# The FAST Platform: An Open and Semantically-Enriched Platform for Designing Multi-channel and Enterprise-Class Gadgets

Volker Hoyer[1,6], Till Janner[1,6], Ivan Delchev[2], Andrea Fuchsloch[1], Javier López[4], Sebastian Ortega[4], Rafael Fernández[4], Knud Hinnerk Möller[5], Ismael Rivera[5], Marcos Reyes[3], and Manuel Fradinho[7]

[1] SAP Research St. Gallen, 9000 St. Gallen, Switzerland
[2] SAP Research Zurich, 8000 Zurich, Switzerland
[3] Telefonica I+D, 28043 Madrid, Spain
[4] Universidad Politecnica de Madrid, 28660 Madrid, Spain
[5] DERI, National University of Ireland, Galway
[6] University of St. Gallen, **=mcm***institute*, 9000 St. Gallen, Switzerland
[7] Cyntelix Corporation, Galway, Ireland
{volker.hoyer,till.janner}@sap.com,
{ivan.delchev,andrea.fuchsloch}@sap.com,
{jlopez,sortega,rfernandez}@fi.upm.es,
{knud.moeller,ismael.rivera}@deri.org,
mru@tid.es, mfradinho@cyntelix.com

**Abstract.** The transfer of the mashup paradigm in corporate environments needs additional capabilities beyond those typically associated with consumer mashups. In this paper, we present the architecture of the FAST platform which allows creating enterprise-class and multi-channel visual building blocks (so called gadgets) in an ad-hoc manner. The design of complex enterprise-class gadgets is supported by an integrated semantic concept which hides the complexity from the actual users. The architectural components of the platform, a technical life cycle model for enterprise mashups, and the FAST gadget ontology are presented. By means of a cross-organizational real-world scenario from the marketing/ promotion event area, we demonstrate the value and potential of the FAST platform.

**Keywords:** Enterprise Mashups, Gadgets, Situational Applications, Semantics, Multi-Channel Visual Building Blocks, FAST Project.

## 1 Introduction and Motivation

After introducing transaction systems such as enterprise resource planning (ERP), customer relationship management (CRM), or supply chain management (SCM) since the beginning of 1990, a next wave in corporate technology adoption, the Web 2.0/ peer production philosophy, addresses ad-hoc and situational applications [1]. It integrates actual end users in order to generate new information or edit the work of

L. Baresi, C.-H. Chi, and J. Suzuki (Eds.): ICSOC-ServiceWave 2009, LNCS 5900, pp. 316–330, 2009.

others. Renowned management scholars such as Andrew McAfee and Don Tapscott envision an Enterprise 2.0 [2, 3]. It leverages new consumer-driven technologies in order to put people in the center of the information-centric work.

In this context, a new software development paradigm, known as enterprise mashups, has gained momentum. At the core of the mashup paradigm are two aspects: First, empowerment of the end user to cover ad-hoc and long tail needs by reusing and combining existing software artefacts. Second, broad involvement of users based on the peer production concept. In contrast to traditional software development concepts aligned with Service-Oriented Architectures (SOAs), enterprise mashups usually aren't constructed by a team of traditional software developers. Instead, they are created by users from the business units characterized by no or limited programming skills. They desire specific functionality that mainstream SOA-based enterprise applications don't provide [4]. In this kind of grassroots computing [5, 6], the focus on delivering a set of user friendly building blocks rather than finished applications enables users to automate also tactical and opportunistic applications.

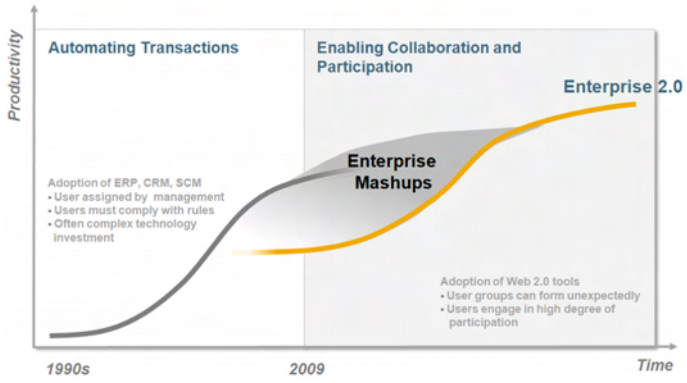


**Fig. 1.** From Automating Transactions to an Enterprise 2.0, adapted from [1]

Market research companies like Gartner [7], Forrester [8], or Economic Intelligence Units [9], and leading management consulting firms like McKinsey [10], forecast a growing practical relevance for the mashup paradigm over the next few years. Gartner sees mashup applications at the mainstream adoption in less than two years in its hype cycle for Web and user interaction technologies 2008 [7]. In addition, several mashup tools came up in the recent years [11], i.e., IBM Mashup Center, Intel Mash Maker, SAP Research RoofTop, Microsoft Popfly, Yahoo Pipes, etc. However, the transfer of the consumer-driven mashup paradigm to corporate environments needs additional capabilities beyond those typically associated with consumer mashup offerings.

