Tackling Interoperability in Composite Applications from an Enterprise Mash-up Perspective

David Lizcano[1], Javier Soriano[1], Rafael Fernández[1], Javier A. López[1], Marcos Reyes[2]

[1] *Universidad Politécnica de Madrid, Campus Montegancedo s/n, 28660 Boadilla del Monte, Madrid, Spain, {dlizcano,jsoriano,rfernandez,jlopez}@fi.upm.es*

[2]*Telefónica I+D, Emilio Vargas 6, 28043 Madrid, Spain, mru@tid.es*

Abstract

The idea of a global mesh of enterprise composite applications is being impeded by the delay of next-generation interoperable mash-up platforms, providing intuitive human-guided resource interaction, reuse and composition. In this paper we revisit current mash-up platforms and analyze their major shortcomings with regard to the development of real composite applications. Then, we elaborate on a novel reference architecture for a mash-up platform called EzWeb/FAST. EzWeb/FAST facilitates the establishment of a user-centered mesh of interoperable resources that can be composed flexibly and easily to create and execute composite applications based on complex mashupable gadgets. The release of EzWeb/FAST will help enterprises to realize how SOA front-end innovations could make them more productive, responsive and, ultimately, more competitive.

Keywords

Composite Applications, Mash-up, SOA, EzWeb/FAST, Web 2.0

1 Introduction

Web Services-based Service Oriented Architectures (SOAs) have attracted a great deal of interest over the last few years as they are expected to act as enablers of real Web-oriented composite applications [Alonso et al. 2004]. At present, however, these solutions mostly exist only within company boundaries. The global provision and consumption of compositional services over the Internet is still at an early stage [Gartner Inc., 2006]. High technical complexity, implementation and maintenance costs, inflexibility and the lack of widely accepted standards for defining service choreographies, as well as message semantics, have been repeatedly identified as key factors that have prevented the emergence of a mesh of interoperable Web Services that could foster development of enterprise composite applications [Schroth, 2007].

Apart from this, Web Services-based SOAs remain focused on automating service-to-service (i.e. machine-to-machine) collaboration, and they do not feature a “face” to human users, as they reside on a merely technical layer [Alonso, Casati, Kuno, Machiraju, 2004]. Another important shortcoming of existing SOAs is related to this gap between human users and services: the lack of a SOA front-end to humans is preventing enterprises from realizing how SOA front-end innovations help to make people more productive [Davenport, 2005].

The lack of next-generation portal mash-up platforms providing intuitive human-guided resource interaction, reuse and composition is, then, a major stumbling block on the path to this mesh of interoperable resources that could build web composite applications [Smith, 2006]. Content-driven mashup-oriented programming (a.k.a. situational programming or instant and composite programming) is a new agile application development paradigm in which users, who do not have previous coding skills but do have extensive domain expertise, visually assemble and combine off-the-shelf gadgets, i.e. discrete self-contained domain data-oriented components with both development (service and data binding and interconnection) and runtime rendering capabilities

[Schroth, Christ, 2007]. However, these "de facto" solutions still have several shortcomings when they are going to be used to create complex composite applications in real enterprise domains [Schroth, Janner, 2007].

With this in mind, we elaborate in this paper on the synergies between Web 2.0, the mash-up philosophy and SOA concepts with regard to the development of a composite application framework [O'Reilly, 2005], and we present a definite reference architecture to deal with this problem. First of all we revisit current mash-up platforms and analyze their major shortcomings with regard to the development of real composite applications. Then, we elaborate on a novel reference architecture for a mash-up platform. This platform facilitates the establishment of a user-centered mesh of interoperable resources that can be composed flexibly and easily to create composite applications. Finally, a later section concludes this paper and presents a brief outlook on future work.

