Open Source for e-Government Application Integration: a PHP-based solution

Simone Gammeri
DIST – University of Genova
Genova, Italy
simone.gammeri@gmail.com

Francesco Di Cerbo
DIST – University of Genova
Genova, Italy
francesco.dicerbo@unige.it

Marco Scotto
DIST – University of Genova
Genova, Italy
scotto@dist.unige.it

Giancarlo Succi
Free University of Bozen
Bozen, Italy
giancarlo.succi@unibz.it

Tullio Vernazza
DIST – University of Genova
Genova, Italy
tullio@dist.unige.it

Abstract – This paper introduces UniGeINT - University of Genoa Integration Tool -, a framework designed to provide integration between several existing tools, databases and data repositories we used in internal administrative processes, by exploiting OSS benefits in terms of costs reduction and bug reporting. Moreover, it adopts state-of-art solutions in Enterprise Application Integration, by using specific design patterns able to match existing IT marrow. So we required in final product common features unmissible for a large scale corporative environment: reliability and fault tolerance, besides to a complete integration with anything previously developed.

I. INTRODUCTION

This paper introduces UniGeINT; a PHP-based Integration Framework; it specifically deals with application context, design pattern and architectural features we used or we planned to introduce.

Due to evolution of IT infrastructure, University of Genoa reconsidered its IT solutions to internal administrative process management, not only about students' information, dealing on didactic and taxes, but also about teachers' and courses' management. In short, a huge part of data repositories were built up without a true integrated approach, causing the creation of several “islands of automation” [1][2], with a low cooperation level.

As a result, any modification to existing structure, like a relocation of features or a new acquisition, would require a great effort to be involved in such a scenario, moreover if that operation expected to integrate two different isles. For instance, a new e-learning or group-ware suite, which would be related on user's accounts both for students and teachers, could not be straitly linked to courses details and institutional web portals.

As a natural evolution in IT process, University of Genoa is now experimenting new solutions to find closer and closer products matching its own needs. On the other hand, to move out from test phase towards a production system infrastructure means to start an expensive process, as for any corporate environment: cost evaluations are growing or getting lower depending on internal flexibility and particular technological choices.

Despite this statements, University of Genoa IT management started a research to improve functional mobility inside its information structures and uniformed some past standardization projects - based on XML data exchanging - with other data entities and software which such interface was not available for.

As integration is the aim of our project, we named it “UniGeINT”: that is a SOAP-based middle-ware framework, which exploits web services' powerful extrinsic benefits such as their unrequested and fixed technological implementation, which permits coexistence of heterogeneous solutions.

What we clearly pointed out in our specifications was to put in the resulting product a strong attention on reliability in messages exchanging, and so to oblige SOAP communications to a complete validation for data type and even for delivered payload. In order to carry out this task, we defined an 'ad hoc' XML Schema by which we get messages semantic and commands formalized.

First UniGeINT test consisted in adding a well-known open source e-learning suite, Moodle [3], and mixing up University of Genoa existing e-learning solutions and data repositories together a group-ware suite.

In a few words, this paper introduces Moodle associated web service, an instance of middle-ware software we produced, and UniGeINT core - the "engine" - which is mainly used to allow easy user interaction with web services.

This paper is organized as it follows: section II aims to present peculiar technologies we adopted in developing our framework; section III spends, instead, a few lines about system design and architecture, while section IV is oriented to explain Moodle plugin and its behavior. At last, section V draws up the conclusions we arrived at.

Finally, let us briefly underline our attitude to public release of web services sources dealing with Open Source Software: we took on the same licenses of original product - preferring anyway GPL among others – as means to benefit of code injection and others skilled contributes from Open Source Communities.

II. USED TECHNOLOGIES

A. XML

What we first had to decide about implementing our project was which technologies it should be based on.