The goal of this paper is to fill this gap by designing an open and semantically-enriched platform which allows creating enterprise-class and multi-channel visual building blocks (so called gadgets). In the course of the EU funded project FAST[1], we are currently implementing the platform. By means of a first cross-organizational

[1] http://fast.morfeo-project.eu, last checked 2009-08-13

real-world scenario from the marketing/ promotion event area, the platform and indirectly the underlying concepts are evaluated.

The remainder of the article is structured as follows: After introducing the terminology of enterprise mashups and elaborating on the requirements for corporate purposes in section two, we present the FAST platform in section three. In particular a life cycle for enterprise mashups, the architectural components, and the designed FAST gadget onotology are presented. Section four includes a demonstration by means of a first B2B mashup scenario. Finally, section five concludes with a brief summary and provides an outlook on future work.

## 2 Related Work and Background

### 2.1 Enterprise Mashups – Definition and Terminology

In the literature, the exact definition of enterprise mashups is open to debate. In this work, we refer to the definition of [12]: *"An enterprise mashup is a Web-based resource that combines existing resources, be it content, data or application functionality, from more than one resource by empowering the end users to create individual information centric and situational applications"*. By simplifying concepts of SOA and by enhancing them with the Web 2.0 philosophy of peer production, enterprise mashups generally focus on software integration on the user interface level instead of traditional application or data integration approaches [13].

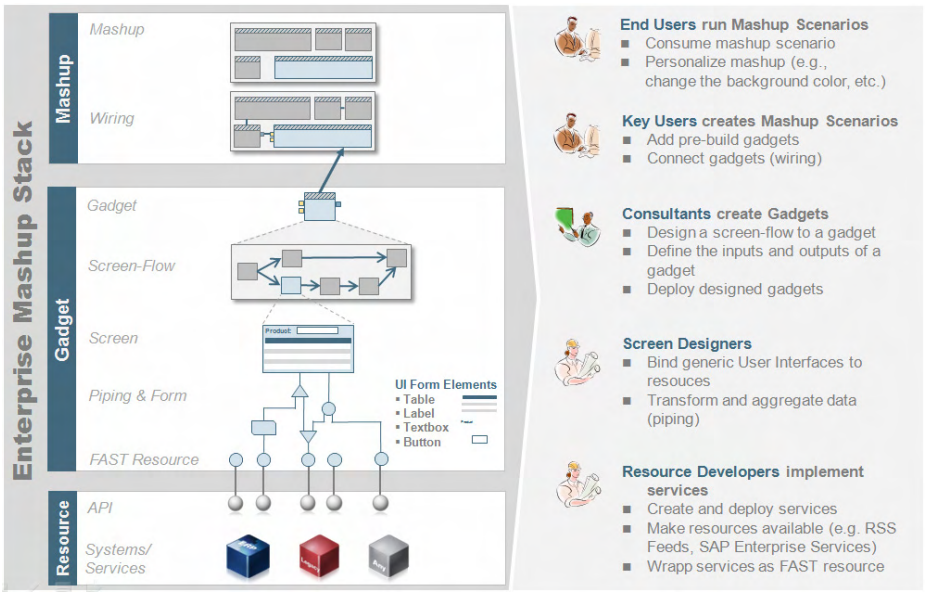


**Fig. 2.** Enterprise Mashup Development Layers, Terminology and User Roles

The relevant architectural components of the enterprise mashup paradigm can be structured in an enterprise mashup stack comprising three main layers (mashup, widget, resource) [5, 12]. On the gadget layer (visual building blocks), we introduce the concepts of screens and screen-flows in order to create powerful gadgets for enterprise purposes. Fig. 1 depicts the resulting terminology which is applied in the FAST project and in this paper. In addition, the relevant user roles including their tasks are mapped to the different architectural terms.

***Resources*** *(services)* contain content, data or application functionality and represent the core building blocks of enterprise mashups. They are encapsulated via well-defined public interfaces (*Application Programming Interfaces*; i.e., WSDL, RSS, Atom Feeds, etc.) allowing for a loose coupling of existing resources – an important feature from the SOA paradigm. These resources are provided by enterprise systems or by external Web providers (i.e., Amazon, Google, etc.) and are created by traditional developers who are familiar with development concepts.

The layer above contains ***gadgets*** *or widgets* which provide a simple user interaction mechanism abstracting from the complexity of the underlying resources. Thereby, the *piping* composition integrates heterogeneous resources by defining composed data processing chains concatenating successive resources. Aggregate, transform, filter or sort operations adapt, mix, and manipulate the content of the underlying resources. A graphical user interface *form* is put on the composed resource. The combination of a form and the piping composition is called a *screen* which is created by the screen designer. This user role is characterized by basic programming skills in order to bind the resources to user interfaces. Screens are fully functional by themselves, and their pre- and post-conditions drive the transitions between them to tie them together, forming a *screen-flow*. A FAST gadget consists of various screens and allows the handling of lots of information in several steps. In a similar way to the resource, input and output ports of a gadget (so-called events and slots) can be defined by a consultant (gadget developer). In addition, the user playing the consultant role is able to deploy a gadget to different mashup platforms. A consultant plays a primary role in IT departments and works quite closely together with key users from the business units.

Now, a key user who understands the business challenge is able to combine such visual gadgets in a mashup platform according to their individual business needs, thus creating a ***mashup***. This visual composition by linking the in-/ outports of a gadget is called *wiring* and requires no programming skills. Finally, the end users consume and run the created mashup scenario. If necessary, they are able to configure the mashup to some extent, e.g. (de)activation of functionalities, moving gadgets, etc.

