The Journal of Systems and Software 86 (2013) 3013–3023

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A model-driven approach to develop high performance web applications

José Luis Herrero Agustin*, Pablo Carmona del Barco

Department of Computer and Telematics Systems Engineering, University of Extremadura, Avda. de Elvas s/n., 06006 Badajoz, Spain

ARTICLE INFO

Article history:
Received 30 March 2012
Received in revised form 27 May 2013
Accepted 10 July 2013
Available online 22 July 2013

Keywords:
Model-driven architecture
Web applications
Rich internet applications

ABSTRACT

The evolution of web technologies in the last few years has contributed to the improvement of web applications, and with the appearance of AJAX and Web 2.0 technology, a new breed of applications for the Internet has emerged: widgets, gadgets and mashups are some examples of this trend. However, as web applications become more and more complex, development costs are increasing in an exponential rate, and we believe that considering a software engineering methodology in the development process of this type of applications, contributes to the solution of this problem. In order to solve this question, this paper proposes a model-driven architecture to support web application development from the design to the implementation model. With this aim, the following tasks have been performed: first a new profile extends UML with new concepts extracted from the web domain, then a new framework supports web application development by composing heterogeneous web elements, and finally, a transformation model generates web applications from the UML extension proposed. The main contribution of this work is a cost and complexity reduction due to the incorporation of a model-driven architecture into the web application development process, but other advantages that can be mentioned are a high performance degree achieved by a prefetching cache mechanism, and a high reusability, since web elements can be reused in different web applications.

© 2013 Elsevier Inc. All rights reserved.

1. Introduction

Traditional web applications are composed by HTML tags and script code, however the type of applications that can be developed is limited and a low degree of interactivity is achieved. With the appearance of Web 2.0 and the introduction of AJAX (Asynchronous JavaScript and XML) (Garrett, 2005), a new breed of web applications with a high degree of interactivity has emerged. RIA (Rich Internet Applications) (Paulson, 2005) has emerged under the umbrella of these new technologies, and has gained much attention and acceptance, since web browsers are the only* execution environment required. For this reason, software vendors are actually adapting their applications to this new trend: Adobe has presented AIR[1] (Adobe Integrated Runtime) to extend web development to the desktop, Google has introduced Google Docs[2] as a new technology to work with online documents, and Microsoft has developed Windows Presentation Foundation[3] to provide support for web browser applications (WBAs). The interest exposed by all of these software vendors demonstrates the attention this technology is gaining.

However, while web applications become more and more complex, development costs have exploded because initial stages of software life cycle are not considered in the development process of this type of applications. Another important implication of this evolution is that the size of web applications have increased exponentially, which provokes a severe increment in the latency degree, since web applications require to be downloaded before they can be used.

In order to solve these problems, MDA (Model Driven Architecture) (OMG, 2003) is proposed in this paper because simplifies modeling, design, implementation, and integration of applications by defining software mainly at the model level. The primary goals of MDA are portability, interoperability, and reusability through architectural separation of concerns (Estefan, 2008), making product development more cost efficient by increasing automation in software development (Teppola et al., 2009).

The objective of this paper is to propose a model-driven architecture to develop high performance web applications that provides a solution to all these problems. This approach has been achieved at three levels: first, a new profile includes web application concepts and extends UML (OMG, 2011) with a new profile, then a new framework supports web application development by

* Corresponding author. Tel.: +34924289300x86868.
E-mail addresses: jherrero@unex.es (J.L. Herrero Agustin), pablo@unex.es (P.C. del Barco).

[1] http://www.adobe.com/es/products/air.html.
[2] http://docs.google.com.
[3] http://msdn.microsoft.com/es-es/library/ms754130.aspx.

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.07.028

composing heterogeneous web components, and finally, a transformation model generates web applications from the proposed UML profile.

This paper is organized as follows: first, Section 2 presents the motivation of this work, an example of a scenario, and the related works. Then, a UML extension is proposed in Section 3, which defines a new profile to design web applications. Next, the framework and performance evaluations are presented in Section 4, while Section 5 describes the transformation model to generate web applications from a UML design. Finally, conclusion and future works are mentioned in Section 6.

2. Motivation and related works

The rapid evolution of web technologies has contributed to the appearance of new types of web entities to support client's requirements. A classification of this type of entities is presented:

- *Widget*: is an interactive single purpose application for displaying and/or updating local data or data on the Web, packaged in order to allow a single download and installation on a user's machine.
- *Gadget*: is a dynamic web content, embedded on a web page, to include new visual elements for a web page. iGoogle and Windows Live are two examples of how gadgets can be aggregated into personal web pages.
- *Mashup*: is a web application composed by contents from different sources, such as those provided by IBM mashup tools or Yahoo Pipes.
- *Web component*: is a new type of web entity, proposed in this paper, which can be activated inside a web application and provides a full description of its interface using IDL (Interface Description Language) (OMG, 2010).
- *Remote service*: is a web entity that supports remote operations.

Currently, web applications are complex software composed by heterogeneous components, and this is the reason why the development process of this type of software is so difficult. In this paper, a web application is considered as a composition of different software components that can be activated at client or server side. A client component is downloaded and activated in the user's web browser, while server components are external resources, implemented in a variety of languages (Java, C, etc.).

Different AJAX frameworks have been proposed with the aim of providing new capacities for web applications. In this regard, Echo3[4] is a platform for building web applications that approaches the capabilities of rich clients, and presents a JavaScript client-side, which is executed directly by the client's web browser, while a Java server-side is stored in a Servlet Container. GWT[5] is a Google proposal for building and optimizing complex web applications, AJAX applications are written in Java and compiled into a highly optimized JavaScript code that runs across all browsers. Dojo Offline[6] is a cross-framework that enables web applications to work offline, where the information generated by the user is stored inside the web browser. In Mesbah and Van Deursen (2008) it is presented a comparative of these approaches, and proposes a new architectural style (SPIAR) which captures the essence of AJAX applications.

Another interesting research studies how to provide connection to remote web services, in this sense (Zdun et al., 2003) defines a framework based on asynchronous invocation of web services. In this work, two proxies (synchronous and asynchronous) support different types of connections at the client side, while the specialization of asynchronous calls is supported by handlers. Asynchronous callbacks in web services have also been proposed by Qian et al. (2006), in this work a web-service-callback proxy is defined. Also, focused in the correlation problem, asynchronous web services interactions have been analyzed in Brambilla et al. (2004).