Since UniGeINT would cooperate with previous XML integration solutions, and once considered that we aimed towards a web service structure, we chose eXtensible Markup Language as our main information vehicle. [4]

XML describes a class of data objects, the XML documents, and partially describes the behaviour of
computer programs which process them. By construction, XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Markup encodes a description of the document's storage layout and logical structure. [5]

XML consists of a powerful syntax which permitted us to split information content from its own representation, and which grants data exchanging all over the WEB. Moreover, it allows to use standard-valid documents and to impose constraints on storage layout and logical structure by a collateral XML technology called XML Schema. [6]

B. XML Schema

XML Schema definition language aims to define and describe a class of XML documents. Schema components constrain and document the meaning, usage and relationships of their constituent parts: data-types, elements and their content and attributes and their values. Schemas may also provide for the specification of additional document information, such as normalization and defaulting of attribute and element values. [6]

Therefore, when one wanted objects to communicate by XML documents, it would be wise to define at first the Schemas which documents must be concerned to, and then to validate every XML document by referring to its own schema definition. It goes without saying that we adopted this kind of procedure all across our project.

C. WEB SERVICES

Our project is intended to offer a Web Service. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format - specifically WSDL - and other systems interact with it in a manner prescribed by its description – in this case SOAP messages. [7]

D. WSDL

Web Services Description Language (WSDL) provides a model and an XML format for describing web services. Quoting World Wide Web Consortium, “WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information” [8]. WSDL enables to separate description of the abstract functionality offered by a service from its realization (see Fig. 1).

WSDL describes a web service in two fundamental stages: one abstract and one concrete. Within each stage, the description uses a number of constructs to promote reusability of the description itself and separate independent design concerns.

At an abstract level, WSDL describes a Web service in terms of the messages it sends and receives; messages are defined independently, without requiring a specific wireframe and using a type system, typically XML Schema.

At a concrete level, a binding specifies transport and wire format details for one or more interfaces. An endpoint associates a network address with a binding. And finally, a service groups together endpoints that implement a common interface.

E. SOAP

Simple Object Access Protocol (SOAP) provides the definition of XML-based information which can be used to exchange structured and typed information between different peers in a decentralized, distributed environment - particularly in a client-server application or a multi-peer cooperative context.

The two main actors in a SOAP scenario are a sender and a receiver, or, to be more accurate, a SOAP server and a SOAP client.

SOAP is a stateless, one-way message exchange paradigm, applications can create more complex interaction patterns by combining such one-way exchanges with features provided by an underlying protocol or application-specific information.

SOAP definitely provides a framework for a way by which application-specific information may be conveyed in an extensible and encapsulated manner. [9]

Facilities provided by SOAP communications also allow application integration design which is not bounded to any implementing technology: this means that a SOAP peer, for instance, could be developed in a PHP environment, on a J2EE platform and even onto Microsoft .NET, without loosing any detail for transported payload.

F. XSL & XSLT

eXtensible Stylesheet Language (XSL) is a family of recommendations to define XML document transformation and presentation.

eXtensible Stylesheet Language: Transformations (XSLT) is a language that describes rules for transforming a source XML tree into a result tree. The transformation is achieved by an XSLT processor associating patterns with templates: a pattern is matched against elements in the source tree and the template result is returned in a new separated document. [10][11]

In order to associate a viewing or a specific-describing model to an XML document, we chose XSL
Transformations to get every kind of document we needed; note that XML Schemas, Xforms and HTML documents are moreover XML well-structured documents, so XSLT processors are able to match template rules also against them.

III. ARCHITECTURE

Our target was to provide a middle-ware layer to permit cooperation between different data entities: relational or hierarchical databases, management programs and web portals. We did have an XML gateway which was previously developed to integrate some specific application and which was mainly used to access students' administrative tasks and repositories.

Nevertheless, due to such a heterogeneous operating context, we chose to redesign our infrastructure, introducing a new and extensible message layer: we wanted to be able to reuse existent solutions, and delivering just XML requests to receivers.

In order to accomplish our needs, first of all reliability, we decided to use a central gateway for messages, which should have a twofold role: not only as a pure deliverer but also as a common administrative interface, both for functional and debug issues.

As means to realize this part of specifications, the design pattern we picked out was the well-known Message Broker [12] (see Fig. 1). According to our requirements, we modified original pattern structure and gave Broker some higher-level functionalities: it can operate in message layer as well as in user interaction.

Broker activity in SOAP layer is induced by the need to decouple SOAP server and client, so that it acts as a main cross connector, and frees SOAP functionalities from network topology, not to forget that it has the chance to maintain a specialized controller on whole messaging infrastructure.

Moreover, we added some user-level functionalities, in order to let Broker interact with other entities as a generic client: in this way we let users - system administrators - to instruct operations directly by Broker.