In summery, the composition principle of the resource layer of traditional SOA environments is transferred to the user interface level where the end users are empowered to create an ad-hoc enterprise-class application. The power of the composition and also the required IT skills are different. A first discussion regarding the composition pattern in enterprise mashup environments can be found at [14].

### 2.2 Requirements

The existing discussion of the mashup principle in the scientific community is driven by technical aspects. In particular, several research activities deal with the lightweight

provision of IT-enabled components [15, 16] as well as their composition on the resource layer [17, 18]. Coming instead from a business perspective, researchers also started to analyse the underlying structure of the resulting open mashup ecosystem [19] and derived first managerial implications for API providers.

However, the discussion about the layers on top of the resources is still missing. By means of a literature analysis of market research institute reports [1, 4, 8, 20] and by taking experiences from first mashup implementations into account [21, 22], we identify the following open challenges concerning the enterprise adoption of the mashup paradigm. They are clustered in three dimensions: technical, organizational, and business perspective.

**Table 1.** Challenges of Enterprise Mashups

| Challenges | Description |
|---|---|
| **Technical Perspective** | |
| Interoperability | ▪ Discovery and composition of gadgets<br>▪ Underlying information model for in-/ output parameters for wiring gadgets |
| Gadget Portability | ▪ Moving gadgets between different mashup environments<br>▪ First standardization activities (e.g., OpenAjax, OpenSAM, OpenSocial, DataPortability, etc.) |
| Information Security | ▪ Gadget-to-resource security<br>▪ Single Sign On (SSO) to multiple company internal and external component sources<br>▪ AJAX Web browser-based mashup execution engine |
| **Organizational Perspective** | |
| Availability of Components | ▪ Integration in the existing IT infrastructure (legacy enterprise systems)<br>▪ Creation of enterprise-class gadgets representing the actual content of enterprise mashup platforms |
| Governance | ▪ Managing grassroots and community-driven mashup environments<br>▪ Balancing between organization concerns such as manageability and fostering user involvement |
| Culture | ▪ Exploitation of enterprise mashups to the right user groups<br>▪ Users have a new kind of freedom |
| **Business Perspective** | |
| Building the Business Case | ▪ Business value for enterprises and the users to introduce the mashup paradigm<br>▪ Providing key performance indicators for the IT management |
| Use Cases | ▪ Real-world scenarios demonstrating the potential |

In course of this paper, we focus on the technical interoperability, gadget portability, and the availablity of gadgets.

# 3 FAST Platform

## 3.1 Enterprise Mashup Life Cycle

Before elaborating on the actual architecture of the FAST platform, we introduce a life cycle for enterprise mashups in order to understand how enterprise-class gadgets are designed and executed. The model is organized by means of two dimensions. First, according to the terminology as presented in Sect. 2, the relationship between the mashable components and the related user roles are structuring the vertical axis. Second, the horizontal axis focuses on the actual life cycle of mashable components. Thereby, each component of the enterprise mashup stack (*mashup*, *gadget*, and *resource*) goes through the four phases *design*, *store*, *deploy*, and *execution*. The resulting enterprise mashup life cycle is depicted in the figure below.

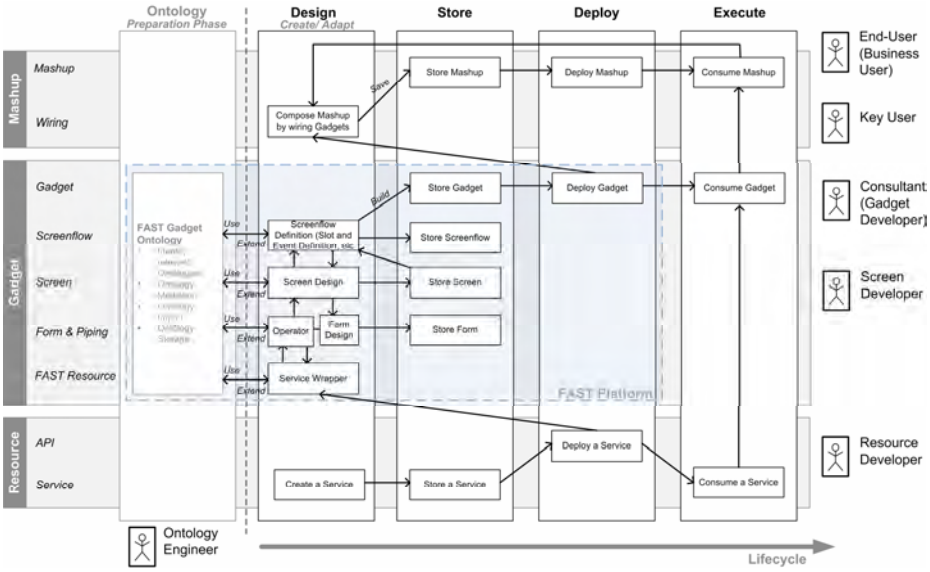


**Fig. 3.** Enterprise Mashup Life Cycle

As already mentioned, the FAST platform leverages semantics in order to hide the composition complexity from the users. Therefore, in a *preparation phase*, an ontology engineer identifies relevant domain specific ontologies. After importing of and mediating between existing ontologies, the FAST gadget ontology is used in the actual *design phase* by the users. On the other hand, users are able to extend the FAST gadget ontology with new instances by using the FAST platform. Mashable components can be annotated with additional semantics – in the FAST scope this is done by the consultants (gadget developers) and the screen developers. After finishing the design of a mashable components (screen-flow, screen design, form design, piping operation, and service wrapper), the persistence is handled in the *store phase*. A catalogue provides a URI to access the components and also allows the reuse of it during

