REST Web Services in Collaborative Work Environments

Luis Oliva[a] and Luigi Ceccaroni[a]

[a] Departament de Llenguatges i Sistemes Informàtics (LSI), Universitat Politècnica de Catalunya (UPC), Campus Nord, Edif. Omega, C. Jordi Girona, 1-3, 08034 Barcelona, Spain

Abstract. This paper presents a case study for the development of an integration framework in the area of collaborative work environments. This framework is based on REST Web services, and has been designed and implemented as part of the Laboranova European project.

Keywords. Web services, AI applications, knowledge engineering, information processing, application integration

1. Introduction

1.1. Web Services

With the introduction of middleware technologies, component-based software development has become a major trend, at least when we focus on enterprise solutions. In addition, Web technologies as well as XML have gained broad application throughout the industry. In almost all solutions HTTP is leveraged as a bridge between the front-end Web browsers and Web servers, while components are used to implement workflow and persistent state in the back-end. Although the computation-driven back-end has always been subject to change, the front-end has remained almost unchanged: it uses a HTML-driven paradigm to transfer and display Web pages from Web servers to human users. The combination of component-based middleware and Web technologies in order to integrate business processes and applications has proved to be insufficient for many reasons. For instance, this type of simple integration approach does not consider issues such as integration of different data models, workflow engines, or business rules, to name a few. *Enterprise application integration* (EAI) solutions have become so widespread in B2B environments because they try to solve most of these issues. However, the available EAI solutions are proprietary, complex to use, and do not interoperate well with each other. Thus, a question arises: Is there a better way to solve the integration dilemma, with a simple and interoperable solution? [1][2] One solution, described here, is to use REST Web services.

Web-based network technologies have become increasingly important for IT solutions. This trend leads to fully connected information systems, but also causes a number of problems developers have to address. For example, all kinds of devices should be able to be connected to network-based systems. Software systems are expected to drive business processes that are no longer constrained by computer-related or company-related boundaries. Hence another question needs to be resolved in this context: How can we connect isolated islands to produce integrated solutions? [1][2] One way, again, is using REST Web services.

1.2. The Laboranova Project

Innovation has become a key element in the strategic agenda of the European Commission to foster competitive advantages in the global economy. Existing *collaborative working environments* (CWEs) are mainly focused on traditional working paradigms of linear workflows by providing IT-based platforms support for planning, scheduling and executing tasks. However, in order to achieve continuous innovation and thus create persistent, competitive advantage, organisations need to increase their capacity for carrying out open-ended and nonlinear problem solving involving a wide participation of people in knowledge-rich environments.

The Laboranova European project[1] is intended to create the next generation *collaborative tools* which have the potential to change existing technological and social infrastructures for collaborating and to support knowledge workers in sharing, improving and evaluating ideas systematically across teams, companies and networks. Laboranova is focused on the development of three specific areas (also called *ideation*, *connection* and *evaluation* spaces): tools supporting idea generation; tools connecting people and ideas among them; tools evaluating ideas.

The intended results of integrating these efforts are innovative collaboration approaches and organisational models for managing early innovation processes, software prototypes and the integration of the models and tools into a *collaborative innovation toolset*. This could change the way knowledge work is done and increase the innovative output of companies and organisations far beyond today performance levels. Specifically, the Laboranova platform aims to:

a) allow the deployment and running of tools developed within Laboranova as a *decoupled but interoperable toolset*, in order to better suit the changing and heterogeneous needs of Living Labs[2] communities;
b) enable the integration of heterogeneous tools, which could be either multi-user Web-based applications, or desktop applications;
c) enable the integration of tools developed within Laboranova with external tools, data repositories or content management systems;
d) support collaborative processes in an innovation context, by providing a rich user experience and seamless context-switch between tools, both desktop- and Web-based;
e) support seamless exchange of data between tools in order to achieve preservation of user context;
f) support scalability in number of users (e.g., communities of up to several hundred users);
g) support security and access control, providing a single authentication and authorization mechanism across all tools.

[1] The Laboranova research project [http://laboranova.com/] is funded by the European Commission within the 6th Framework Programme for RTD (contract IST-5-035262-IP).

[2] See [http://www.openlivinglabs.eu/], last visited on 2009-06-02.

