vacation.
- A drag-and-drop-based browsing and selection module that will allow end-users to browse and select the resources of their interest, putting them in their operating environment, thus enabling further interactions.

Apart from a resource catalogue and its supporting modules, it is also necessary a runtime environment devoted to the execution of the user's operating environment. Such operating environment (or workspace) will act as a container, enabling the interaction between the end-user and service front-ends. We envisage an operating environment runtime with the following functionalities:

- Flexible layout composition enabling the free disposition of gadgets on the viewport.
- Multiple service front-end views (implemented as tabs, for example) allowing the user to group and compose different gadgets for different problems or situations.
- Publish and subscribe facilities enabling the interconnection between the different gadgets.

A Universal Framework for Ubiquitous and Context-Aware Service Front-Ends

The "Seamless Context-Aware User-Service Interaction" principle will drive the construction of novel service front-ends capable of using contextual information to influence their behaviour, thus supporting human users in a more effective and personalized way. Particularly, Context-Awareness will provide the following benefits to service front-ends:

- User Interface harmonization for every device or mode of interaction. Contextual information will enable automatic content and application delivery tailored to each target environment.
- User-Service interaction enhancement:
 - Contextual information can be exploited to present a different workspace depending on each situation. Each operating environment may include different functionalities or interaction modes depending on the situation (at home, at work, on vacation, on the move, on a business trip ...).
 - Context can be used by resource catalogue search and retrieval modules to recommend the best gadgets to be used under a given situation, matching rich resource descriptions with the characteristics of the target environment.

To materialize our vision, new tools, formalisms and engineering methods need to be devised in order to simplify the development of context-aware service front-ends:

- A device and modality independent Declarative Authoring Language (and associated standard context-based adaptation policies). This is a fundamental piece for those service front-ends intended to work on multiples devices or modes of interaction.
- A Context Framework composed, at least, by the following elements:
 - A Universal and Extensible Context Model enabling the development of Context Vocabularies.
 - A Context Mediation Layer, including binding formalisms, hiding applications from the complexities of dealing with multiple and heterogeneous Context data sources.
 - A Universal Context API, providing a simple, uniform programming abstraction for the development of context-aware services.

Once this Context Framework is in place an application can infer which actions should be triggered, what data is relevant, and what topics are interesting for the user within an specific context. In the following sections we describe with a finer level of detail the elements that compose our envisaged Context Framework (see figure below).

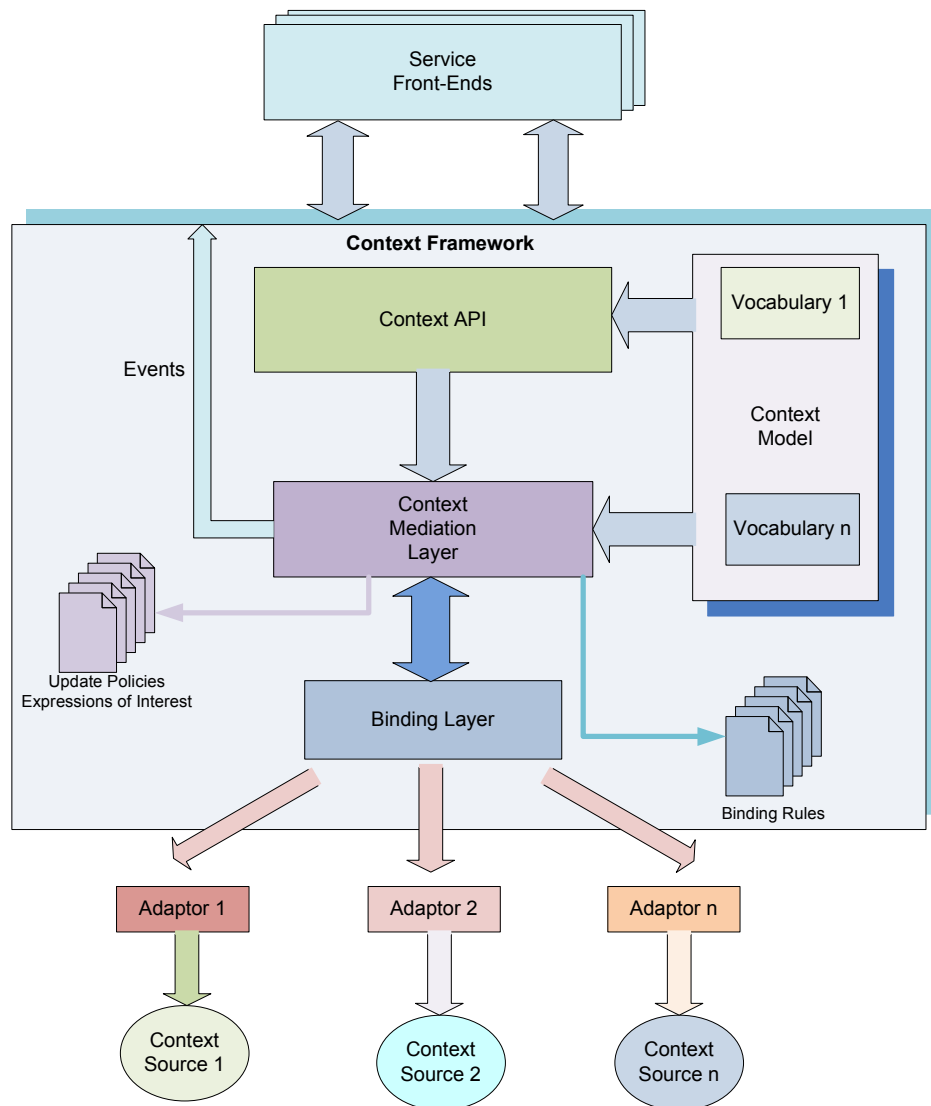


Fig. 2. Proposed Context Framework

Universal Context Model

In our vision the Context is a set of Properties that describe the characteristics and the situation of different Components that are relevant for a service front-end in a given environment. Every *Component* is an instance of an *Aspect*. An Aspect is a Context Component type. Examples of Aspects are “Web Browser”, “Device”, “Camera”, “Display”, “Battery”, “Proxy”, “User Profile”, “Preferences”, and the like. There are some Aspects with only one possible Component in every situation, such as “Preferences”, while others can have more than one Component, for instance “Battery” or “Camera”.