the design phase. In order to consume one of the three executeable components (resource, gadget, mashup), the *deployment phase* takes care of the publication to external platforms. In context of a gadget, the FAST platform provides a set of potential target mashup environments (enterprise, social, mobile, and desktop environments). Now, the key user is able to compose deployed gadgets with each other (design phase of the mashup layer). Finally, in the *execution phase*, the gadget in a mashup scenario is consumed.

As depicted in Fig. 3, the enterprise mashup life cycle is characterized by permanent loops between the different phases of the life cycle. The result of the FAST platform is self-contained gadget, i.e., a piece of code that is executeable without using the infrastructure of the FAST platform in the execution phase.

### 3.2 FAST Gadget Ontology

One of the aspects that set FAST apart from other platforms is that the aim is to create what we call intelligent or smart gadgets. This means that the individual gadgets, as well as the reusable parts they are composed of, are formally described, using terms from a common ontology, the *FAST gadget ontology*[2]. It addresses the interoperability challenges as identified in the requirement section. These formal descriptions are utilised in different ways. (i) The *inputs and outputs* (pre- and post-conditions, respectively) of gadget components (e.g., screens) can be matched automatically. This enables the FAST platform to suggest screens which can be connected to other screens, or which screens are missing from a screen-flow in order to make it executable. (ii) *User preferences and current work context* which are equally described in terms of the ontology can be matched with the gadget and component descriptions, in order to suggest the right building blocks for a given task. (iii) *Descriptions of existing third-party resources* can be mapped to the FAST gadget ontology in order to make them available to the FAST platform.

In terms of its domain model, the FAST gadget ontology covers components on all levels of granularity – from complete screen-flows over screens and operators down to individual UI elements (as outlined in Sec. 2) –, *backend services* which provide data and functionality to screens and *users* of the FAST platform (see Fig. 2). The ontology is formalised using the Resource Description Framework (RDF) with OWL-DL semantics [23]. Classes and properties which are unique to the FAST platform (e.g., screens and screen-flows) have been modelled in the dedicated FAST namespace, whereas more generic terms (e.g., users, properties for annotation) have been adopted from external vocabularies and ontologies, such as FOAF and Dublin Core. For an in-depth discussion of the FAST gadget ontology we refer the reader to [24], which includes details on the methodology adopted for its development, the scope and domain model and a complete list of classes and properties with documentation.

However, at this point we would like to highlight a central feature of the ontology, namely the modelling of pre- and post-conditions, which is crucial both for the composition of screen-flows, as well as their excecution. Each such condition is expressed as a graph pattern, i.e., a set of one or more RDF triple patterns. The patterns of post-conditions will be instantiated into a common RDF graph, while the patterns of

[2] http://purl.oclc.org/fast/ontology/gadget, last checked 2009-08-13

pre-conditions will be executed as SPARQL queries against this graph to determine if they are fulfilled. For example, if the pre-condition of a product selection screen P is that a user has successfully logged into the system, then this could be expressed as simple graph pattern saying *"There is a resource of type `sioc:User`"* as follows[3]:

```
?user a sioc:User.
```

Now, if the post-condition of any screen currently present in the screen-flow contains this pattern (e.g., from a login screen L), then P is executable. Obviously, graph patterns can be more complex. We could imagine that the post-condition of the login screen L is *"There is a user resource which has an account name. There is also a person resource which has a name, and which has the user resource as its online account"*. Formally in FAST, this could be expressed as follows:

```
?user a sioc:User;
  foaf:accountName ?account_name.
?person a foaf:Person;
  foaf:holdsAccount ?user;
  foaf:name ?person_name.
```

In defining pre- and post-conditions of screens in this way, the FAST platform is capable of suggesting to a user which screens out of the set of available screens could be added to a given screen-flow during its development, or which of the screens already present in the screen-flow are executable or not.

### 3.3 Architecture

In order to support the presented enterprise mashup life cycle and FAST gadget ontology, we have designed a high level architecture of the FAST platform. By using the Fundamental Modeling Notation (FMC), we model the architectural components, their relationships and how the different user roles interact with the platform. In contrast to the technical-oriented UML notations, FMC focuses on human comprehension of complex systems[4].

Taking into account that the main objective of FAST is to allow users to compose gadgets from reusable building blocks and deploy them on various mashup platforms, the most natural mean is providing a rich internet application. Therefore, we have devised a robust architecture comprising the FAST client running on a Web-based FAST client, which deals with user interactions, and the FAST server, which takes care of the semantics, the storage capabilities and the deployment to external parties. Fig. 4 depicts the resulting FAST architecture.

The FAST client, which is called *Gadget Visual Storyboard (GVS)*, consists of three main architectural components.