1.3. REST vs. SOAP

The most common approaches to Web services are *simple object access protocol* (SOAP) and *representational state transfer* (REST). SOAP is usually combined with WSDL to define the services interfaces and UDDI to provide service discovery. REST considers Web services as resources, which are used through given representations. For a detailed comparison between these approaches, see the work of Pautasso et al. [3].

Because of Laboranova requirements, REST has characteristics that, in the context of the project, match technological needs better than SOAP, and these characteristics are described as follows. Because REST defines a uniform interface for any resource, it does not require WSDL to define it (although some sort of human-readable documentation is still needed). REST is a set of architectural constraints defined by Fielding [5][6], used to implement Web services. These constraints can be summarized in the following points: a stateless client/server protocol; a uniform interface; use of hypermedia; a universal syntax for addressing. These constraints allow creating Web services in order to expose a tool's API, while preserving decoupling, by means of its uniform interface and addressing system. Moreover, consuming services through its client/server protocol ensures the possibility to integrate different kinds of tools (from desktop applications to any Web-based tool) as long as they can consume the Web services provided. All needed is an address to access the Web service. By means of the hypermedia and the uniform interface, clients can browse all the services provided and can consume them without having to adapt the code to the service provider. Scalability is obtained thanks to the stateless nature of the architecture: as any request has to contain all the necessary information to be processed independently from any previous request, this allows the server to avoid storing any information regarding the client and its previous actions.

The minimal information unit in REST is a resource. Resources are data sources which store the functionalities and the application state of a system. "Anything that can be named can be a resource: a document, a temporal service, a collection of resources, a non-virtual object (e.g. a person), and so on. Any concept that might be the target of an author's hypertext reference must fit within the definition of a resource. A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time" ([5], p.88). Resources need to be uniquely named to be able to be identified; therefore they need to be addressed through some sort of *globally unique identifier* (GUID).

Finally, the simplicity derived from a uniform interface avoids burdening APIs, thus facilitating the maintenance and growth of services. REST applications run in the World Wide Web using the HTTP protocol to transfer data. It is not necessary to implement an additional layer over the ISO/OSI stack [4]. Furthermore, REST does not require any toolkit to be installed on client machines.

1.4. Laboranova Live

Laboranova Live is an integration environment that creates a space where a set of tools can share data and services to support the innovation process. This environment implements an integration architecture defined in the Laboranova project, and provides a common place to store and share data through a REST interface. Laboranova Live implements the REST architecture over HTTP/1.1 [7], which is the most known implementation of the REST architecture. HTTP is a stateless client/server transport protocol used for retrieving hypertext documents. This protocol defines eight methods (or verbs), though only four are generally used in REST services. These four methods are: GET, POST, PUT and DELETE, and allow implementing a CRUD (Create, Retrieve, Update and Delete) interface to access any hyperlinked resource. GUIDs are implemented using URLs, whose structure complies with [8]. The Laboranova Live framework also defines guidelines for tool developers to create Web services that can enrich and extend the set of services provided within it. These extensions allow different deployments of Laboranova Live while allowing Laboranova tools to remain a decoupled but interoperable toolset.

2. Technical Issues

Technically, the integration problem can be partitioned into a set of aspects emerging on different levels. The heterogeneity in these levels is primarily caused by different: network technologies, devices and OSs; middleware solutions and communication paradigms; programming languages; services and interface technologies; domains and architectures; data and document formats; ontologies. The problem becomes a multidimensional problem when integrated solutions have also to cope with non-functional requirements such as security aspects, availability, transactions, to name a few [1].

2.1. Levels of Integration

Laboranova Live defines four levels of integration (see **Figure 1**).

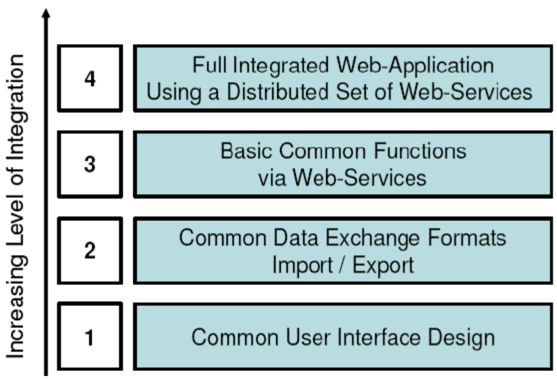

Increasing Level of Integration
4
Full Integrated Web-Application Using a Distributed Set of Web-Services
3
Basic Common Functions via Web-Services
2
Common Data Exchange Formats Import / Export
1
Common User Interface Design

Figure 1. Levels of integration. Source: Laboranova project

The *first level* establishes common user interfaces and usability guidelines to provide a homogenous user experience and similar look and feel.