The Context is actually the aggregation of the different Components, (including their characteristics and state) which are relevant in a situation. As a consequence the actual value of a Property in a given Context can be influenced by the individual Component Property Values. This is a case where the whole (the Context) is more than the sum of its parts (the Context Components). For example, the value of property “*supportedImageFormats*” for the whole Context can depend on the value of this property for the “Web Browser”, “Device” and “Content Transformation Proxy” Aspects.

At the same time every Context Component can encompass other child Components. Such child Components can influence on the parent Component Property values. For example, a “Device” Component will have as its children one or more displays, one or more batteries and

possibly one or more cameras, among others. The corollary is that the Context itself as a whole can be also modelled as an especial Component representing the aggregation of all the child Context Components present and relevant in a situation.

Last but not least, the Context is dynamic in nature, thus Context Components can appear and disappear at different moments in time. Besides, while certain properties will be barely modified over time, like user name or age, others may change frequently, like time or position. When Context Properties are queried they should all be up to date, or at least marked as expired. As a result the Context Model should also define a set of standard events related to the dynamic changes that can occur in the Context.

We advocate the adoption of a baseline, *Universal Context Model* capable of representing formally the Context as it has been conceived and described on the previous paragraphs. Such Context Model will model a universal set of Aspects, Properties and related Events useful for any problem, enabling at the same time to introduce new ones that can be of interest in a specific domain. For instance, in the Location Based Services domain an specific property called “*zipCode*” or an event named ‘*enteringPlace*’ might be defined

The adopted Context Model should have the following features:

- **Modular.** The Context Model should enable the introduction of different Vocabularies of Properties and Events (with different granularity levels) devoted to different Aspects (and Components) of the Context. Every Context Vocabulary should be able to express the meaning of its Properties and Events in terms of the Universal Context Model. As a result, different context-aware frameworks, with potentially different vocabularies, will be interoperable between them.
- **Extensible.** The Context Model should be ready to be extended allowing the introduction of new Properties and Aspects, both standalone or derived from existing ones. For example, a Property called “*niceDay*” could be automatically inferred using a given temperature and an atmospheric condition.

Finally, we propose to formally represent the following Context Aspects as part of the Universal Context Model:

- **The Delivery Context:** the set of attributes that characterizes the environment in which a service front-end is going to be delivered. It includes, among others:
 - the characteristics of the device and user agent (the access mechanism)
 - the network capabilities (maximum and current bandwidth, cost of connection and the like).
 - Local user's settings (also known as “Preferences”) local to a service or to the access mechanism such as the preferred font size or colours, the screen brightness, the audio speaker volume settings ... etc.
- **User Context:** This aspect includes properties such as User identity, Profile and Roles, social network and overall interests.
- **Surrounding Environment,** including the spatial location, speed, course, light conditions, temperature, level of noise, nearby objects / things ...
- **Situation / Moment** which has to with variables such as date and time, weather, season, at home or at work, on vacation, on a business trip ...

The starting point for a standard, minimal and universally-accepted Context Model could be the W3C's Delivery Context Ontology [11] (work in progress). Such ontology could be generalized and extended with additional modules capable of representing new general-purpose entities.

Context Mediation Layer

The Context Mediation Layer will be in charge of providing a uniform Context view and abstraction to service front-ends, in accordance with the Context Model described on the previous section. Specifically, the functionalities provided by this layer should be:

- An adaptation layer (implemented via a set connectors) between the Context itself and the sources of contextual information (device description repositories, sensors, nearby objects, network elements, static user profiles, external devices, well-known services published on the Net, etc.). Different connectors should be provided, thus hiding applications from all the proprietary communication protocols, specific APIs or other artefacts needed for gathering Context Properties.
- A declarative mechanism (for example an XML format) to bind Context Properties to specific sources of contextual information. This binding could also include transformations to be done over the data. For example, the “*location*” property may be bound to a GPS device or to a mobile operator network service and then transformed from WGS-84 coordinates to UTM coordinates.
- A declarative mechanism (for instance an XML format) that allow applications to express their interest in Context Properties beforehand. The expression of interest should also include requirements about update policies for each Property. For example, the “*location*” property may have different update requirements than the “*batteryLevel*” or “*freeMemory*” property. The mediation layer should make available (through the Context API) the values for those Properties of interest to a service front-end, respecting the at the same time the update policies.
- Context Event notification to service front-ends in accordance with the registered event handlers (see below)

Universal Context API

A critical point for addressing interoperability in service front-end context-awareness is the provision of a harmonized API that makes it possible to gain access to Context Properties in any programming language, regardless of where the contextual information actually is. This Universal Context API should have the following characteristics:

- Aligned with the Context Model described above, thus making it possible to retrieve Context, Aspect and Components Property Values
- It should provide an expression language syntax to query the context in a more declarative manner
- Vocabulary-Independent, allowing to work with different Vocabularies of Properties and Aspects
- Aware of Property and Property Value metadata, such as units, data types, expiration times, etc.
- It should include both query-response and event-based mechanisms for dealing with contextual information changes. For example, the Context API should allow to register handler code to be invoked when a property value reaches a threshold, or takes a value.

The W3C’s DDR Simple API (Proposed Recommendation) [14] and DCCI (Candidate Recommendation) [17] should be considered as starting points for this Universal Context API.