Trends in web application development are not limited to the specification of new frameworks, and the design level is also considered. A review of the principles of structured programming and the preferred characteristics of web applications is explained by Marquis (2002). This work presents a mapping of how the traditional guidelines may be applied to web applications development. In the field of MDA, there is a consensus about the benefits that this technology offers to web application development: a reduction of sensitivity to the inevitable changes that affect a software system (Atkinson and Kühne, 2002), a reduction of cost and complexity (Mukerji and Miller, 2001), and an increase of abstraction (Booch et al., 2004). An interesting analysis about the existing problems in the field of web engineering and how they can be solved by model-driven development approaches is presented by Gitzel et al. (2007), which identifies the problems encountered in the development process of web applications such as their dependence on the HTTP protocol, compatibility issues due to the heterogeneity of web browsers, and the lack of performance because of the increase in the latency degree.

Different proposals extend web engineering methods for developing web applications; Fraternali et al. (2010) presents a survey of existing web engineering methods to develop this type of applications. The Object-Oriented Hypermedia Design Model (OOHDM) uses abstraction and composition mechanisms in an object-oriented framework to, on one hand, allow a concise description of complex information items, and on the other hand, allow the specification of complex navigation patterns and interface transformations. The UML-Based Web engineering (UWE) (Koch et al., 2008) is a software engineering approach for the Web domain aiming to cover the whole life-cycle of web application development, which supports a UML-based domain, and a model-driven methodology. Taking into consideration system requirements in web engineering, Escalona and Koch (2006) show how concepts can be defined independently of its representation, and proposes a new metamodel that includes the key concepts for specifying requirements in web applications. The RUX-Model (Linaje et al., 2007) is a representational model that offers a method for engineering the adaptation of legacy model-based Web 1.0 applications to Web 2.0 UI expectations. An extension of this model is proposed by Preciado et al. (2008) where a model-driven approach to web application development by combining UWE method for data and business logic modeling, with the RUX-Method for the user interface modeling of RIAs, is defined.

Focused on the presentation layer *Abstract Data Views* (Urbieta et al., 2007) are defined to specify the structure and behavior of user interfaces in web applications, and according to this model (Valverde and Pastor, 2008) propose an interaction model, based on interaction patterns, to define new semantic to deal with the model-driven web application development.

WebML (Ceri et al., 2002) is a conceptual model for specifying the content structure of web applications and the organization and presentation of contents in one or more hypertexts. WebML has been extended in different ways; (Brambilla et al., 2007) proposed an extension to design semantically web applications and (Moreno et al., 2006) defined a UML Profile to model web applications using WebML.

A novel method for designing web applications that employs a UML notation is proposed by Dolog and Stage (2007). In this model, interaction spaces represent identifiable elements of a software system that support a specific part of the user interaction.

[4] http://echo.nextapp.com/site/echo3.
[5] http://code.google.com/webtoolkit.
[6] http://dojotoolkit.org/offline.

Another perspective for developing web applications is the data-driven model, where the state of the applications resides in a database, and users interact with this persistent state through web clients. According to this model (Reischuk et al., 2012) a framework for personalizing web applications is proposed, and suggested that users have the capability of customizing web applications to fit her requirements.

However, all the new entities (widget, gadget and mashup) that have been consolidated in the web domain have not been considered in these works. Another important issue is that latency and performance evaluations have not been taken into consideration.

3. Web application modeling

The development process presented in this work allows developers to model web applications by composing heterogeneous components. With this aim, an extension of UML is proposed in this section, and a new profile (Fig. 1) has been developed to specify concepts of the web application domain, and new stereotypes and tagged values have been defined in order to facilitate the modeling process of this type of applications.

The main element of this profile is the «*Web Resource*» stereotype that represents a general element in the web, and a software component is represented by the «*URL Resource*» stereotype. Software applications for the web are represented by the «*Web application*» stereotype, and according to previous considerations, they can be composed by other components, represented by the «*Web Element*» stereotype. Web applications, and also web elements, can be bound to web services in order to invoke remote operations. Gadgets, widgets, mashups and web components are represented by stereotypes with the same name. According to the work proposed by Liu et al. (2011), mashups are classified according to data integration