The *second level* defines the set of resources which are shared by all Laboranova tools; this set can be modified or extended depending on which tools are part of a specific Laboranova deployment.

The *third level* is aimed to achieve data level integration by means of REST Web services [9] over a central *idea repository*. This level is based on a *service-oriented architecture* (see **Figure 2**). This approach allows Laboranova tools to exchange data with central repositories while keeping platform or language independency, thus facilitating interoperability.

The *fourth level* presents guidelines and best practices supporting the publication of Laboranova tools' own Web services. In this way Laboranova tools can provide specific content (not stored in the central repository) and services as stand-alone applications.

The base facets of Laboranova Web services technology stack are outlined in the rest of this section.

2.2. Idea Repository

This central repository stores the resources shared by all the tools that conform a Laboranova system. Each resource has two different representations: an XML representation to store and transfer data, and a read-only HTML representation to provide a human-readable documentation. There is no convention or standard way to document REST Web services. In Laboranova Live, Web services are documented using a human-readable approach, which is the most accepted approach in the REST community. Each resource provides a documentation containing at least the following information:

- name and a short description in natural language;
- URL that identifies it;
- for each representation of the resource:
  - MIME type and a short description in natural language (or an XML schema);
  - for each available method of the representation: name, and description of what it does; HTTP response code and message.

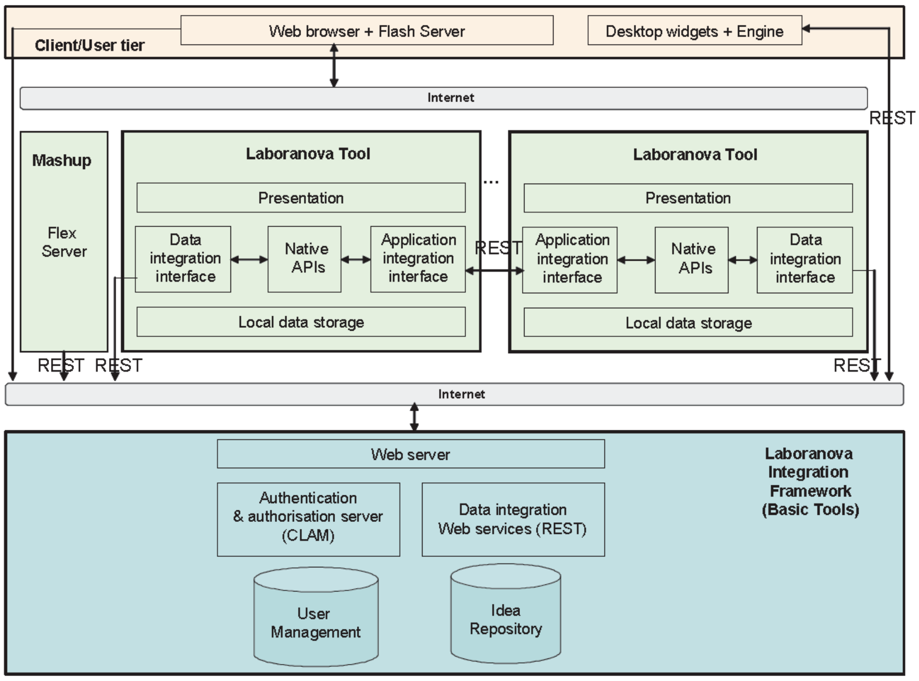
2.3. Data Synchronization

The system provides a specific resource to synchronize any change in the repository. In the same way that any other resource in the system, this resource has also two representations: an HTML representation for documentation purposes and, instead of having an XML representation, this resource provides an RSS representation to allow subscriptions, thus spreading any change in the system dynamically. Specifically, it informs of any POST, PUT or DELETE operation performed on a resource. The structure of this RSS contains at least information with regards to: which resource was changed (the URL of the element added, updated or removed), when this change took place, who did it and which HTTP method was used. This resource is configurable, allowing tools to subscribe for specific filtered synchronizations (e.g.: only changes performed by a specific tool).