2 Current mash-up platforms and their shortcomings on the road towards composite-applications interoperability

Current mash-up solutions allow users to create their own composite interface using multiple and disparate gadgets (iGoogle, NetVibes...) or mixing several data sources in order to create another one (Yahoo! Pipes). But the use of the components of almost all of these solutions is confined to platform boundaries (i.e. there is no room for interoperability with other applications: it is impossible to add an iGoogle gadget to the NetVibes homepage). Additionally, a gadget within a given platform cannot communicate with other platform elements. These gadgets are isolated applications put together in the same window, and they cannot, therefore, be considered as composite applications. So, it is almost impossible to build real and complex applications, which need different gadgets to share data and functionalities. The main difficulties a developer has to deal with to build gadget-based applications are:

- Even though there are a few platforms in which is possible to include some hacks to achieve some sort of gadget communication, most of the mash-up platforms do not support (or even allow) intergadget communication. They do not provide an API or core functionalities to share either data or code between the gadget instances included in the user window. There is another barrier to sending and receiving data between gadgets: shared data types. If two gadgets were able to send data to each other, it would be necessary to agree on the shared data types not only at a syntactic level (i.e. the implementation data type, like strings or integers), but also at a semantic level (the intended concept represented by the data type). There should be ontologies (or some other kind of shared conceptualization), including the different concepts (data types) to be shared by gadgets. Without an agreement on data semantics, gadgets will not able to share data, and it will be impossible to build complex applications.
- Gadgets are considered as the new face on SOA [Schroth, Christ, 2007] and, if they are going to be used to create complex applications, they must access and interact with remote data sources and services (i.e. Web resources). Current data sources do not have a friendly interface (like Web services), meaning that they cannot be used easily, or do not have a standardized uniform access (Web contents), meaning that, to be used, they have to be hardcoded into the gadget itself. It is very hard then to create generic gadgets that can access any given resource selected by a user (or even develop gadgets uniformly). One of the main benefits of implementing a SOA is the reusability and composability of its components, and this will not be possible with current gadget architecture.
- Another key principle of SOAs is to have a shared resource catalogue, like UDDI [OASIS, 2004] to facilitate service composition and reusability. Currently, there is no centralized shared gadget catalogue enabling users to create complex items based on simpler ones (following the Web 2.0 mash-up phenomenon [O'Reilly, 2005]). Therefore, it is very

difficult to create gadget-based applications, because they have to be built from scratch with no reuse or adaptation of existing gadgets.

- There is no gadget standard that includes all the requirements for building composite applications. Every mash-up platform has its own gadget model, and there have been only few attempts to create gadget standards (e.g. UWA [NetVibes 2007], tries to create platform-independent gadgets). None of them considers the complexity of a composite application made up of a number of gadgets sharing data, using and modifying remote data sources or services, and interacting with other gadgets. There should be a formal definition of the gadget model, including the definition of its communications, shared data, connections with remote resources and so on in order to achieve interoperability and automatically compose gadgets.

3 Reference architecture for real interoperability in mashup-oriented composite applications

Following on from this, we present a reference architecture of an open Enterprise 2.0 mash-up platform that empowers its users to co-produce and share instant composite applications [McAfee, 2006]. This architecture will help to evolve current Web solutions with an approach in which all the stakeholders will be able to collaboratively develop capabilities and innovate operating procedures by remixing and integrating already available services [Gartner Inc., 2006] through the core ideas of enterprise mash-ups and collaborative resource repositories [Smith, 2006].

Following this reference architecture, user-service interaction must embrace a number of principles to ensure the widest acceptance by end-users. The most important that we have identified when conceiving this architecture are:

- *Users must feel fully empowered* and able to serve themselves from available resources that provide them with access to the content and services they can use to set up their own personalized operating environment in a highly flexible and dynamic way**.**
- *Active user participation must to be enabled.* Users must be able to contribute new and improved versions of resources, as well as share further expertise about these resources, their use, and their interrelationships.
- *Community-based collaborations need to be fostered.* The introduction of a *share, reuse and assembly culture of collaboration* will boost and speed up this process thanks to a network effect [McAfee, 2006].

Based on these basic design principles, we have conceived a reference architecture. The key ideas of this architecture and their interrelationships are further explained in the following.

3.1 Unified access to components through RESTful resources

As mentioned above, enterprise front-ends usually provide users with data and functionalities from legacy IT through applications built on SOA [Schroth, Christ, 2007]. This architecture revisits SOA to focus on end-users as service clients by providing a RESTful [Fielding, 2000] data front-end, transforming this enterprise legacy into a uniform layer of human-accessible resources.

The architecture aims to allow users to hack (remix, compose and feed) these resources through a friendly and homogeneous user interface. This interface enables end-users to access (using social search) a collaborative catalogue of resources and BPM solutions, which will be further explained later.