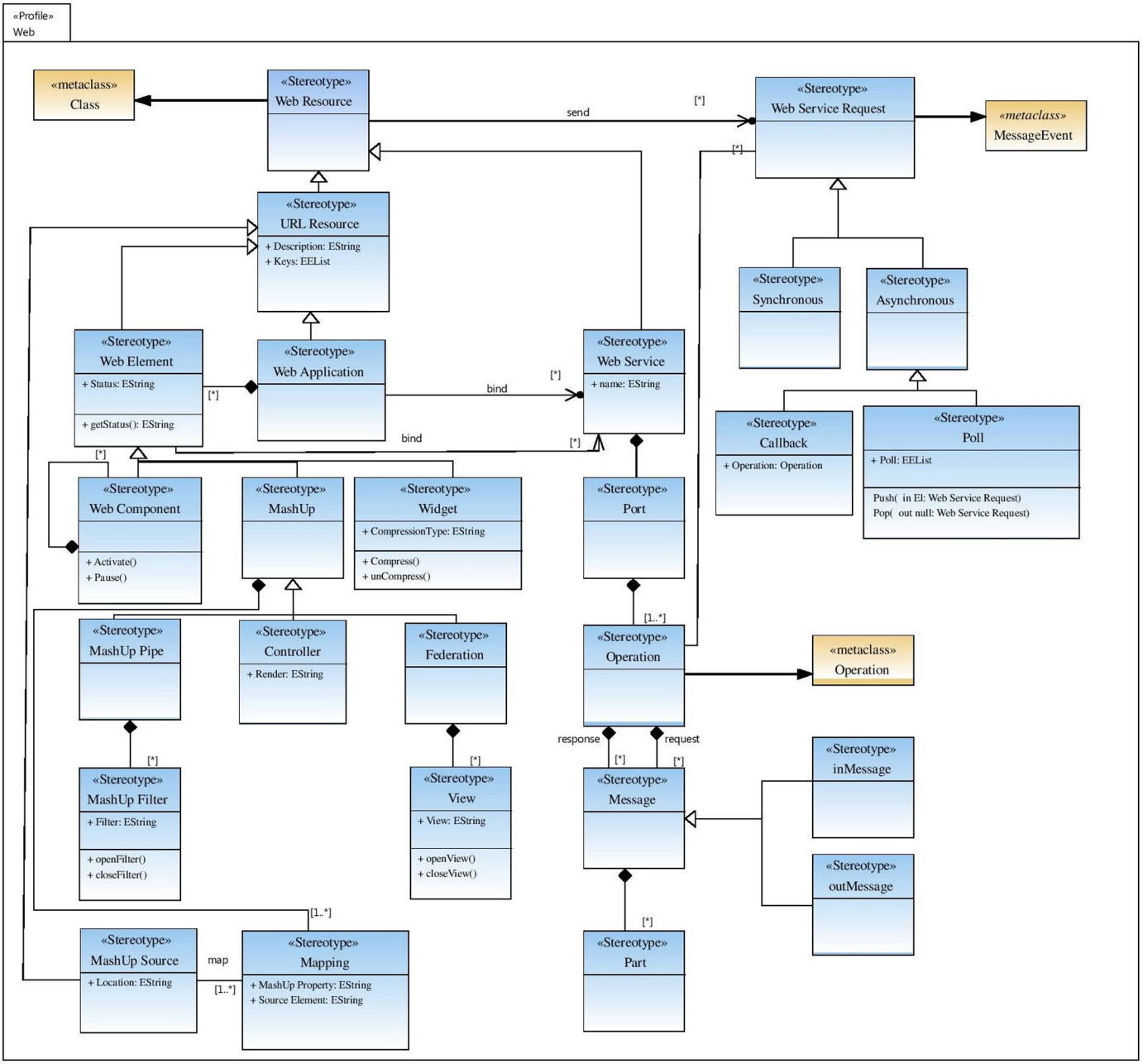

Fig. 1. UML web application profile.

patterns as follows: the «*Mashup Pipe*» stereotype defines a set of pipes or filters that must be applied to the information in order to obtain the final output, the «*Mashup DataFederation*» stereotype specifies different views of the same data, while the «*Mashup Controller*» stereotype describes how the data is rendered. Web sources involved in a mashup, are represented by the «*Mashup Source*» stereotype, and the mapping between a mashup and its sources is specified by the «*Mapping*» stereotype. Different operations (filter, render or view) can be applied to a mashup according to its type.

Remote services are referenced by the «*Web Service*» stereotype and all the elements required to model a web service are also

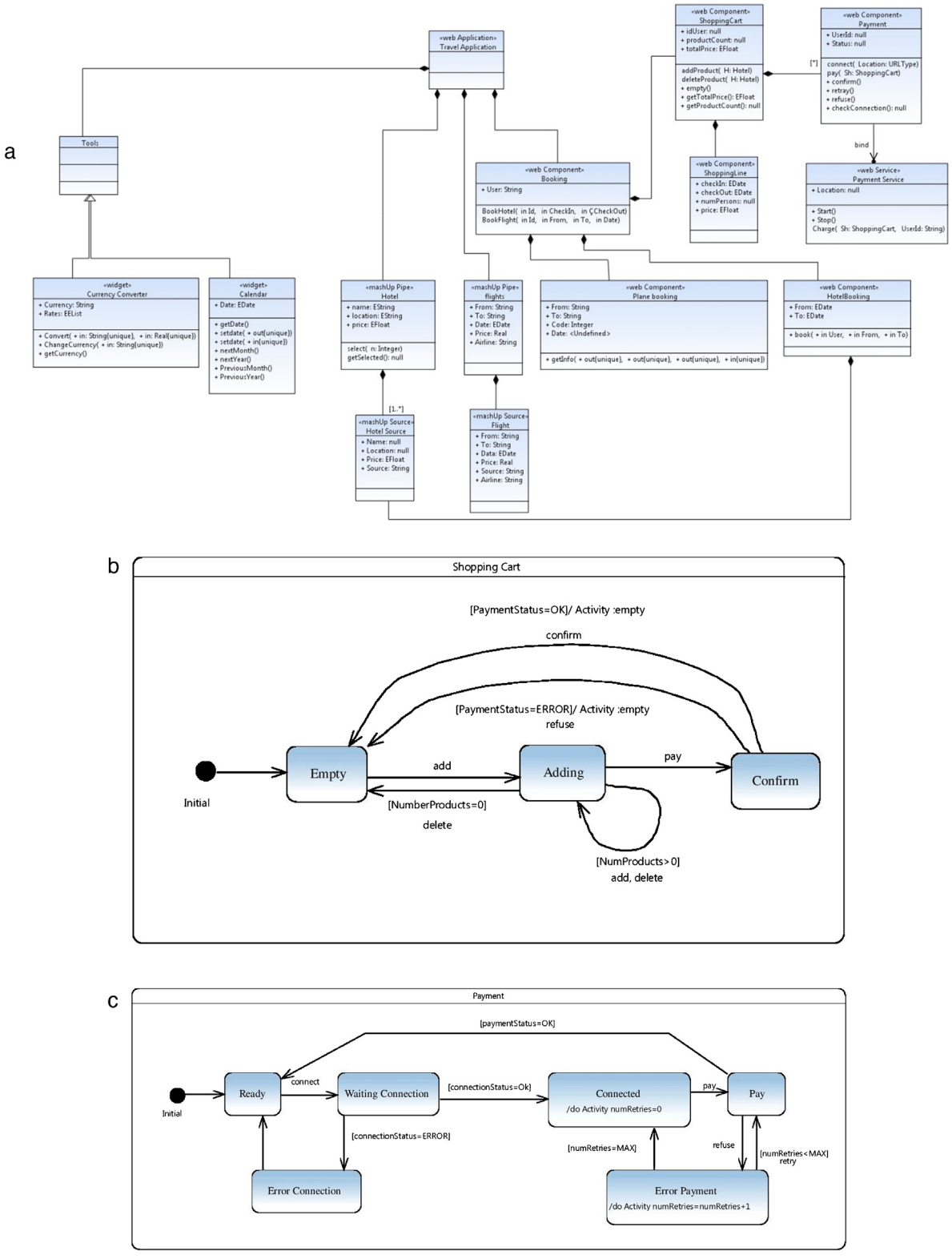

Fig. 2. (A) Holiday web portal design. (B) Shopping cart state chart. (C) Payment state chart.

defined according to the work proposed by Skogan et al. (2004). Requests sent by web resources to web services are represented by the «*Web Service Request*» stereotype, which can be classified according to the following communication patterns: synchronous («*Synchronous web service request*»), and asynchronous («*Asynchronous web service request*»). The asynchronous communication is extended with the «*Asynchronous call back request*» stereotype, which includes the operation executed when the callback is activated. The «*Asynchronous pull request*» stereotype incorporates the *pool* property to hold responses, and the *push* and *pop* method to store and retrieve elements from the pool.