2.4. Lost Update Problem

Laboranova Live allows simultaneous access to the idea repository, thus it is possible that two tools try to modify the same resource at the same time. This situation can generate what is known as the *lost update problem*, described as:

1. Tool 1 accesses resource X and starts editing it.
2. Tool 2 also accesses resource X and starts editing it.
3. Tool 1 saves its modifications.
4. Tool 2 also saves its modifications but as tool 2 does not know about the previous modifications, the first update is lost.



Client/User tier
Web browser + Flash Server
Desktop widgets + Engine
Internet
REST
Mashup
Flex Server
Laboranova Tool
Presentation
Data integration interface
Native APIs
Application integration interface
Local data storage
REST
...
Laboranova Tool
Presentation
Application integration interface
Native APIs
Data integration interface
Local data storage
REST
REST
REST
Internet
Web server
Authentication & authorisation server (CLAM)
Data integration Web services (REST)
User Management
Idea Repository
Laboranova Integration Framework (Basic Tools)

Figure 2. Service-oriented architecture applied to Laboranova. Source: Laboranova project

To handle this problem, Laboranova Live implements a solution based on *unreserved checkout with automatic detection and manual resolution* [10][8]. This makes use of *entity tags* (ETags) and *precondition* request-headers [7] to detect updating conflicts and allows parallel access to resources, thus avoiding any tool locking resources for a long time, which could halt the benefits obtained by implementing the real time synchronization explained above. Specifically, the lost update problem is solved as follows. Whenever a tool requests a resource, the response returns its ETag (i.e., the codification of the resource content and the timestamp of its last version). If later the tool modifies that resource, the request will contain an *if-match* request-header field. That precondition field will enclose the ETag received by the server when the resource was firstly retrieved. Finally, one of the following two situations will be dealt with:

1. The ETag of the targeted resource matches the one given in the if-match field and the request is executed as normal.
2. There is no match and the server returns a 412 response code: a conflict has been detected. Tools are free to handle this situation with exception of overwriting the resource (e.g.: warning the user, retrieving the new version and merging the modification, cancelling the request).

2.5. Service versioning

During the lifespan of a resource it is possible that some of its representations change, along with its corresponding Web services. This might be a problem to keep legacy clients running properly. For those representations whose MIME type uses a standard format (e.g.: RSS, JPEG) this is not the case, as their structure is defined and known. In other words, if the change in a representation implies using a standard MIME type or a standard format, there is no problem, as how to manipulate those representations is known. In the case of representations which rely on MIME types with non-standard formats or semantics (XML files, for instance, with non-standard structure) problems may arise. Currently, developers seem to have addressed Web service versioning in two main ways: (1) modifying the URL and (2) versioning the media type. According to Fielding ([5], section 5.2.1.1, p.90), URLs should change as little as possible, therefore the second option seems to be more compliant with REST and is the solution chosen for Laboranova Live.

It uses the MIME types to manage Web service versioning [11][12]. Basically, when a tool uses its own defined representation, it should also define its own MIME type. For example, if the representation consists on an XML file, its MIME type could be *application/vnd.ToolName+xml* ([11], section 3.2), which means that the representation is an XML file and its structure is defined by *ToolName*. Then, if there is a change in this representation and it is necessary to keep the old version for legacy systems, creating a new MIME type is sufficient (e.g.: *application/vnd.ToolName-v2+xml*). Clients will distinguish among versions by means of the *Accept* request-header. Old clients would launch GET requests to the resource URL with application/vnd.ToolName+xml in its *Accept* header, while new ones would use the new MIME type. In this way there are no changes in the resource URL, but clients have to be instructed to use the *Accept* header for content negotiation when using the tool.