Based on the above three design principles, the notion of *resource* emerges as the basic concept around which all the reference architecture is organized. Resources are simple components designed following the REST (representational state transfer) architectural style [Fielding, 2000]. A resource owns a URI (uniform resource identifier) that identifies the content or service it represents, and responds to the basic http verbs (i.e. get, post, put, and delete), thus wrapping the access to that content or service through a uniform interface.

Embracing the REST paradigm will imply incorporating the principles of complexity-hiding and uniformity since this paradigm significantly reduces programming efforts and forces both clients and resources to adhere to a common set of supported operations and interface descriptions.

Resources handle basic http requests by gathering and processing data (content or output from services), and delivering output to other resources or directly to the knowledge worker's browser. Some are designed to deal with presentation functions, i.e. in response to an http request, they produce output using a data format that can be directly integrated at the user interface level (e.g., they can produce data in a markup language that can be processed and rendered by browsers). Others are designed to just produce data that can be processed by other resources, and enable their remix and assembly. Ultimately, this means that resources are a standard means to offer a uniform front that end-users can "see and touch" (cf. Figure 1).

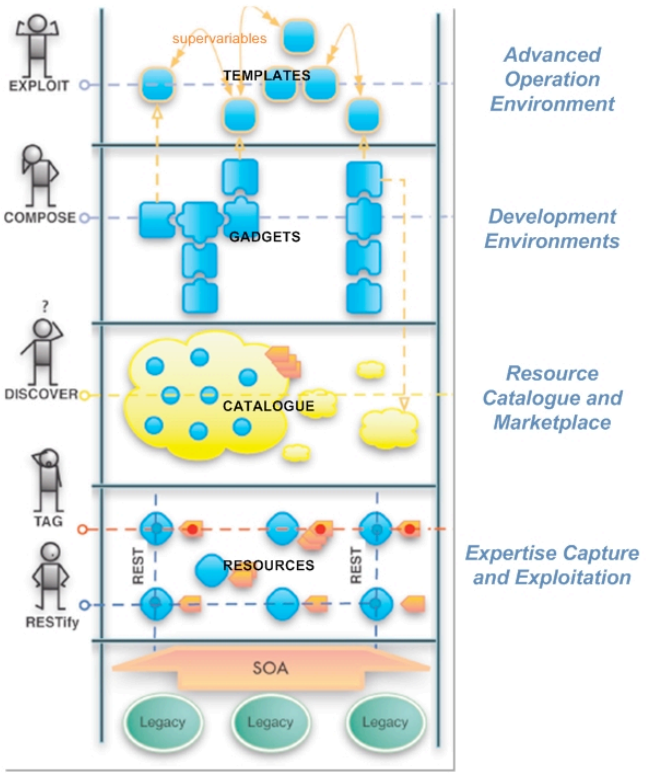

Figure 1: Proposed reference architecture

The conception of contents and services as resources adds a lot to the idea underlying traditional mash-ups (hybrid Web applications or services composed of contents and services from a range of disperse sources), as it provides uniform access to them all. From this viewpoint, the mash-ups will be created by composing and interconnecting these resources with a uniform interface without it being necessary to remix data, code or low-level services, as it is with heterogeneous interfaces.

3.2 Interoperability in the proposed reference architecture

In a mash-up platform, the way building blocks (that are combined to create composite applications) are related to other blocks and to their own platform is vital [Schroth, Christ, 2007]. As many use cases have illustrated, these resources must intercommunicate with each other to allow really powerful compositions and mashupping. Also they should provide a flexible and simple support for managing fixes, fits, and configurations of their own look, appearance or operating modes [Smith, 2006]. This section explains how our reference architecture supports these issues, fostering the remixing/fitting of resources through a simple representation of resources called template and empowering gadget-to-gadget and gadget-to-platform communication based on a programmatic idea called *super variables*.

3.2.1 Template-based interoperability provided by mash-up platforms to their own runtime components

Developers creating a new resource usually have to program this resource in the XHTML and JavaScript languages, following a set of rules established by a particular mash-up platform API. In our architecture this programming job is divided into two parts: create a resource body and elaborate an XML template that will allow this component to be added to each mash-up platform as a new fixable/mixable building block.