A framework for user's knowledge capture and exploitation in service front-ends

The “End-Users Knowledge Exploitation” principle aims to exploit users' domain knowledge and collective intelligence for enhancing service front-ends. This principle can be materialized by:

- Encouraging users to create and compose their own service front-ends as a way to solve problems that were not anticipated and that require thorough domain knowledge.
- Stimulating users to share their knowledge by means of semantic descriptions and annotations for service front-ends. This point is really challenging since it will imply the integration between folksonomies (light semantics) and ontologies (formal semantics).
- Monitor users behaviour and input to service front-ends in order to:
 - Enrich contextual information about the user by employing *automatic user profiling*, to overcome the problem of incomplete or inaccurate user-provided profiles. Users with similar interests can also be grouped into classes, and assumptions about individual's interests can be made based on the interests of the whole group. This leads to the concept of social profiling.
 - Enhance interaction, for example current search engines use previous inputs to assist the user while searching. The same idea could be applied, for instance, to application forms.
 - Detect interaction or composition patterns in order to infer new processes suitable to be later automated (on the backend).

The accomplishment of the previous ideas requires research on new formalisms and technologies such as:

- Advanced user-centric integrated development environments (User-Centric IDEs) that make it possible the creation of new and not anticipated service front-ends with a couple of clicks and without requiring deep IT knowledge. Such user-centric IDEs should provide easy-to-use tools with graphical abstractions that hide the technology complexities. For instance, with this approach the publish-and-subscribe-based connection between gadgets can be graphically represented using boxes and connectors .
- A catalogue browsing module enhanced with collaborative, user-generated semantics (folksonomies). These folksonomies will be constructed incrementally as users assign tags to gadgets found in the catalogue. A rating mechanism can also be used to promote the best quality service front-end resources. This user-generated metadata should be integrated with the formal metadata (coming from an ontology, for example) about service front-ends and will serve to improve the search and recommendation processes.
- Collaborative filtering techniques aimed at analyzing user behaviour in order to:
 - Create and extrapolate implicit user profiles taking into account which links are more frequently clicked or the time spent on each one.
 - Capture the underlying user's knowledge to be later used, for example to assist form filling
 - To extract repetitive interaction patterns that could drive the automation of new, not anticipated processes

All the novel approaches described above should be the starting point for a new innovation culture based on continuous improvement via reusing and sharing of knowledge. In fact, users will connect and use resources in their workspace according to their skills, situation or others. Wheel-reinventing in organizations or by individuals regarding service front-ends will no longer happen. Competition will rise as there will be an ecosystem of providers and consumers of

resources willing to provide the best solution for a problem. Users and not marketing departments will evaluate, vote and choose the best resources for a problem.

A new generation of collaborative, user-centric and context-aware IT systems

The “Universal Business Ecosystems” principle aims at the incorporation of advanced user-centric, context-aware business front-ends to enable employees and other stakeholders to exploit, and share their extensive business expertise. As a consequence employees, customers, developers and providers will collaborate to create and improve enterprise applications, sharing, reusing, changing and combining existing context-aware components (services, contents, things...).

To materialize this principle it is necessary to explore alternatives that could be used in the development of enterprise applications and adopt a fully user-centric approach. This call for the need to:

- Extend the concept of end-user to people who develop, use, market and exploit the system.
- Consider all user-roles as a fundamental part of systems, in order to develop innovative, comprehensive and coherent solutions to B2C and B2E solutions.
- Enterprise Collaboration Systems (ECS) evolution towards a new paradigm in which the more skilled employees should be co-producers not only of information, but also of software, services and applications.
- Empower collaboration without requiring any programming skills, only thorough business knowledge. The main aim will be both to foster the company’s competitive advantage without involving IT departments, and sharing of user-created solutions with the remainder of the organization.
- Promote a new innovation culture introducing new operating procedures by mixing and integrating available services.
- Break the frontiers between systems, companies and users in favour of a universal platform, the Internet of Services.
- Embrace the Software as a Service (SaaS) model as an effective software-delivery mechanism. This will change the department’s focus from deploying and supporting applications to managing the services implemented by these applications.

Proposed Architecture

Overview

On this section we propose a novel architecture for the next generation service front-ends which has been devised in accordance with the presented guided principles. For the sake of clarity we have separated the authoring and run time phases of the service front-end lifecycle (see figure 2).

Gadgets will be the main building blocks of such architecture. A gadget implements the user interface and application logic necessary to interact with one or more underlying services. Gadgets are self-contained front-end components focused on a single goal and, consequently, of limited complexity. Gadgets can be grouped into workspaces.

Our proposed architecture for the authoring phase includes two main components:

- A **“Gadget Authoring Tool”** which it is a user-centric IDE devoted to gadget design and creation. This is a visual tool that assists non-IT-aware users in creating their own service front-end resources. Using this user-oriented IDE gadget authors will be able to visually design, reuse and share gadget screens, flows and back-end resource compositions or connectors among others.
- A **“Workspace Editing Tool”** intended to the design of custom user’s workspaces. This tool will permit the visual design, reuse and sharing of user’s workspaces by selecting, connecting and composing the most suitable gadgets for dealing with a domain problem.

These two tools will be supported, at least, by the following formalisms:

- A Declarative Authoring Language for the description of device and modality independent user interfaces.
- An standard format and infoset for the description of gadget metadata (Gadget Template).

At run time we propose a **“Workspace Access Layer”** that will be in charge of:

- Rendering each user’s workspace (and the contained gadgets) as per the characteristics of the target Context.
- Implementing all the artefacts needed to support the execution of gadgets at runtime, such as publish and subscribe communication mechanisms or persistence of gadget data and state.