2.6. Service Discovery

REST does not use or support any formal service directory or service discovery system, although private solutions to solve this problem are emerging[3]. With respect to service discovery, Laboranova Live follows the following convention. Each tool provides a resource called *services*. This resource is accessible through the root path of the server (e.g.: http://www.LaboranovaLive.com/services/). This resource shows all the services exposed by the tool. At least an XML representation is implemented for this resource as well as an HTML representation, so that a human user can navigate from the resource *services* to any other resource exposed and browse its documentation. It would be more REST-compliant if this resource was accessed through the root path alone (e.g.: http://www.LaboranovaLive.com) but, these being Web applications, this could interfere with their expected behaviour.

2.7. Single Sign-On

Laboranova Live uses a *centralized login and authentication management service* (CLAM)[4] to provide a *single sign-on* authentication service for any tool in the system. Any Laboranova application can request a validation of the user's credentials from the CLAM server. The request contains the username, password and the application name. The CLAM server then checks the user login details against the information contained in the *user database* and, if the user details are confirmed (i.e. the user belongs to one

[3] See [http://www.3scale.com/], last visited on 2009-06-02.

[4] See Crawford, The CLAM System – Overview [http://foss.ulster.ac.uk/file/overview.pdf?file_id=6].

of the groups allowed to access that particular application), it returns the application a *ticket*. This is an XML file containing user information (e.g., internal id, first name, last name) and the user groups he belongs to, in the case of successful authentication, or an empty XML file, in the case of unsuccessful authentication. CLAM's *user database* stores information about the applications which require authentication, individual user accounts, and the user groups to which users subscribed. This gives CLAM the flexibility needed to achieve application specific authentication for each system's user.

3. Conclusions

The integration of heterogeneous applications offers many benefits; as a consequence, the goal is not to try to eliminate the heterogeneity, but to help developers to master it and manage it. There have been many approaches to integrate heterogeneous technologies in the context of the Web. Most of them cannot provide a common solution because they try to solve the problem using an incomplete set of proprietary technologies. To establish a holistic, commonly accepted and standardised approach would be preferable. This paper describes how Laboranova Live, a platform based on REST Web Service, is used to deploy an application set implemented within the Laboranova European project. This platform can be used to integrate any sort of collaborative tools or any set of applications which may need to interoperate among them while keeping autonomous functioning. In this approach the systems and technologies remain heterogeneous, but their interface and collaboration patterns are standardised using lightweight standards such as XML and Internet protocols.

4. References

[1] Stal, M. *Web Services: beyond component-based computing*. Communications of the ACM, Vol. 45, No. 10 (2002-10)

[2] Curbera, F., W.A. Nagy, S. Weerawarana. *Web Services: Why and How*. IBM T.J. Watson Research Center (2001)

[3] Pautasso, C., O. Zimmermann, F. Leymann. *RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision*. Proceeding of the 17th international conference on World Wide Web, Beijing, China (2008) 805-814

[4] International Telecommunication Union. I*TU-T Recommendation X.200. Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model.* [Online] 1994. http://www.itu.int/rec/T-REC-X.200-199407-I/en.

[5] Fielding, R. T. *Architectural styles and the design of network based software architectures*. Doctoral Dissertation, University of California, Irvine, CA, USA (2000)

[6] Fielding, R. T., Taylor, R. N. *Principled Design of the Modern Web Architecture*. ACM Transactions on Internet Technology (TOIT) (New York: Association for Computing Machinery) 2 (2): 115–150, ISSN 1533-5399 (2002-05)

[7] Fielding, R. T., et al. *RCF 2616 - Hypertext Transfer Protocol -- HTTP/1.1*. [Online] June 1999. http://tools.ietf.org/html/rfc2616.

[8] Berners-Lee, T., Fielding, R. and Masinter, L. *RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax*. [Online] January 2005. http://tools.ietf.org/html/rfc3986.

[9] Rodriguez, A. *RESTful Web services: The basics* [Online] 6 November 2008.http://www.ibm.com/developerworks/webservices/library/ws-restful/

[10] Frystyk, H. and LaLiberte, D. *Editing the Web - Detecting the Lost Update Problem Using Unreserved Checkout*. W3C. [Online] 10 May 1999. http://www.w3.org/1999/04/Editing/.

[11] IANA. *MIME Media Types* [Online] http://www.iana.org/assignments/media-types/

[12] Freed, N. *RFC 4288 – Media Type Specifications and Registration Procedures* [Online] http://tools.ietf.org/html/rfc4288