3.1. A demonstration scenario

In order to test the profile proposed in this paper, the following scenario is suggested: a holiday web portal offers hotel vacancies in different locations around the world, and users can select the best alternative according to their preferences (price, date, location. . .). The web portal also offers flight information in order to help users when planning their travels. We consider that the web portal must also provide a shopping cart, payment services, and several tools to facilitate the selection of a specific date and currency conversions. Fig. 2a–c shows the model of this example according to the proposed UML profile.

As several web sources are required to support this scenario, hotel and flight information (*Hotel* and *Flight* sources) must be combined in order to present data in homogenous structures; this is the reason why mashup Pipes (*Hotel* and *Fights*) have been defined. Another important component of this application is the *Booking web* component that represents a travel booking, considering hotel bookings (*Hotel Booking* web component) and flights booking (*Plane Booking* web component). A shopping cart web component (*ShoppingCart*) is attached to a booking component in order to specify the payment information and support connections to bank payment services (*Payment Service*). Finally, different tools have been designed to allow users the selection of a specific date (calendar) and currency conversions (currency converter).

4. The framework

Web applications considered in this paper are characterized by the following properties:

- *Downloaded*: they are downloaded from the web and activated in a web browser environment.
- *Heterogeneous*: web applications are composed by different types of web entities.
- *Distributed*: web components and web services are distributed in the web.

In order to provide all this features, a new framework (Fig. 3) to support the development of web applications by composing heterogeneous resources has been defined (Herrero et al., 2011). In this proposal, web applications are built in the client's web browser, while web components are downloaded and activated only when they are required.

The framework combines three different elements: web applications, web resources and web services.

4.1. Web applications

Web applications require several software components to be downloaded and activated in the client's web browser before they can be executed. In order to guarantee this requirement, the following activation mechanism has been specified:

- The first step involves the invocation of a web application by the client. This task is performed by introducing the URL in the client's web browser.
- When the request is received by the server, the following elements are returned:
  - The components of the framework (remote connections, resource acquisition and AJAX engine).
  - The web application source code.

When all these components have been returned, they are activated inside the client's web browser by the addition of new nodes in the DOM tree of the web application. Finally, the source code of the application is executed.

4.2. Web resources

Gadgets, widgets and mashups, are resources spread over the Internet that can be dynamically downloaded and activated in a web application. This process is supported by the Resource Acquisition Component, which performs the following tasks:

- When a web application requires an external URL resource, a message is sent to the Resource Acquisition Component which builds a request and passes it to the *Ajax Engine*.

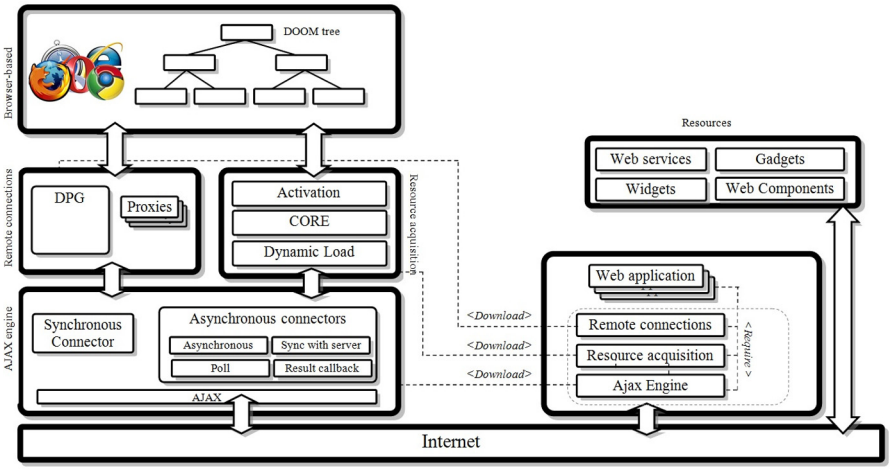

Fig. 3. The framework.

a

```
<WEB COMPONENT>
  <NAME>ShoppingCart</NAME>
  <TYPE> Local </TYPE>
  <DESCRIPTION> ShoppingCart description</DESCRIPTION>
  <AUTHOR>ShoppingCart Author</AUTHOR>
  <REQUREMENTS>
    <REQUIRES> ShoppingLine </REQUIRES>
    <REQUIRES> Payment </REQUIRES>
  </REQUIREMENTS>
</WEBCOMPONENT>
```

b

```
Interface ShoppingCart
{
  integer idUser;
  integer productCount;
  float totalPrice;

  addProduct(Hotel H);
  deleteProduct(Hotel H);
  Boolean empty();
  float getTotalPrice();
  integer getProductCount();
}
```

Fig. 4. (A) XML shopping cart definition. (B) IDL shopping cart interface.

- The AJAX Engine transforms the request into several asynchronous calls according to the callback pattern.
- After the responses have arrived, the Dynamic Load Component parses the answers and builds a local object.
- Requirements are evaluated by the Core Component, and only if all of them are satisfied, the web element can be activated. In other case, the activation is delayed.
- The Core Component activates the local object inside the web application by adding new nodes to the DOM tree.

One of the main benefits extracted from this process is that web elements are independently deployed and they can be reused in different web applications.

4.2.1. Web components

Web components are proposed in this paper as a new type of web resource in the sense that they are downloaded and activated in a web application, but also an interface is specified in order to provide a successful interaction mechanism with web applications. The specification of a web component is divided into the following elements:

- *Definition*: the name of the web component, according to a namespace, a full description, and the set of elements required to activate the web component are specified in this section. XML is the language selected to specify this component, and the following tags have been defined:
  - <WEBCOMPONENT>: represents the beginning of a web component
  - <NAME>: is the name of the web component
  - <TYPE>: defines the type of the web component
  - <DESCRIPTION>: includes the description of the web component
  - <AUTHOR>: contains the web component developer's name
  - <REQUIREMENTS>: represents the beginning of the requirement structure
  - <REQUIRES>: identifies a specific requirement.
- *Interface*: this element specifies the set of operations provided by the web component using IDL language.
- *Functionality*: the code of all the operations is stored in this element.

According to the example proposed in this paper, the shopping cart definition (Fig. 4a) and interface (Fig. 4b) are shown.