Additionally there will be a set of horizontal modules dedicated to different aspects that are common to the runtime and authoring phases:

- A **Resource Catalogue** containing all the metadata about the different building blocks of the architecture (gadgets, screens, flows, workspaces, content delivery resources, application data resources, resource compositions, etc.).
- A **Context Framework** implementing all the concepts, formalisms and artefacts described previously on the present work.
- **Identity and Session Management** for dealing with user’s session and identity among others.
- A **Knowledge Framework** following the approach described on the corresponding section of this paper.

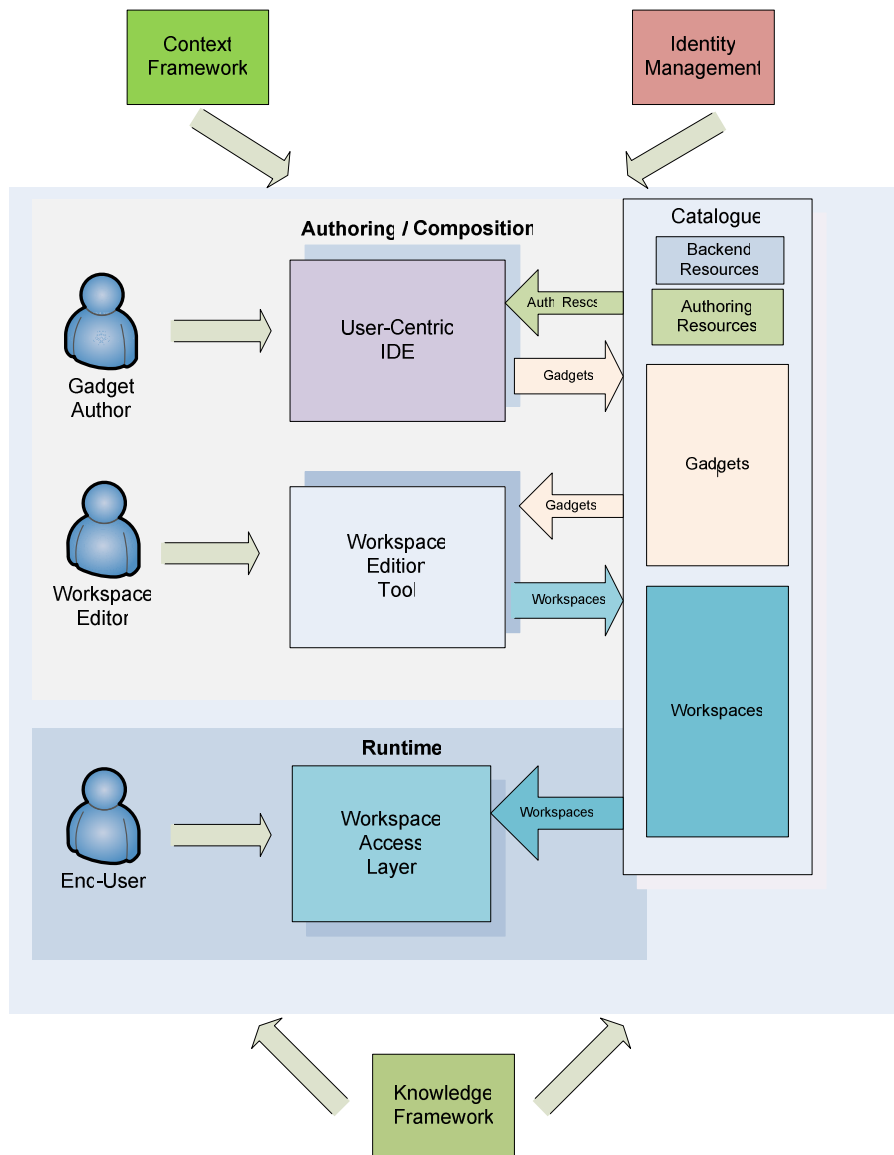


Fig. 3. Proposed Architecture for Next Generation Service Front-Ends (Overview)

Gadget Authoring Tool

The Gadget Authoring Tool is a user-centric IDE devoted to the rapid design and automatic compilation of gadgets. This IDE is mainly oriented to non-IT-aware users who wish to create their own gadgets to deal with a domain problem. This IDE will manage a set of concepts and visual abstractions that will hide from the complexities of dealing with the underlying markup, script languages or application programming interfaces.

Figure 4 summarizes how this tool would work. Gadget Authors (end-users creating new gadgets) will visually compose a gadget from a series of building blocks (authoring resources) available in a palette. This palette is actually a specific view of the resource catalogue and can contain UI artefacts (screens), operators, screen flows, ready-to-use back-end resources and compositions, etc. The outcome produced by the gadget authoring tool will be:

- A Template Descriptor with all the needed machine-readable metadata to use and deploy the gadget

- A set of declarative markup, script, style sheets or other artefacts ready to be used at runtime. These artefacts will constitute a runtime machine-interpretable description of the gadget interface and behaviour