Now it is clear where and how we modified original pattern to match our needs, including the catalogue of services we represented by WSDL documents.

Once considered all these requirements, we called this central core "engine".

In order to exploit the pre-existent structure with an XML gateway support, and, at the same time, not to abandon our design philosophy, we also decided to use SOAP, which was already implemented for some isles, as the message-standard, so that we left us the chance for the future addiction of a group-ware suite.

To enable SOAP awareness to applications, we also adopted "message gateway" pattern [13] and obtained an encapsulation of messaging functionalities, without discarding original applications. This pattern instructs to develop SOAP capabilities in a separate software module, as mean to preserve original software integrity. Where it was available, we interacted directly with the exposed API; otherwise, we persecuted the least-intrusive approach, very easy to follow in case of Open Source software, or, where even this was not possible, we directly operated on applications' data repositories.

During this process, we have written two different interfaces, the first is oriented towards system messaging and exposes system's features, while the dual one makes users able to approach framework capabilities and program elaboration methods.

We called this interaction applications "plugin" (see Fig. 3).

From a technical point of view, we implemented a SOAP server for the external plugin interoperation, which would be used by the generic engine-resident SOAP client or from third party. Moreover, in order to reach the highest structure reliability, we had to pay attention to eventual errors on SOAP payload, that is the part of SOAP message which has got a particular semantic built up to support commands and data exchange between SOAP clients and servers. In order to catch up this purpose, we ensured communication safeness in our messaging protocol, through developing an “ad hoc” XML Schema to validate SOAP payload for message exchange.

For an exemplification of a plugin structure, we are now introducing Moodle plugin.
XSLT transformations and XML Schema document validation.

Fig. 3: Moodle plugin diagram

A. Importation

This part of interface is composed by three methods and allows Moodle to make a massive importation of users, students or categories, on a SOAP messages framework.

By succeeding in this, first of all we had to write an XML Schema in which we defined data-types and structures for the XML acquisition document: we found out data fields in Moodle database, in order to represent them via XML tags. We immediately understood that the best way to keep the use of our import-system flexible was to hold necessary data fields separated from unsubstantial ones, so we distinguished them into “description” or “Moodle-oriented” and set up the latter with default values.

Then we entrusted SOAP client-server procedures for all the tasks about getting and de-enveloping messages, while we took care about parsing WSDL document, validating it and finally collecting data for database insertion.

Another important aspect we want to highlight is that our scripts are not intrusive at all against the original environment philosophy: any operation on Moodle database is committed using the functions which were provided in data management library.

What we obtained at last was to get information imported from external source in a formal, rigorous and above all automated way. Import interface allows to:

• import users;
• subscribe users to a course;
• import courses;
• import course-categories topology.

B. Exportation

This second part of interface was built in order to offer a method to export information from Moodle system with an XML valid document.

Even if we did know that Moodle was already able to store backup information of its entire database onto an XML file, we did not find any XML Schema describing its structure. Moreover, although it was possible to select which information one wanted to collect in a backup file, this kind of customization was not useful for our intents: we needed in fact to simply export users, courses and categories with a flexibility of integration towards the external modules or databases, storing data about University of Genoa's IT system.

In order to obtain what we wanted, we primarily decided to adopt the same schema we had created for importation methods; then we wrote some script instructions which evoked the “datalib” functions provided in Moodle administrative tools and we filled up an XML document.

Plugin process goes on as it follows: once data is well formatted, XML message is encapsulated in a SOAP envelope and delivered to the main engine of our application. These operations are accomplished without any possibility of losing information, due to direct validation from XML Schemas as for SOAP message as specifically for its payload.

Therefore it is interesting to note that our application can export towards any other system:

• users of the entire Moodle
• users of a single category or course
• all or single courses
• all or single courses from a specific category.

V. CONCLUSIONS

This work, even if it is still in progress, has reached, as a first result, a concrete benefit for Open Source community, providing a useful extension for universities or formation-operative groups. When concluded, we could achieve an important economical benefit for University of Genoa, in spite of commercial integration solution costs.

VI. REFERENCES

[9] SOAP Version 1.2 Part 0: Primer (W3C Recommend-

Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 204-208
Proceedings of the First International Conference on Open Source Systems
Genova, 11th-15th July 2005
Marco Scotto and Giancarlo Succi (Eds.), pp. 204-208