4.3. Web services

Complex tasks may require special computers to be performed, or may demand too much time to be downloaded (latency), and for these reasons web elements are not the best approach in these situations. The solution comes from computer networking, where resources are distributed across multiple computers, providing a highly load-balanced application. According to this trend, complex tasks are hosted in remote computers and can be requested through a communication mechanism.

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 in a manner prescribed by its description using SOAP messages. Web services is the model selected to connect web applications for two reasons: (i) offers cross-platform, cross-programming and internet-firewall-friendly and (ii) can be invoked from web applications using the HTTP protocol and SOAP messages, which simplifies the connection mechanism.

One of the main problems when developing distributed applications is the complexity that communications protocols introduce. In order to solve this problem, the proxy pattern (Gamma et al., 1994) provides a surrogate or placeholder for another object to control the access to it and facilitates the development process of distributed applications. The remote proxy pattern is derived from the proxy pattern, and it is used to hide the communication mechanisms, offering a transparent connection between clients and remote resources.

4.3.1. Service binding

The bidding process proposed in this paper (Fig. 5), asks a service registry for a specific service, and generates a proxy object from the WDSL specification returned. This proxy is responsible for marshaling the invocations and sends them to the correct service provider.

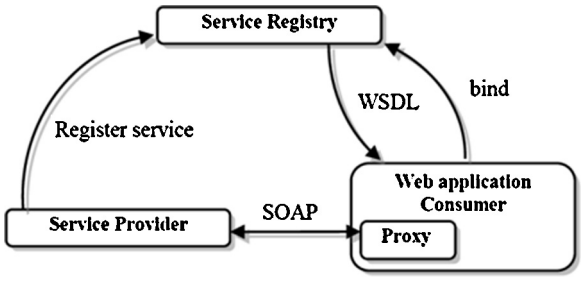

Fig. 5. Web service binding process.

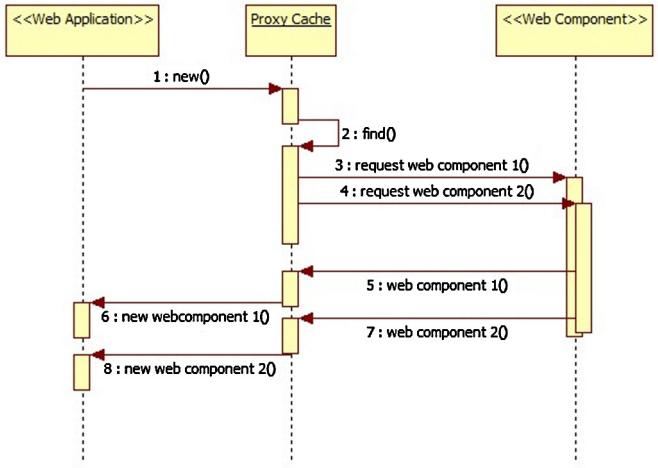

Fig. 6. Cache prefetching process.

The following code presents the binding process of a web service (*service_name*), the generation of the proxy (*proxy_1*), and the invocation of a remote operation (*operation_A*).

```
bind(service_registry, port, service_name);
proxy_1 = new service_name();
proxy_1.operation_A();
```

In this case, the binding process generates a new dynamical class (*service_name*) which provides accessibility to the operations published by the service. Then, an object (*proxy_1*) is instantiated from this generated class, and finally an operation (*operation_A*) is invoked.

4.4. Increasing the performance degree

The main problem of this proposal is that the performance degree is severely affected by latency, since web components are required to be previously downloaded before a web application is activated in the user's web browser. In order to reduce this problem, and increase the performance degree of web applications, it is proposed a proxy cache mechanism that intercepts requests and performs the following tasks:

1. Stores a list of downloaded web elements.
2. Checks if a requested web element has been downloaded previously, avoiding duplicated requests.
3. Applies a prefetching algorithm that identifies the web elements to be downloaded before they are requested.

Data prefetching has widely been used in memory cache strategies, where it has been used to anticipate cache misses and fetch data from the memory system before the processor needs the data (Tien-Fu, 1997; Kai-Feng et al., 2005). In this section a software prefetching technique is adopted to minimize latency in web applications. However, the main problem of this technique is to predict the web elements that will be requested, and to download them without interrupting the normal workflow of the web application.

In order to identify the web elements that will be downloaded in advance, the cache proxy performs a searching process and identifies specific patterns in the source code of the web application that correspond with the requesting or manipulation of web elements. The result of this process is a list of predicted web elements that will be downloaded in a background process.

According to the process shown in Fig. 6, web resources are not downloaded when they are required, which can interrupt web application workflow affecting the latency degree. Instead, the prefetching mechanism predicts the set of web elements that will be required and downloads them when the web application is activated, minimizing latency.

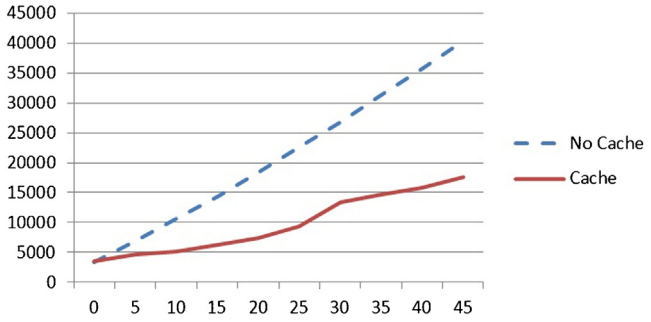

Fig. 7. Test results.

4.5. Design of experiments

In order to demonstrate the increase in the performance degree achieved by the cache mechanism proposed in this paper, and according to the framework proposed, we have generate 50 web applications of different sizes (5–500 KB), and 45 web components (from 1 KB to 200 KB). Each web component has been coded in JavaScript language and a new tool has been developed to build web applications by adding different web components in a random distribution.

A simple test consists of incorporating a specific number of web elements (0–45), randomly distributed in the source code of a web application. Then, the web application is executed activating and inhibiting the cache mechanism, and the results are stored and compared. Fig. 7 presents the results obtained from these tests, where *X* axis indicates the number of resources added to the web applications and *Y* axis represents the latency (ms) obtained by calculating the latency average of all the applications with the same number of web components. As it can be seen, in the case that the proxy cache is not activated, latency increases in a linear rate, otherwise the latency increment is lower.