**Workspace Manager (GUI).** This component is responsible for building and rendering the user interface and then populating it with the pieces required for designing an enterprise-class gadget. The AJAX-based user interface is composed by several areas: the building block palette, which shows a domain-specific subset of the existing building blocks stored in the server-side catalogue; the design area, in which the user

[3] Using SPARQL notation and terms from the SIOC and FOAF vocabularies.
[4] http://www.fmc-modeling.org, last checked 2009-08-13

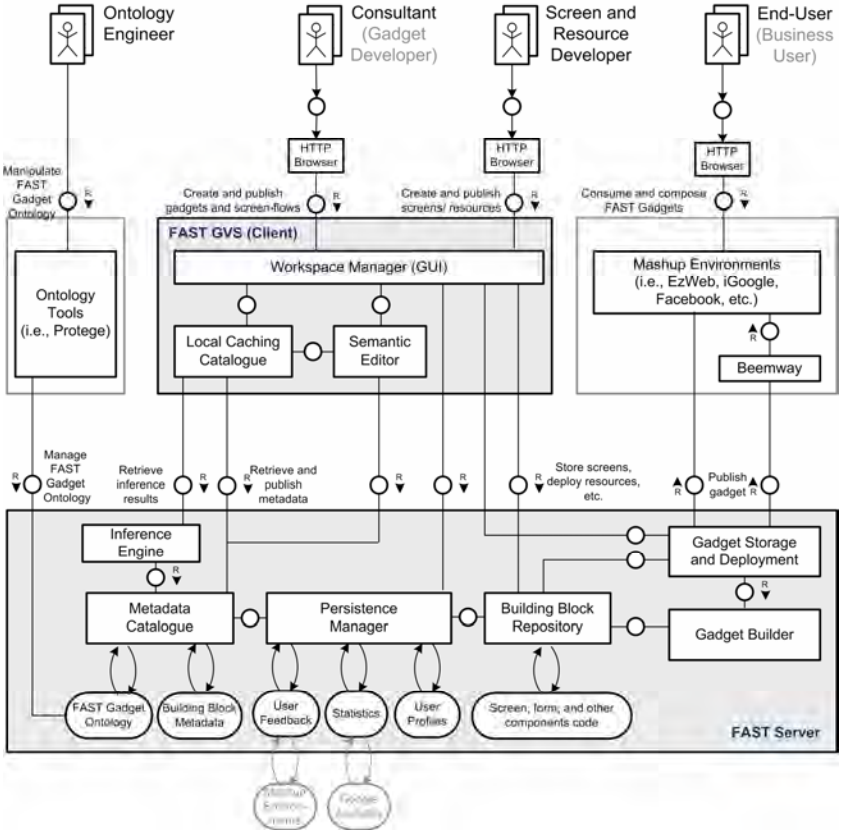


**Fig. 4.** FAST High Level Architecture (FMC Notation)

composes the gadget by mixing the pieces coming from the palette in a visual manner; and finally, a number of property editors and inspectors which show to the user the most relevant information about the screen-flow (or the screen). Fig. 6 in case study section depicts a screenshot of the FAST GVS user interface.

**Local Caching Catalogue.** The local catalogue retrieves and caches building block metadata coming from the FAST server metadata catalogue and being used for the designing of a gadget (or another lower-level piece, such as a screen). Moreover, this component provides the workspace manager with recommendations about what building blocks to use among other assistive features. These recommendations are provided by the server-side inference engine which is described below.

**Semantic Editor.** Building block reuse is empowered by the exploitation of semantics. Therefore, during their design and creation, it is necessary to use the existing semantic information and important to further enrich the elements being composed with semantic annotations. The semantic editor component allows the user to perform this duty in an integrated and user-friendly fashion.

The FAST server in the backend implements a REST API that offers the required functionality to deal with building block management, workspace persistence, gadget storage and its deployment. Additionally, the open APIs allow the integration of required third party tools (i.e., Protégé for managing domain-specific ontologies). In order to request information about the mashable components, we provide JSON and RDF/XML payloads. In particular, JSON reduces the programming effort in the FAST client. The FAST server-side itself is modularized into several cohesive components allowing independent development, even using different technologies. The main components are explained below.

**Metadata Catalogue.** The FAST metadata catalogue is in charge of the storage and indexing of information about every piece of a gadget, ranging from components such as screens or screen-flows all the way down to ontology terms describing the scope of a gadget. The structure of these components is formally defined in the FAST gadget ontolgy. Hence, every element in the catalogue is an instance of a concept from the FAST gadget ontology (or any other ontology), or indeed the ontology terms themselves. Consequently, its three main purposes are: (i) finding the most relevant building blocks for a given context (domain, user preferences and current workspace). (ii) The support for social interactions allowing community enrichment of the mashable component base (see semantic editor of the FAST client). (iii) The ability to deal with different domain-specific building blocks allowing users to create enterprise-class gadgets. In order to appropriately infer within those different domains, different ontologies must be used by the metadata catalogue. The main problem is that the most valuable gadgets usually are created by mashing up several application domains, so the catalogue component is also designed to manage the relationship between different ontologies (i.e., using ontology mapping techniques). The metadata catalogue is based on the RDF repository Sesame 2 which also provides a RESTful HTTP interface for SPARQL Protocol for RDF. As an abstraction to access the triple store, the RDF2Go library is integrated.