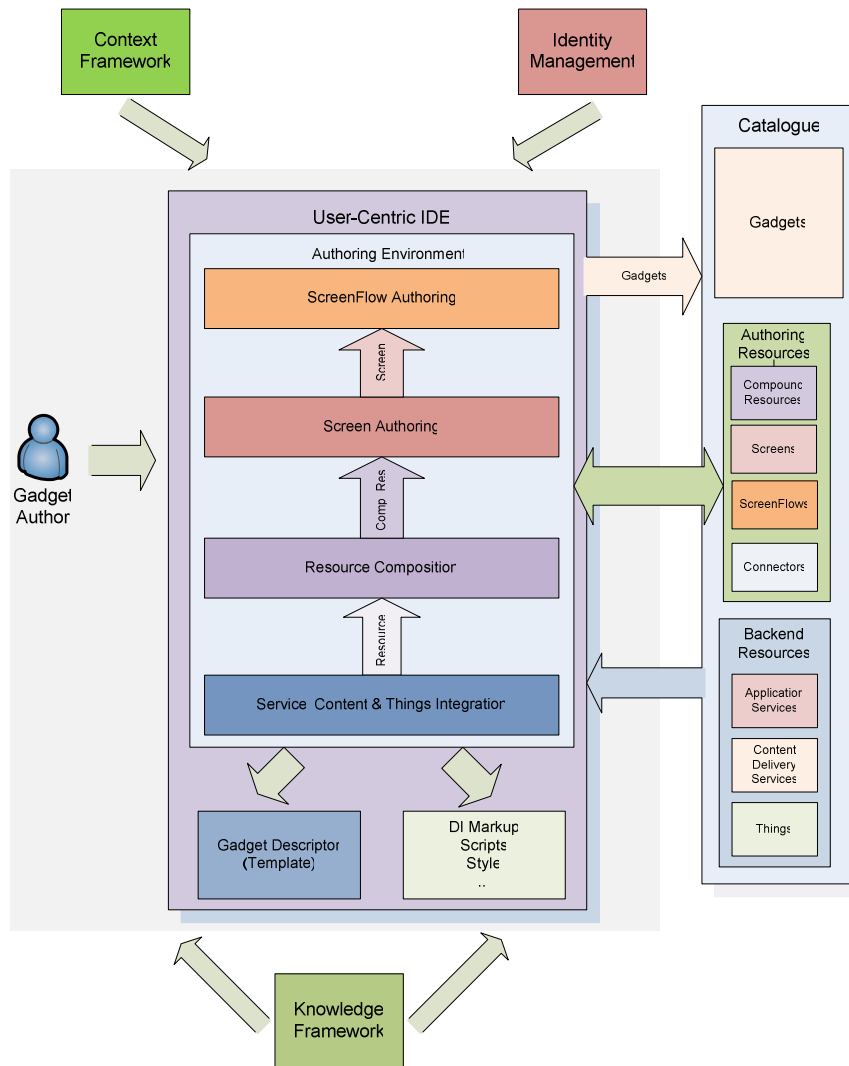


Fig. 4. User-Centric IDE Proposal

Declarative Authoring Language

Traditional user interface development approaches are insufficient for supporting the new generation service front-ends. On the one hand they are oriented to specific devices or modes of interaction (normally a PC device). On the other hand they promote an imperative, platform driven, development style which increases the effort, as UI designers cannot fully concentrate on the real application requirements. For example, Web & AJAX-based user interfaces are currently developed using HTML (which it is neither device nor modality independent) and scripting (which is both device dependent and imperative), see [4].

Going one step further, traditional UI platforms and toolkits lack the formalisms necessary to deal with Context-Aware service front-ends. For example, there are no declarative mechanisms to specify how an interface should adapt according to different Delivery Contexts. Instead the developer needs to do the adaptation manually, using an ad-hoc and costly approach which does not promote reuse or standardization.

Compared to traditional application development models declarative approaches offer the following benefits:

- Reduced development and maintenance costs throughout the application lifecycle, making it easier for people with different roles to work on different aspects of the development process, e.g. task models, domain models, user interaction, and graphical look and feel
- Simplified development process that allows people outside of the IT department to develop applications without the need for extensive programming skills
- Improved security, accessibility and usability
- Easier delivery to a wide range of devices and platforms

We propose a layered approach to the development of UI for services front-end:

- Abstract UI (device independent)
- Concrete UI (device dependent)
- Physical realisation for specific devices

All of the layers (with the possible exception of the physical realisation) can be represented in XML. Each layer embodies a model of behaviour at a progressively finer level of detail. The model view controller pattern should be used to cleanly separate the user interface, the dialog behaviour and the data it operates on.

Each layer can be considered as the result of a transformation, driven by different adaptation policies, of the layer immediately above it. Transformations will have access to the Context, which models user preferences, device and web browser capabilities and environmental conditions.

Adaptation policies are declarative statements that permit UI developers to express their intentions about the realization of the user interface under different Contexts. Examples of Adaptation Policies are “ordered by distance to the user’s location”, “the best in this Context”, “use this image if the target device is a PDA”, etc. The beauty of “Adaptation Policies” is that authors only express their intentions and they do not care about how their desires will be finally accomplished.

The physical realisation can be implemented via an optional mapping to a low level markup language or through compilation.

Gadget Descriptor Template

A “Gadget Descriptor Template” is a machine-readable gadget description. It should contain, among others, the following metadata items:

- Author names and affiliations
- Date
- Icons
- Human-Readable description
- Pointers to the gadget source code
- Publish-Subscribe metadata:
 - Data Items published by the gadget (including their type, name, semantics, etc.)
 - Data Items consumed by the gadget (including their type, name, semantics, etc.)
- Context Metadata

- Expression of interest on Context Properties
- Update policies for Context Properties
- User preferences (local settings) accepted by the gadget

Workspace Editing tool

The “Workspace Editing Tool” is a “Mashup’s Editor” aimed to create and design new workspaces composed by different gadgets connected between them. The final target is to create new, modular and anticipated service front-ends (instant applications) by the combination of smaller pieces (gadgets). Each user can have any number of workspaces and can share them with other members of the community.