These experiments identified a few cases where the latency degree behaves worse when the cache proxy was activated. This situation appears when the number of required resources is very low. This is an obvious implication because a new proxy has been added, and a searching pattern process is performed, and both facts incorporate an initial latency that is only compensated when the number of required resources is not very low. But even in

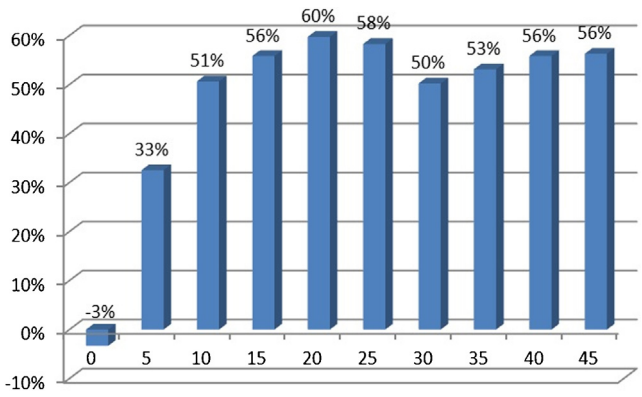

Fig. 8. Decrement in the latency degree.

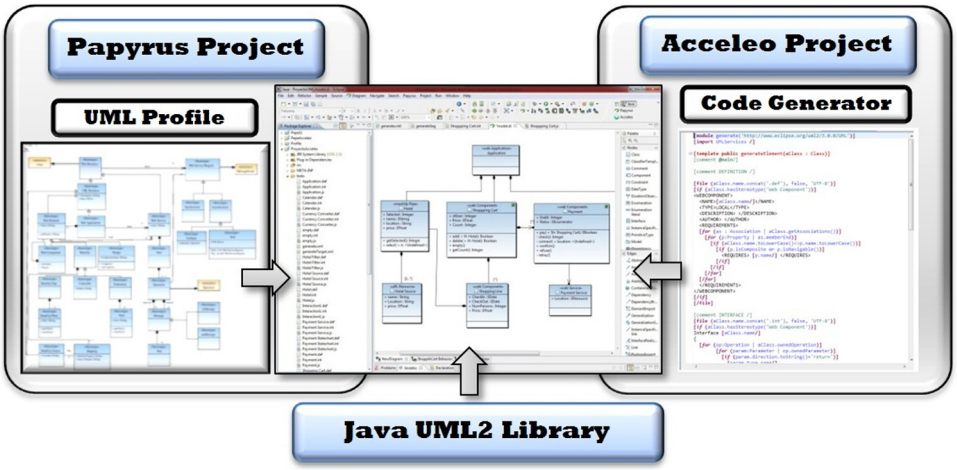

Fig. 9. Transformation tool.

the worst situation, the difference in the latency degree is very low (3%).

According to the obtained results shown in Fig. 8, we can conclude that the average in latency degree has been reduced (47%) applying the proposed proxy cache mechanism.

5. Model transformations

One of the keys of MDA is the capacity of defining transformations from higher-level models to platform specific models guided by a set of transformation rules. With this aim, a generation tool (Fig. 9) is proposed.

This tool is based on Eclipse platform, and different plugins have been used in order to support UML development (Papyrus Project) and code generation (Acceleo Project). A library has been also developed with the aim of requesting UML designs and extracting information about the model.

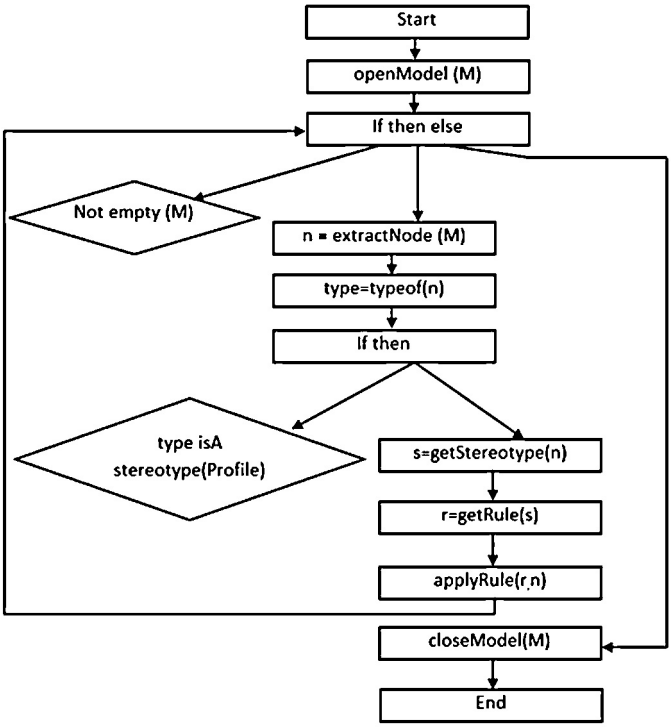

Fig. 10. Transformation algorithm.

5.1. The transformation algorithm

The algorithm proposed in Fig. 10 shows how the transformation algorithm works. First of all, a UML model (M) is opened, next the structure of the UML design is crawled in order to identify each element (n), then these elements are extracted from the model, parsed and identified. In the case of n is tagged with a stereotype (s) defined in the proposed profile, the transformation rule (r) is selected and applied.

The process to transform a UML design, according to the extension proposed, into a web application performs the following tasks:

- The web application stereotype is transformed into the components of the framework (Remote Connections, Object Acquisition and Ajax Engine).
- The web service stereotype activates a binding process that compares the WDSL specification of the web service with the UML design.
- An software component will be generated in the following cases:
  - Gadget or Widget stereotype included in the model.
  - A Mashup stereotype indicates the addition of new object to retrieve information from the web sources. This component also includes filtering, viewing or rendering functions to process the information retrieved.
  - The web component is also transformed into the following elements:
    - *Definition*: the information required to generate this element is retrieve directly from the designed element and,

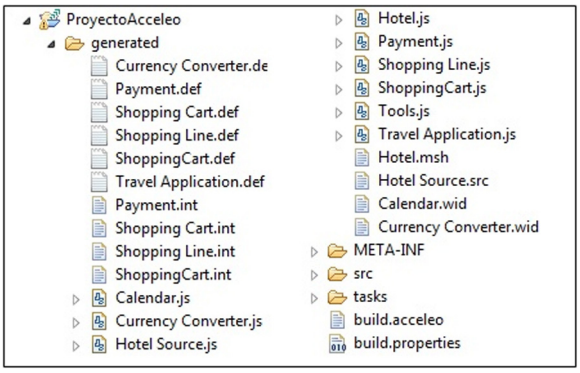

Fig. 11. Generated files extracted from the holiday example.

requirements are obtained automatically following the modeled associations.