This template gathers all the implementation details that have nothing in common with the resource business layer, reducing the components code coupling. Bearing in mind the traditional object-oriented programming model called the model-view-controller (MVC), this template part could be considered as a mix of view-controller data set, while the resource body represents the model functionality. This separation is necessary to improve components creation productivity, improve performance and reduce changes and chaos in the event of resource modifications or hacking. Since, resources must, as a part of mash-up philosophy included in the revolutionary Web 2.0 paradigm, support changes, constant beta evolution and lightweight programming ideas [O'Reilly, Musser, 2006], the template is vital to allow components remixing, fitting, hacking and changing support.

Accordingly, templates contain nothing about how resources carry out operations or calculations, encapsulating only in/out information, how to feed and extract data to/from internal processes, and information about resource visualization and configuration. Considering all these design principles, templates should include details about (cf. Figure 2):

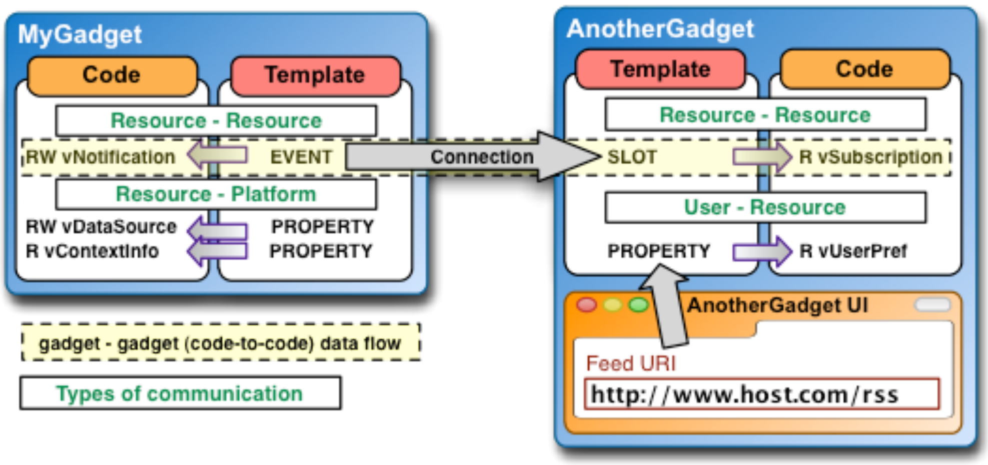

Figure 2: Proposed interoperability framework

- User preferences, which condition how resources work and what they look like. These preferences can be modified and saved through the template.
- Persistent properties governing resource behavior. These properties can be modified and read by components code, which interacts internally with the template.
- Intercommunication resource data. The template should include information about how a building block interacts with the platform and with other components. Specifically, this data is presented as:
  - Events propagated by the resource, which can be consumed by other resources or by the platform.

– Events that can be consumed by the resource, that is, there are rules and code that implement how a component should react and operate in the presence of these events. This information, related to intercommunication issues, will be explained in detail in the next section on *super variables*.

3.2.2 Interoperability between resources

As is mentioned above and further explained in this section, resources interact with each other and with the mash-up platform through *super variables* that are contained in the template. This *super variables* concept is a general mechanism used to integrate resources into their environment by creating data flows (weakly coupled) with other entities (platform or other resources) through special variables managed by the platform and stored in a component's template. Accordingly, a resource communicates with its environment through a basic mechanism based on import/export data. These *super variables* implement basic operations to manage data notification and receipt, and the persistence of this data through a typical get/set/callback interface that a resource's internal code can invoke. Based on the use of this interface, there are two subtypes of *super variable*: RW (Read and Write) and R (only read) variables.

With this general reference infrastructure, the mash-up platform can build high-level mechanisms to establish data flows between resources in a highly scalable way. These flows are transparent to resources and can effectively support their environmental integration. Examining this issue in more depth, there are three types of flows for the implicated roles:

1. **Resource – Resource Communication.** This scenario includes the publication of information for other resources through notification events (EVENT variables in templates, which are RW) and subscription to information that comes from other resource events, using SLOT variables, which are R.
2. **User – Resource Communication**. End-users can modify any user preference of any resource in their dashboards at execution time, using a UI generated by their own platform (which looks like a personalization web form). This scenario uses PREFERENCE variables that are managed and stored in each resource's template. These variables are R (they can only be modified through the platform's UI and not by internal code, which only can read this data).
3. **Resource – Platform Communication**. This scenario supports a vital communication flow between resources and the mash-up platform. This supports component data persistence (the resource template has PROPERTY RW *super variables* that manage persistent transactions), manages feedback information about mash-up execution and exploitation (through template-supported FEEDBACK RW variables) and, finally, accesses platform-created context information to supply specific resources with data (thanks to CONTEXT variables, that are R also, included in the component's template).
4. Briefly, these *super variables* solve all current issues about resource communication in a smart and homogeneous way. From the resource viewpoint, a new CONTEXT information item, new data incoming through a SLOT or changes made by end-users using PREFERENCEs are tackled in the same way, invoking a callback function of R variables. This way, these events are transparent to resources, and are managed automatically without further code support having to be provided. In fact, these different events can be interchanged without touching resource programming.

3.3 Interoperability resource publication, discovery and recommendation through a collaborative catalogue

If ungoverned and unsorted, the huge number of resources that are envisioned to be available in our user-centric reference architecture will become unmanageable and thus useless for end-users.

Dedicated catalogues will be required to provide navigation services for users and help them to find out which resources they need to create the composite applications they want. We propose to address this need by providing a user-generated, "living" catalogue of resources founded on the Web 2.0 vision for user co-production and harnessing of collective intelligence (cf. Figure 3). This would provide users with a collaborative Wiki, and tagging and searching-by-recommendation capabilities for editing, remixing and locating software components of their interest.

The catalogue sets out the expertise and knowledge available within a community for composing instant applications in a graphical and intuitive fashion.

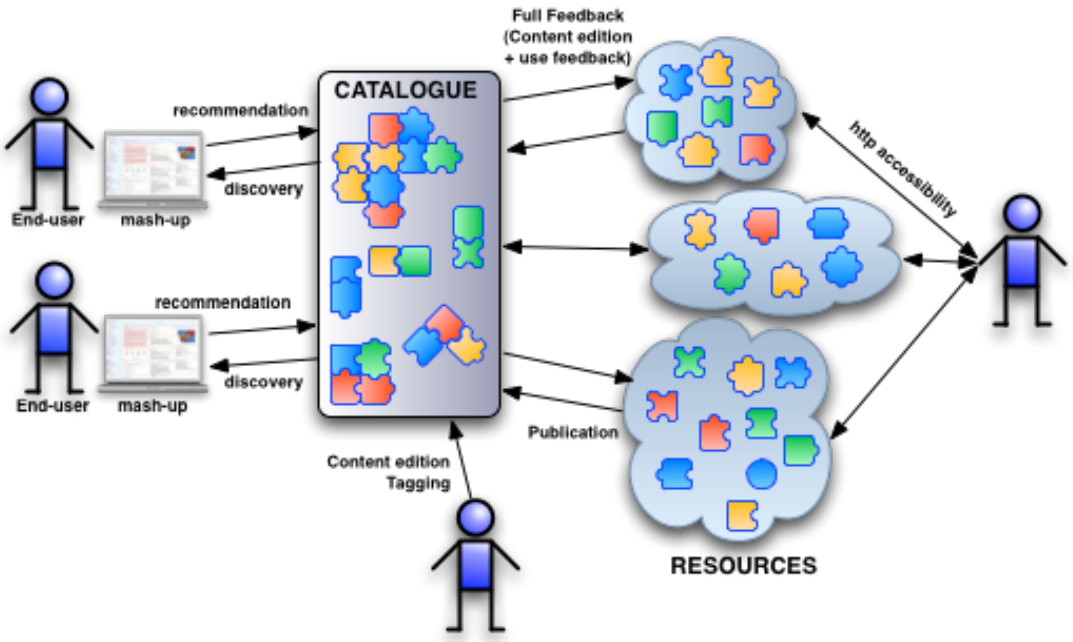

Figure 3: Cataloguing Resources

The catalogue manages two types of knowledge: a collective pool of resources with capabilities for cloning, modifying and publishing new resources, and a list of resources with the business processes they support. The catalogue allows users to create composite applications by just looking for (or being recommended) "pre-cooked" or off-the-shelf resources and customizing these resources to suit their personal needs, interconnecting resources, and integrating the outcome in their user workspace. "Folksonomies" of user-created tags will emerge, grow as users add information over time and act as important facilitators of our proposed reference architecture.