**Inference Engine.** Due to the importance of semantics, we distinguish the inference engine as the sub-component responsible for reasoning. It allows extracting and deriving new information given a certain knowledge base. It interacts directly with the triple store of the metadata catalogue and follows a forward-chaining policy, hence whenever new data is added to the catalogue, it also triggers a set of rules, and newly inferred data is added to the catalogue. Following a forward-chaining policy in the metadata catalogue makes sense, because it allows for faster query answering which is crucial for the performance of the overall FAST platform. Insertion of new data which would be favoured by backward-chaining is much less crucial in FAST. The set of rules being used by the inference engine is composed by a subset of the RDFS entailment rules[5] and the inverse of some of these rules.

**Persistence Manager.** The persistence manager is responsible for storing the relevant information between different browsing sessions. It stores user information (e.g., user profile) and settings, some usage statistics and user feedback, which can be used by the metadata catalogue to retrieve building blocks more accurately. As indicated in Fig. 4, data from external systems such as Google Analytics for monitoring designed

[5] http://www.w3.org/TR/rdf-mt/#RDFSRules, last checked 2009-08-13

and deployed gadgets as well as user feedback from the runtime environments (mashup platforms) is integrated in the persistence manager and therefore in the FAST ecosystem.

**Building Block Repository.** Once a component, for instance a screen or a screen-flow, is designed it must be stored in order to allow reuse at a later stage or even to create gadgets. The building block repository component is responsible for managing the existing building blocks' implementation. The acutal metadata is stored in the catalogue. By doing so, we separate between the actual code and metadata of the mashable components in the FAST platform.

**Gadget Builder.** When consultants finish their work and decide to create a gadget to be used in a mashup platform for execution, it is necessary to package the final gadget's code. It is the actual implementation of the designed functionality by using the modelled building blocks. The gadget builder is triggerd by the workspace manager of the FAST client and deals with this task. It processes each of the building blocks to create its associate code, and setting the defined relationships between them. The result is a self-contained, platform-independent gadget.

**Gadget Storage and Deployment.** The gadget code is stored and automatically adapted to the different mashup environments and their specifics. By attaching to the gadget's code the platform-compliant implementation of the target gadget API, the FAST gadget can be executed. The next section explains the FAST deployment concept in more detail.

### 3.4 Deployment of Multi-channel Gadgets

The final output of the FAST gadget development process is a gadget, which needs to be first stored and subsequently deployed to a chosen target destination, such as a start page (e.g., Netvibes, iGoogle, etc), mobile device, social networking site (e.g, Facebook, Bebo, etc), desktops of operating systems (e.g.: Windows Vista, Safari, etc) and finally, enterprise mashups (EzWeb).

In order to allow gadget deployment in one or multiple target platforms we have designed a flexible runtime gadget architecture. To achieve this platform independence, an important architectural design decision taken was to have the three layered approach as depicted in Fig. 5: The first layer corresponds to the *screen-flow implementation* of a specific enterprise-class gadget created by a user. The FAST platform empowers the user with the capability of emulating the runtime execution of the screen-flows, thus allowing for the experimentation of the final gadget. However, it is necessary the existence of a runtime execution environment, which corresponds to the other depicted two layers:

- *FAST Gadget Player*. This player enables building block interactions and guides the execution flow from one screen to another and keeps track of the facts.
- *FAST Gadget API*. This layer is responsible for the actual abstraction of the target destination mashup platforms.

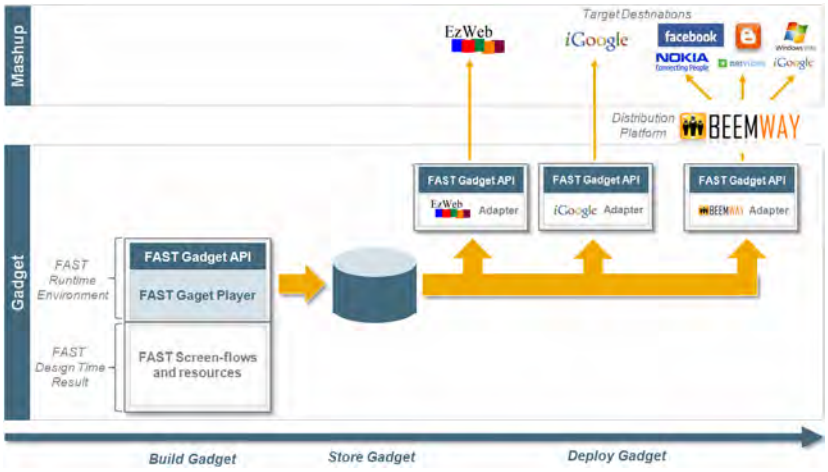


**Fig. 5.** Multi-Channel Gadgets

The three step deployment process begins after the FAST gadget has been created:

- **Build.** The first phase consists of packaging the complex gadget, namely the screen-flows and the corresponding resources, into a runtime environment that will execute independent of the FAST platform.
- **Storage.** With regards to deployment, it is important to take into consideration that most target destinations do not support the actual storage of gadgets. Therefore, the target destination usually keeps track of the URL where the gadget is stored. Consequently, once the gadget is built, it is placed within a repository.
- **Deployment.** This phase focuses on the placement of the gadget within the target designation platform, using the URL of where the gadget is stored. The actual deployment can take two alternative paths. First, the gadget is installed directly into the target destination platform by using an adapter. Second, the gadget is deployed via a distribution platform (e.g, Beemway[6]), which transparently installs the FAST gadget onto multiple destinations thus supporting the paradigm of build once, run everywhere.