- *Interface*: elements that conforms the interface are transformed in IDL.
- *Functionality*: a class is generated, and attributes and methods are identified and transformed.

Following the example proposed in this paper, Fig. 11 presents some of the elements generated by this algorithm.

5.2. The transformation rules

In this section some of rules to transform web components into final code are presented, which are divided in three groups: definition, interface and functional code.

5.2.1. Web component definition rule

This rule generates the definition section of a web component and composes a header with the name and type of the component, the author and description. The other part of this algorithm identifies web component requirements by crawling the UML structure and identifying web component associations. Every association or aggregation attached to a web component or a web service, is transformed into a requirement.

```
Input: a UML model (M)
Output: the definition of a web component.
1  foreach Cm ϵ Elements(M) do
2   if typeof(Cm)=WebComponent
3    openfile(Cm+".def")
4    generate(Header(Cm))
5    foreach As ϵ Associations(Cm) do
6     foreach End ϵ AssociationEnds(As) do
7      if (End<>Cm)
8       if (typeof(As)=Aggregation or typeof(AS)=Association)
9        generate("<REQUIRES>"+End+"</REQUIRES>")
10      endif
11     endif
12    endfor
13   endfor
14   closefile()
15  endif
16 endfor
```

Where *Cm* is an element from the model, *As* is an Association and *End* is a component of an Association.

5.2.2. Web component interface rule

This rule generates the interface according to the IDL language, and with this aim, all methods (*Op*) and parameters (*Param*) of a web component (*Cm*), are identified and transformed into the IDL language.

```
Input: a UML model (M)
Output: the interface of a web component.
1  foreach Cm ϵ Elements(M) do
2   if typeof(Cm)=WebComponent
3    openfile(Cm+".int")
4    generate("Interface")
5    generate(Cm)
6    generate ("{")
7    foreach Op ϵ Operations(Cm) do
8     foreach Param ϵ Params(Op) do
9      if (typeof(Param)=return)
10      generate(typeof(Param))
11     endif
12    endfor
13    generate(Op)
14    generate("(")
15    foreach Param ϵ Params(Op) do
16     if (typeof(Param)<>return)
17      generate(typeof(Param))
18      generate(Param)
19     endif
20    endfor
21    generate(")")
22   endfor
23   generate("}")
24   closefile()
25  endif
26 endfor
```

5.2.3. Web component functionality rule

This last rule generates a skeleton of a web component. In this fragment, a web component *Cm* is transformed into a JavaScript class, and all its methods and attributes (*At*) are identified and transform into JavaScript code. A crawling process is also activated in order to identify aggregations, since this type of associations are transform into arrays.

```
Input: a UML model (M).
Output: the functional code of a web component.
1  foreach Cm ϵ Elements(M) do
2   if typeof(Cm)=WebComponent
3    openfile(Cm+".js")
4    generate("function")
5    generate(Cm)
6    generate("{")
7    foreach At ϵ Attribute(Cm) do
8     generate("this.")
9     generate(At)
10   endfor
11   foreach As ϵ Associations(Wc) do
12    foreach End ϵ AssociationEnds(As) do
13     if (typeof(As)=Aggregation)
14      if (Ocurrences(End,As)>1)
15       generate("this.")
16       generate(End)
17       generate(i)
18       generate("= new Array()")
19      else
20       generate("this.")
21       generate(End)
22       generate(i)
23       generate("=new ")
24       generate(End)
25       generate("()")
26      endif
27     endif
28    endfor
29   endfor
30   generate("}")
31   for (each Operation(Wc) Op)
32    generate (Op)
33    generate("=function(")
34    for (each Param(Op) Param)
35     if (typeof(Param)<>return)
36      generate(Param)
37     endif
38    endfor
39    generate("{ }")
40   endfor
41   closefile()
42  endif
43 endfor
```

6. Conclusions and future works

In this paper a model-driven architecture for developing web applications is proposed and, with this aim, different stages of the software development process are considered. At the design level, a new profile extends UML with new concepts from the web application domain. Then a new framework supports web application development by composing heterogeneous web elements. Finally, a transformation model generates web applications from the UML extension. The main benefits provided by this proposal are:

- Cost and complexity reduction are achieved since a model-driven approach is proposed for developing web applications. Productivity is raised because the generation tool takes care of the transformation process, relieving software engineers of the burden of performing this tedious work, and this implies a very large cost-effective savings.
- As has been stated, the latency degree has been minimized with the introduction of a cache prefetching mechanism.
- As web resources are deployed independently, the can be reused in different web applications, and as such, the reusability degree is increased. Consider now another web application that offers on-line sports equipment products, the *Shopping Cart*, *Shopping Line*, *Payment* web components, and also the *Payment service* can be reused in this domain, since they do not depend on the domain they are activated.
- Web components also provide a number of benefits because they simplify application development and maintenance, and thus, they allow systems to be more adaptive and to respond rapidly to changing requirements (Garcia-Castro et al., 2008). Consider now that the holiday example requires offering new payment methods such as PyaPal, in this scenario, only the *Payment* web component must be updated in order to connect with a PayPal service.

A prototype of a holiday web portal has been presented along this paper, the scenario of the example has been introduced, the design, according to the UML extension, has been modeled, and finally an example of a generated web component has been exposed.

Future works will try to incorporate new types of web entities such as applets or dynamic web contents generated by server languages into the development process of web applications. Moreover, historical information will be included in the prefetching algorithm in order to predict more efficiently the web elements required to be downloaded in advance. Although the evaluation of the quality is out of the scope of this paper, we consider an important aspect of our approach, so another future research will try to apply the International Standards for the Evaluation of Software Quality ISO/IEC 9126 and the Software Product Quality Requirements and Evaluation ISO/IEC 25000, to evaluate functionality, efficiency, compatibility usability, reliability, security, efficiency, maintainability and portability quality characteristics of our approach.

References

Atkinson, C., Kühne, T., 2002. The role of metamodeling in MDA. In: International Workshop in Software Model Engineering.