Earlier approaches to service discovery and description like UDDI are not adequate to support human beings in easy resource retrieval and evaluation. By contrast, the exploitation of collective intelligence and user-driven resource categorization is beneficial for users. From an enterprise viewpoint, catalogue accessibility and ease of use encourages all users to continuously improve their operating or desktop environments, taking advantage of the business knowledge generated from colleagues' viewpoints.

4 Conclusions and future trends

In this paper, we elaborated on the current mash-up platforms approach to composite application development as an alternative to traditional Web Services-oriented SOA for integrating distant and potentially heterogeneous Web-based resources. Existing mash-up platforms have neither lived up to their promise of facilitating a global network of loosely interconnected resources nor provided an appropriate framework for creating complex composite applications, which involve interoperability and communication issues. Consequently, new approaches are required to rise to this challenge of enabling a composite application framework based on Web-based interoperable resources.

User-centric mashup-oriented approaches to SOA, such as the reference architecture proposed in this paper, will be a major step forward, providing solutions to currently hard-to-solve problems in the traditional composite applications paradigm. The emergence of such platforms will solve key problems in two different scenarios. Companies may capitalize on faster application development, a more agile system landscape and the empowerment of their employees to design their own instant and composite applications that best satisfy their unique requirements, and to share this knowledge with other employees.

On the other hand, end-users benefit from a strongly increased capability of personalization and participation through our reference architecture. The proposed approach will provide users with intuitive, unsophisticated IT ways to discover, remix and use those Web-based resources that they consider interesting and useful and build new applications by interconnecting them. It will also allow them to participate, swap information with other users and service providers and actively contribute in a way that encourages extensive use of the resources offered. This speeds up the service innovation pace. Focusing on the "long tail" advanced by Chris Anderson rather than a limited number of sophisticated experts, the new composite applications paradigm will involve the bulk of private users or knowledge workers and allow for "customer self-service".

Future work will concentrate on evolving the proposed open source mash-up platform that builds on all the key architectural principles described above. It will also concentrate on developing and extending FAST: a framework for developing new complex gadgets and resources based on the ideas for interoperability contributed in this paper.

Acknowledgement

This work is being supported in part by the European Commission under its 7th FP (FAST Project, Grant Agreement No. 216048) and by the European Social Fund and UPM under their Research Personnel Training programs.

References

Alonso, G., Casati, F., Kuno, H. & Machiraju, V.(2004). Web Services Concepts, Architectures and Applications. Springer, 2004

Davenport, T. H.(2005). Thinking for a Living: How to Get Better Performance and Results from Knowledge Workers. Harvard Business School Press, Boston, MA, USA. 2005.

Fielding, R. T.(2000). Architectural styles and the design of network-based software architectures, Ph.D. thesis, University of California, Irvine, 2000

Gartner Inc. (2006). Hype Cycle for Software as a Service, Gartner Research, 10 August 2006.

McAfee, A.(2006). Enterprise 2.0: The Dawn of Emergent Collaboration. MIT Sloan Management Review, Vol.47, No.3 (pp. 21-28).Spring 2006.

NetVibes ® (2007). Universal Widget API specification, retrieved from http://blog.netvibes.com/?2007/03/09/125-new-developer-website-preview-of-universal-widget-api-uwa

OASIS ® (2004). Oasis UDDI specifications TC – Committee Specifications, retrieved from http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

O'Reilly, T.(2005). What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

O'Reilly, T. & Musser, J.(2006). Web 2.0 Principles and Best Practices. O'Reilly radar, November 2006.

Schroth, C. & Christ, O. (2007). Brave New Web: Emerging Design Principles and Technologies as Enablers of a Global SOA. In Proceedings of the 2007 IEEE International Conference on Services Computing (SCC 2007) (pp. 8): IEEE CS. Retrieved 2007-07-11, from http://www.alexandria.unisg.ch/publications/37038.

Schroth, C. & Janner, T. (2007). Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services. IEEE IT Professional Vol.9, No.3(pp.36-41) , June 2007.

Smith, R.(2006). Enterprise Mashups: An Industry Case Study. Keynote at the New York PHP Conference & Expo, Manhattan, New York, USA, 14-16 June 2006