## 4 Case Study: Cross-Organizational Promotion Scenario

After elaborating on the technical issues of the innovative FAST platform, this section is devoted to demonstrate the business value by means of a case study. Our demo scenario covers the usage of the FAST platform during the design, storage and deployment phase of the gadgets and the EzWeb[7] mashup platform for the building and execution of enterprise mashups in a cross-organizational context. The business scenario is involving two companies: The first company is a promotion agency

[6] http://www.beemway.com, last checked 2009-08-13
[7] http://ezweb.morfeo-project.eu, last checked 2009-08-13

*PromoBueno*, a SME company with 47 employees located in Madrid, Spain. The company offers different services to its customers, i.e. the organization of promotion activities at fairs/ events, brand promotion and marketing campaigns, etc. *PromoBueno* uses the FAST platform to develop gadgets to make their internal work more efficient and also to enable its customers to place promotion requests directly at them by using FAST gadgets. The latter gadget will be developed and published to a publicly available enterprise mashup platform, EzWeb, and allows interested customers of PromoBueno to create promotion requests and send them directly to PromoBueno. Figure 6 depicts a screenshot of the FAST prototype on how to define a screen-flow (in this case it consists of two screens - "available crew" and "incoming request") and on how to deploy the gaget to mashup platform (EzWeb).

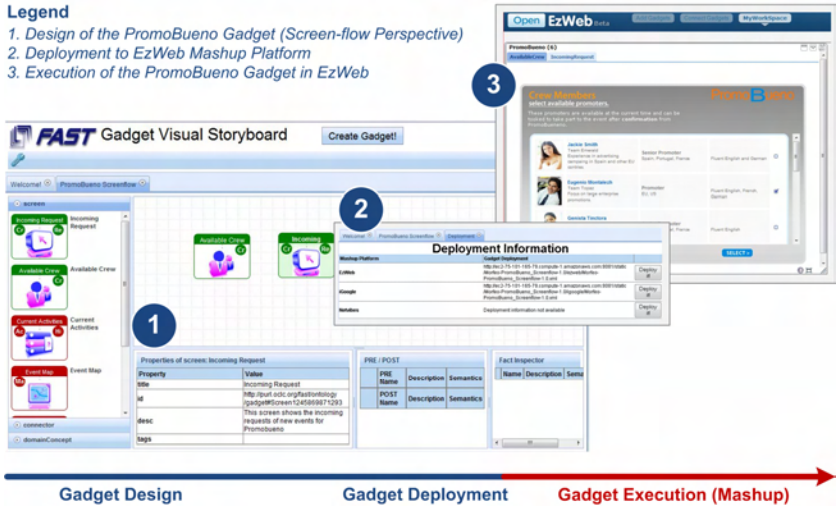


**Fig. 6.** Design, Deployment, and Execution of an enterprise-class Gadget

Now, the second company in our scenario (*AllSports*) is a sports equipment and nutrition producer, a large enterprise with 3227 employees located in Hamburg, Germany. Recently, *AllSports* created a new protein bar for high endurance athletes. The sales and marketing departments arrange and organize so-called point-of-sale (POS) promotion activities supported by different gadgets via the company internal enterprise mashup environment. When *AllSports* decides to introduce and sell their new product in Spain, they are interested in collaborating with different promotion agencies to request support for promotion activities at trade fair events. It is important for them that they can quickly establish the promotion request process with new agencies, as there are many available.

The created gadget can be used as follows to support the interconnection and collaboration between the two firms. A sales employee of *AllSports* has the need to request a promotion crew for a sport event. As it is her first time of organizing a booth at a fair in Spain, she needs the help of a local promotion agency. The sales employee of *AllSports* searches the gadget catalogue of the EzWeb platform and finds the

published “Promotion Request Gadget” of PromoBueno. The integration of the new “Promotion Request Gadget” is done by a key user of *PromoBueno*. The sales employee now carries out the POS process including the booking of the event and also the staffing of the promotion crew directly via the gadget of the promotion agancy. The promotion manager at *PromoBueno* gets the incoming request displayed at her monitoring gadget and can send a confirmation back to *AllSports*.

## 5 Conclusion and Future Work

The aim of this paper is the design of an open and semantically-enriched platform which allows creating enterprise-class and multi-channel gadgets. In order to achieve this, first, we introduced the main terms related to enterprise mashups and identified the challenges in order to transfer consumer-driven mashup paradigm to corporate environments. In a second step, we present the FAST platform. By means of a life cylce model, the relationship of the mashable components of an enterprise-class gadget is described. The FAST gadget ontology and the resulting software architecture are presented. Finally, a first implemented mashup scenario in the marketing/ promotion event area demonstrated the potential of the FAST platform.

Apart from other existing mashup and gadget platforms [4], the presented FAST platform aims at providing intelligent or smart gadgets by leveraging semantics. The followed multi-channel deployment approach allows the usage of designed FAST gadgets in various environments. For example, users from the business unit are empowered to develop and publish gadgets on their daily portal environment (EzWeb) and also on mobile devices without involving the IT department.