Booch, G., et al., 2004. An MDA manifesto, business process trends. MDA Journal, http://www.bptrends.com/publicationfiles/05-04COLIBMManifest20-Frankel-3.pdf

Brambilla, M., et al., 2007. Model-driven design and development of semantic Web service applications. ACM Transactions on Internet Technology (TOIT) 8 (1), http://dx.doi.org/10.1145/1294148.1294151, Article 3 (November 2007).

Brambilla, M., et al., 2004. Managing asynchronous Web services interactions. In: ICWS'04 Proceedings of the IEEE International Conference on Web Services, p. 80, http://dx.doi.org/10.1109/ICWS.2004.73.

Ceri, S., et al., 2002. Designing Data-Intensive Web Applications, first ed. Morgan Kaufmann Publishers Inc., ISBN 978-1-55860-843-6, Copyright © 2003 Elsevier Inc. All rights reserved.

Dolog, P., Stage, J., 2007. Designing interaction spaces for rich internet applications with UML. In: 7th International Conference on Web Engineering, pp. 358–363.

Escalona, M.J., Koch, N., 2006. Metamodelling the requirements of web systems. In: Proc. Second Int'l Conf. Web Information Systems and Technologies, pp. 310–317.

Estefan, J., 2008. Survey of Model-Based Systems Engineering (MBSE) methodologies, INCOSE-TD-2007-003-01.

Fraternali, P., et al., 2010. Engineering rich internet applications with a model-driven approach. Journal ACM Transactions on the Web (TWEB) 4 (2), http://dx.doi.org/10.1145/1734200.1734204.

Gamma, E., et al., 1994. Design Pattern: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Garcia-Castro, R., et al., 2008. Towards a Component-Based Framework for Developing Semantic Web Applications. The Semantic Web Lecture Notes in Computer Science 5367, 197–211.

Garrett, J., 2005. Ajax: A New Approach to Web Applications, http://www.adaptivepath.com/publications/essays/archives/000385.php

Gitzel, R., et al., 2007. Using established Web Engineering knowledge in model-driven approaches. Science of Computer Programming 66 (2), 105–124, http://dx.doi.org/10.1016/j.scico.2006.09.001.

Herrero, J.L., et al., 2011. Web services and web components. In: Seventh International Conference on Next Generation Web Services Practices, pp. 164–170.

Kai-Feng, W., et al., 2005. Path-based next N trace prefetch in trace processors. Microprocessors and Microsystems 29 (6), 273–288.

Koch, N., et al., 2008. UML-based web engineering. web engineering: modelling and implementing web applications. Human–Computer Interaction Series 2008, 157–191.

Linaje, M., et al., 2007. Engineering rich internet application user interfaces over legacy web models. Journal IEEE Internet Computing archive 11 (6), 53-59, http://dx.doi.org/10.1109/MIC.2007.123.

Liu, Y., et al., 2011. Composing enterprise mashup components and services using architecture integration patterns. Journal of Systems and Software archive 84 (9), 1436–1446, http://dx.doi.org/10.1016/j.jss.2011.01.030.

Marquis, G., 2002. Application of traditional system design techniques to web site design. Information and Software Technology 44 (9), 507–512, http://dx.doi.org/10.1016/S0950-5849(02)00050-2.

Mesbah, A., Van Deursen, A., 2008. A component-and push-based architectural style for AJAX applications. Journal of Systems and Software 81 (12), 2194–2209, http://dx.doi.org/10.1016/j.jss.2008.04.005.

Moreno, N., et al., 2006. A UML 2.0 profile for WebML modeling. In: ICWE'06 Workshop proceedings of the sixth international conference on Web engineering, http://dx.doi.org/10.1145/1149993.1149998.

Mukerji, J., Miller, J., 2001. MDA Guide version 1.0.1, OMG document number: omg/2003-06-01, http://www.omg.org/cgi-bin/doc?omg/2003-06-01

OMG, 2003. Model Driven Architecture, http://www.omg.org/mda/

OMG, 2010. Interface Definition Language, http://www.omg.org/gettingstarted/omg_idl.htm (accessed November 2012).

OMG, 2011. The Unified Modeling Language v2.4.1, http://www.omg.org/spec/UML/2.4.1/

Paulson, L.D., 2005. Building rich web applications with Ajax. Computer, IEEE 38 (10), 14–17.

Preciado, J.C., et al., 2008. Designing rich internet applications combining UWE and RUX-method. In: Eighth International Conference on Web Engineering, pp. 148–154, http://dx.doi.org/10.1109/ICWE.2008.26.

Qian, K., et al., 2006. Asynchronous Callback in Web Services. In: Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp. 375–380, http://dx.doi.org/10.1109/SNPD-SAWN.2006.23.

Reischuk, R., et al., 2012. SAFE extensibility for data-driven web applications. In: 21st international conference on World Wide Web, pp. 799–808, http://dx.doi.org/10.1145/2187836.2187944.

Skogan, D., et al., 2004. Web service composition in UML. In: Eighth IEEE International of the Enterprise Distributed Object Computing Conference, pp. 47–57, http://dx.doi.org/10.1109/EDOC.2004.30.

Teppola, S., et al., 2009. Challenges in deployment of model driven development. In: Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on Software Engineering Advances, 15–20, 20–25, doi:10.1109/ICSEA.2009.11.

Tien-Fu, C., 1997. Reducing memory penalty by a programmable prefetch engine for on-chip caches. Microprocessors and Microsystems 21 (2), 121–130.

Urbieta, M., et al., 2007. Designing the interface of rich internet applications. In: Fifth Latin American Web Congress, pp. 144–153, http://dx.doi.org/10.1109/LA-Web.2007.14.

Valverde, F., Pastor, O., 2008. Applying interaction patterns: towards a model-driven approach for rich internet applications development. In: 7th International Workshop on Web-Oriented Software Technologies, pp. 13–18.

Zdun, U., et al., 2003. Design and implementation of an asynchronous invocation framework for web services. In: Web Services – ICWS-Europe 2003, Volume 2853, pp. 11–35.

José Luis Herrero Agustin received the M.Sc. and Ph.D. degrees in computer science from the University of Granada, in 1995 and 2003. He is currently a member of the Department of Computer and Telematics Systems Engineering at the University of Extremadura (Spain). His current research involves web applications and model-driven architecture.

Pablo Carmona del Barco received the M.Sc. and Ph.D. degrees in computer science from the University of Granada, Spain, in 1994 and 2003, respectively. He is currently with the Department of Computer and Telematics Systems Engineering at the University of Extremadura (Spain) and his research interest include linguistic fuzzy modeling and other soft-computing techniques.