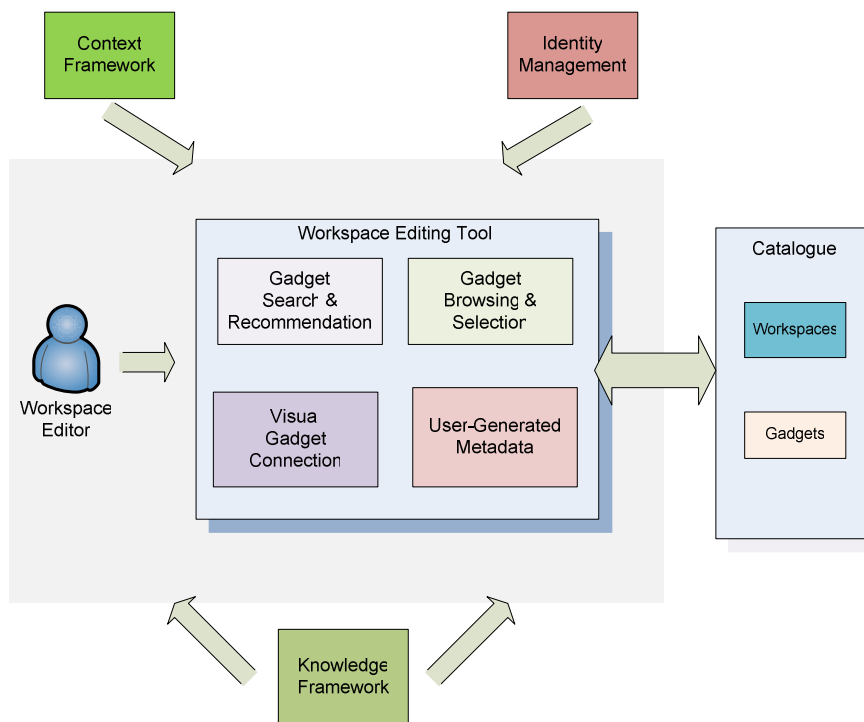


Fig. 5. Workspace Editing tool

An end-user who is designing a new workspace adopts the role of “Workspace Editor”. We propose a “Workspace Editing tool” (see figure above) composed of the following modules:

- A “**Search and Recommendation Module**” that will allow users to find and discover new gadgets ready to be used in their workspaces. To locate the most appropriate gadgets this module will need to use the gadget metadata stored in the catalogue.
- A “**Selection and Browsing Module**” dedicated to catalogue browsing and subsequent gadget selection and inclusion into the workspace.
- A “**Visual Gadget Connection Tool**” that will allow users to connect graphically the different gadgets present in a workspace. Such connections are aimed to share data between gadgets. At runtime the user-made connections will be technically implemented by means of publish-subscribe mechanisms.
- A “**User-Generated Metadata Module**” that permits users to enrich the knowledge about the gadgets present in the catalogue. This metadata can be tags coming from a folksonomy, a voting mechanism for rating those gadgets of better quality and the like. This “User Knowledge” will serve to enrich other kind of knowledge about gadgets automatically inferred by the “Knowledge Framework”.

Workspace Access Layer

This layer is responsible for giving access to one or more workspaces to end-users. On the one hand it should adapt a workspace and the contained gadgets to the restrictions dictated by the target Context. On the other hand it should provide a runtime environment for gadget execution. This layer can be divided into the following modules:

- **Context-Aware Adaptation Layer** devoted to adapt the user-workspace-gadget interaction to the contextual characteristics imposed by the access mechanism or other contextual conditions. In fact this layer will be in charge of rendering the user interface in accordance with descriptions and adaptation policies expressed by means of the declarative authoring language. There are two main modules that use the functionalities provided by this layer:
 - **Gadget Delivery Layer** in charge of rendering each gadget according to the rules defined in the workspace, the gadget user interface defined at authoring time and the target Context.
 - **Workspace Delivery Layer** in charge of rendering a workspace according to the presentation rules indicated by the end-user at editing time (order, position, layout and the like) and the Context restrictions. In addition it should support user-workspace interactions.
- **Gadget's Runtime Platform** It is a mashup platform responsible for providing technical capabilities to gadgets: publish-subscribe facilities, persistence mechanisms or screen flow execution among others.

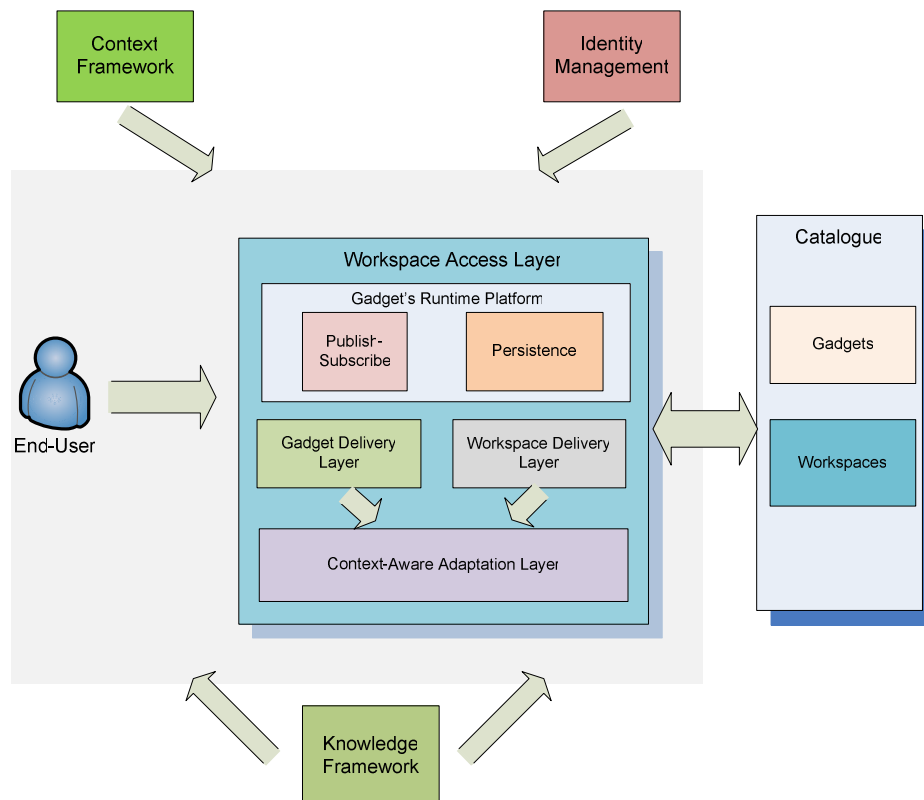


Fig. 6. Workspace Access Layer

Proposed Conceptual Model

The figure below gives an overview of the proposed conceptual model.