What is still missing is a general concept on how to integrate existing legacy enterprise systems in enterprise mashup environments. Currently, the consumed resources from backend systems (SAP Enterprise Service) are integrated manually. Future reseach will also deal with the implementation of a complete version of the marketing/ promotion event scenario that covers the ad-hoc interaction between several parties across company borders.

**Acknowledgments.** This work is supported in part by the European Commission under the first call of its Seventh Framework Program (FAST STREP Project, grant INFSO-ICT-216048) and in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2).

## References

1. Chui, M., Miller, A., Roberts, R.P.: Six Ways to make Web 2.0 work. The McKinsey Quarterly (February 2009)
2. McAfee, A.P.: Enterprise 2.0: The Dawn of Emergent Collaboration. MIT Sloan Management Review 47(3), 21–28 (2006)
3. Tapscott, D., Williams, A.D.: Wikinomics: How Mass Collaboration Changes Everythink, Portfolio, New York (2006)
4. Carrier, N., Deutsch, T., Gruber, C., Heid, M., Jarrett, L.L.: The Business Case for Enterprise Mashups, Web 2.0 Technology Solutions, IBM White Paper (2008)

5. Hoyer, V., Stanoevska-Slabeva, K.: Towards a Reference Model for Grassroots Enterprise Mashup Environments. In: Proceedings of the 17th European Conference on Information Systems, Verona, Italy (2009)
6. Cherbakov, L., Bravery, A., Goodman, B.D., Pandya, A., Baggett, J.: Changing the Corporate IT Development Model: Tapping the Power of Grassroots Computing. IBM Systems Journal 46(4), 743–751 (2007)
7. Gootzit, D., Phifer, G., Valdes, R., Drakos, N., Bradley, A., Harris, K.: Hype Cycle for Web and User Interaction Technologies, Gartner Research G00159447 (2008)
8. Young, O.G.: The Mashup Opportunity: How to make Web 2.0 work, Forrester Resesarch, May 6 (2008)
9. The Economist Intelligence Unit: Serious Business – Web 2.0 goes Corporate, Report of the Economist Intelligence Unit (2008)
10. McKinsey Global Survey Results: Building the Web 2.0 Enterprise, The McKinsey Quarterly (2008)
11. Hoyer, V., Fischer, M.: Market Overview of Enterprise Mashup Tools. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 708–721. Springer, Heidelberg (2008)
12. Hoyer, V., Stanoevska-Slabeva, K., Janner, T., Schroth, C.: Enterprise Mashups: Design Principles towards the Long Tail of User Needs. In: Proceedings of the IEEE International Conference on Services Computing, Honolulu, Hawaii (2008)
13. Daniel, F., Matera, M., Yu, J., Benatalla, B., Saint-Paul, R., Casati, F.: Understadning UI Integration. A Survey of Problems, Technologies, and Opportunities. IEEE Internet Computing 11(3), 59–66 (2007)
14. Janner, T., Siebeck, R., Schroth, C., Hoyer, V.: Patterns for Enterprise Mashups B2B Collaborations to foster Lightweight Composition and End User Development. In: Proceedings of the IEEE 7th International Conference on Web Services, L.A, CA (2009)
15. Abbott, R.: Open at the Top, Open at the Buttom; and continually (but slowly) evolving. In: Proceedings of the IEEE Conference on Systems of System Engineering (2006)
16. Pautasso, C., Zimmermann, O., Leymann, F.: RESTful Web Services vs Big Web Services: Making the Right Architectural Decision. In: Proceedings of the 17th International World Wide Web Conference, Beijing, China (2008)
17. Maximilien, E.M., Hernan, W., Nirmit, D., Stefan, T.: A Domain-Specific Lanaguage for Web APIs and Service Mashups. In: Proceedings of the 5th International Conference on Service Oriented Computing (2007)
18. Rosenberg, F., Curbera, F., Duftler, M.J., Khalaf, R.: Composing RESTful Services and Collaboration Workflows. IEEE Internet Computing 12(5), 24–31 (2008)
19. Yu, S.: Innovation in the Programmable Web: Characterizing the Mashup Ecosystem. In: Proceedings of the 2nd International Workshop on Web APIs and Services Mashups (2008)
20. Bradley, A.: Addressing the Seven Primary Challenges to Enterprise Adoption of Mashups, Gartner Research G00164390 (2009)
21. Hoyer, V., Gilles, J.T., Stanoevska-Slabeva, K.: SAP Research RoofTop Marketplace: Putting a Face on Service-Oriented Architectures. In: Proceedings of the 7th IEEE International Conference on Web Services (ICWS), L.A., CA (2009)
22. Lizcano, D., Soriano, J., Reyes, M., Hierro, J.J.: EzWeb/FAST: Reporting on a Successful Mashup-based Solution for Developing and Deploying Composite Applications in the Upcoming Web of Services. In: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, iiWAS (2008)
23. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantic and Abstract Syntax, Recommendation W3C (2004), `http://www.w3.org/TR/owl-semantics`
24. Möller, K.: Ontology and conceptual model for the semantic characterisation of complex gadgets, FAST Project Deliverable 2.2.1 (2009)