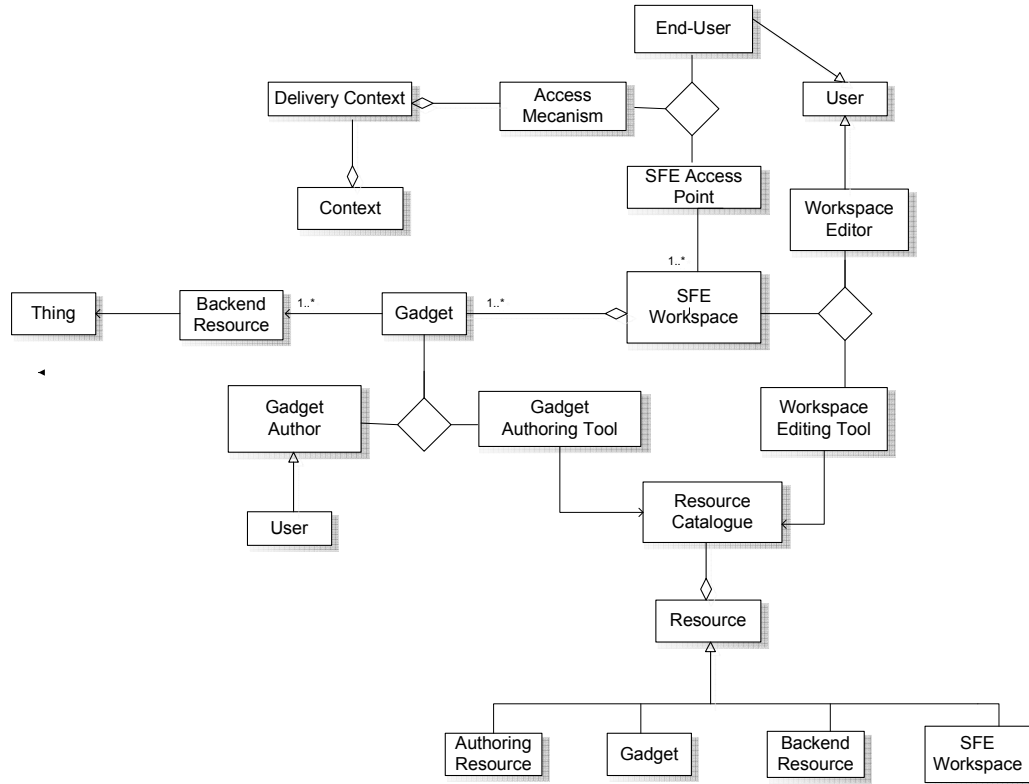


Fig. 7. Conceptual Model

Glossary of terms for Service Front-Ends

General Concepts

- **Resource** Anything that can be identified by a Universal Resource Identifier (URI).
- **Authoring Resource** A resource available to authors to build their own gadgets.
- **Back-End Resource** A resource that acts as an intermediary (and possibly adaptor) between a back-end service, content delivery service or thing and a service front-end.
- **Resource Composition** A resource that is the result of the composition of two or more back-end resources.
- **Thing** Any physical entity suitable to be wrapped using a back-end resource
- **Resource Catalogue** A repository dedicated to store meta-information about resources.

User Concepts

- **User** A human who perceives and interacts with service front-ends adopting at different times the roles of gadget author, workspace editor or end-user.
- **Gadget Author** A user who conceives, designs and shares gadgets (typically using a user-centric IDE).

- **Workspace Editor** A user who creates, reuses and shares workspaces using a Workspace Editing tool.
- **End-User** A user who accesses to their workspaces to interact with gadgets.

Context Concepts

- **Context** Describes in, an aggregate manner, the situation of several entities that are considered relevant to the user-service interaction.
- **Delivery Context** A set of attributes that characterizes the environment in which devices interact with services. It includes the capabilities of the access mechanism and the local preferences of the user when using a service, such as the volume and brightness, the font size, etc.
- **Access Mechanism** A combination of hardware (including one or more devices and network connections) and installed software that allows a user to interact with services.
- **Context-Awareness** the ability of applications and services to adapt their behaviour, interface or contents to the target Context, providing a more consistent and effective interaction.

Service Front-End Specific Concepts

- **Gadget Authoring Tool** A user-centric IDE that allows non-IT-aware users to author their own Service Front-End-Resources and to upload them to a resource catalogue.
- **Workspace Editing tool** A visual editor that allow users to search, discover and connect (mashup) gadgets to solve a domain problem.
- **Service Front-End Access Point** A resource that provides access to one or more *Service Front-End Workspaces*.
- **Service Front-End Workspace** A group of two or more Service Front-End Resources that have been composed (using the Workspace Editing tool) and combined to solve a domain problem. The number of Service Front-End Workspaces accessible at a given time depends on the Context.
- **Service Front-End Resource (Gadget)** A resource that implements the user interface and application logic necessary to interact with one or more underlying services. Gadgets are self-contained front-end components focused on a single goal and, consequently, of limited complexity. They publish events to, and consume events from, other Service Front-End Resources.

Relevant Projects

MyMobileWeb

MyMobileWeb [13] is the open source reference implementation of the next generation content and application adaptation platform for the Mobile Web. MyMobileWeb provides the following technologies:

- IDEAL language for the (semantic) description of the device independent user interface
- Context ontologies and APIs
- Content and application adaptation runtime (semantics-based)
- Mobile AJAX Framework

- Content and service discovery
- Semantic Bar (enriches browsing exp.)
- Authoring tools (Eclipse Plugin)

MyMobileWeb enables the creation (in time to market) of high quality mobile applications and contents capable of adapting to multiple Delivery Contexts, satisfying user and content providers' expectations. MyMobileWeb is intended to the development of high-quality dotMobi portals and Rich Mobile Web applications intended to work in multiple delivery contexts

EzWeb

The EzWeb [7] project pursues the development of an enriched enterprise mash-up platform and the development of key technologies to be employed in building the **front end layer** of new **generation SOA architecture**. EzWeb aims at supporting the following criteria:

- **End-users must feel fully empowered**, They must be able to self-serve from a wide range of available resources, providing access to content and application services, in order to set up their own personalized operating environment in a highly flexible and dynamic way ("Do it yourself", IKEA philosophy).
- **Active participation of users has to be enabled**, allowing them to create resources as well as share and exchange both knowledge and resources with others and learn together, thus accelerating the way innovations and improvements in productivity are incorporated.
- **Interaction must be adapted and relevant to context**, giving the term "context" the widest possible meaning, in a way that comprises both user context (knowledge, profile, preferences, language, information about social networks the user belongs to, etc.) and delivery context (static and dynamic characteristics of the device used for access, geographical and time location, connection bandwidth, etc.). Dynamic context variability and user mobility must also be taken into consideration.

FAST

FAST [8] aims at providing an innovative visual programming environment that will facilitate the development of next-generation composite user interfaces. It constitutes a novel approach to application composition and business process definition from a top-down user-centric perspective. The main objective of the project is to create a new a visual programming environment that will facilitate the development of complex front-end gadgets, involving execution of relatively complex business processes that rely on back-end semantic Web services.

The approach adopted will be user-centric rather than program-centric: instead of first building programs that orchestrate available semantic Web services and then try to figure out how to implement interaction with the users at given points of the process execution flow, programmers will start from the front-end gadgets that the user will see and interact with, and then, afterwards, they will visually establish the connection to back-end Web services going through process execution flows if necessary. The way programmers will visually establish this connection will follow an approach similar to what sequence diagrams in UML look like.

Acknowledgments

This work has been partially supported by the EU through the NEXOF-RA IP Project, contract number FP7-216446.

The editors wish to acknowledge significant written contributions from:

- Francisco Garijo (Telefónica I+D)
- Javier Soriano, David Lizcano and Miguel Jiménez (Universidad Politécnica de Madrid)

Bibliography

- [1] Alonso, G., Casati, F., Kuno, H. & Machiraju, V.(2004). Web Services Concepts, Architectures and Applications. Springer, 2004
- [2] Cantera J.M., Hierro J., Jiménez M., and Soriano J. (in alph. order), Delivering Mobile Enterprise Services on Morfeo's Mobility Channel Open Source Platform. In proceedings of the 1st International Workshop on Tools and Applications for Mobile Contents (TAMC'06), at the 7th IEEE International Conference on Mobile Data Management (MDM'06). Published as IEEE Computer Society Digital Library.
- [3] Cantera J.M., Jiménez M., López G., and Soriano J. (in alph. order), Morfeo's Semantic Mobility Channel: Ubiquitous and Mobile Computing Meets the Semantic Web. Transactions on Engineering, Computing and Technology, Vol. 11, pp. 24-30, February 2006, World Enformatika Society. ISSN 1305-5313.
- [4] Cantera J.M., Marin I. et al, Declarative Models for Ubiquitous Web Applications .- Morfeo-MyMobileWeb Position Paper, W3C's International Workshop on Declarative Models for Ubiquitous Web Applications, Dublin, June 2007, <http://www.w3.org/2007/02/dmdwa-ws/Papers/jose-m-c-fonseca.html>
- [5] Dey.A.K. Providing architectural support for building context-aware applications. PhD Thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2000. Director Gregory D. Abowd.
- [6] Dourish P., Seeking a foundation for context-aware computing. Human-Computer Interaction, 16(2/4):229-241, 2001
- [7] EzWeb Project, <http://ezweb.morfeo-project.org>
- [8] FAST Project, <http://www.fast-project.eu>
- [9] Fernández R., Lizcano D., López J., Reyes M., and Soriano J.(in alph. order), Tackling Interoperability in Composite Applications from an Enterprise Mash-up Perspective. Proceedings of the 14th International Conference on Concurrent Enterprising, ICE 2008, Lisboa, Portugal, June 23-25, 2008. To appear.
- [10] Gartner Inc. (2006). Hype Cycle for Software as a Service, Gartner Research, 10 August 2006.
- [11] Lewis R., Cantera J.M., Delivery Context Ontology, W3C Working Draft, April 2008, <http://www.w3.org/TR/dcontology/>
- [12] McAfee, A.(2006). Enterprise 2.0: The Dawn of Emergent Collaboration. MIT Sloan Management Review, Vol.47, No.3 (pp. 21-28).Spring 2006.
- [13] MyMobileWeb Project, <http://mymobileweb.morfeo-project.org>
- [14] Rabin J., Cantera J.M. et al., Device Description Repository Simple API, W3C Working Draft, to be published as W3C's Proposed Recommendation on August 2008, <http://www.w3.org/TR/DDR-Simple-API/>
- [15] Soriano J., Lizcano D., Hierro J., Reyes M., Schroth C, Janner T., Enhancing User-Service Interaction through a Global User-Centric Approach to SOA, *icns* pp. 194-203,2008.
- [16] Soriano J., Lizcano D., Cañas M., Reyes M. and Hierro J., Fostering Innovation in a Mashup-oriented Enterprise 2.0 Collaboration Environment. System and Information Science Notes, Vol 1 Num 1, pp.62 - 69, July 2007. ISSN 1753-2310. SIWN International Conference on Adaptive Business Systems (ICABS'2007) Chengdu, China, 22-24 July 2007.
- [17] Waters K., Hosn R. et al., Delivery Context Client Interfaces (DCCI), W3C Candidate Recommendation, December 2007, <http://www.w3.org/TR/DPF/>