Chapter XXXI

A Tool for Model-driven Design of Rich Internet Applications based on AJAX

Marco Brambilla
Politecnico di Milano, Italy
Piero Fraternali
Politecnico di Milano, Italy
Emanuele Molteni
Web Models S.r.l., Italy

ABSTRACT

This Chapter describes how the design tool WebRatio (and its companion conceptual model WebML) have been extended to support the new requirements imposed by Rich Internet Applications (RIAs), that are recognized to be one of the main innovations that lead to the Web 2.0 revolution. Complex interactions such as drag and drop, dynamic resizing of visual components, graphical editing of objects, partial page refresh are addressed by the RIA extensions of WebRatio. The chapter discusses what kinds of modelling primitives are required for specifying such patterns and how these primitives can be integrated in a CASE tool. Finally, a real industrial case is presented in which the novel RIA features are successfully applied.

INTRODUCTION

The advent of Rich Internet Applications (RIA, for short) has allowed a much broader set of user interaction possibilities within Web applications. Complex interactions such as drag and drop, dynamic resizing of visual components and graphical editing of objects were once a prerogative of desktop applications, while now are available as standard patterns in many Web applications too. These patterns enable more flexible and usable interfaces, but at the same time require a more complicate application logics, both at client side and server side. Correspondingly, if model-driven design is adopted, new primitives and design patterns must be devised.

This chapter aims at discussing what kinds of modelling primitives are required for specifying Rich Internet Applications and discusses how these primitives can be integrated in a CASE tool. In addition, a real industrial case is presented in which the novel RIA features are successfully applied.

The viewpoint presented here is somehow opposite to the typical academic research paper, where an abstract solution to the investigated problem is first designed and verified formally, and then applied top-down to a prototype implementation. In this chapter we report on a bottom-up approach, which has extended a real world modelling notation and tool progressively, following the penetration of RIA features in the market and the raise of interest in the customers.

The chapter deals with four main aspects related to the coverage of RIA requirements in Web application design:

- extensions to the conceptual model;
- extensions to the CASE tool elements and properties;
- architectural issues and code generation aspects;
- implementation examples in real industrial scenarios.

The conceptual modeling primitives cover the following aspects of RIAs: management of new hypertextual link behaviour, including partial page refresh, in-page popup windows, splash screens, dynamic tooltips, and animations; interaction among page objects through drag and drop and dynamic dependencies; and advanced form specifications, including text autocompletion, on-event actions, and field dependencies.

Besides the modeling aspects, the chapter will describe how they are implemented within the WebRatio tool and how they are exploited through automatic code generation. The architectural description of the adopted design framework is provided, together with the analysis of the best mix of technologies that can be leveraged for implementing this kind of features. The designed architectural framework extensively exploits the XMLhttpRequest method and consists of implementing each conceptual hypertext page with two dynamic pages that interact for providing the rich interface features: the first is a back-end dynamic XML page that stores the data of interest for a specific navigation context; the second is the front-end JSP page (including the required JavaScript needed for event management) that is shown to the user. The latter invokes extraction of data fragments from the back-end XML page according to the user behaviour.

The original contribution of the chapter stands in the mix of conceptual aspects and industrial-based solutions that lead to a comprehensive conceptual view of the development of RIAs. To our knowledge, this is the first attempt to bring together academic research and industrial implementation in conceptual modeling of RIAs.

To validate the approach, we exemplify the usage of the devised components in a real business scenario in which WebRatio has been adopted for designing and implementing RIA applications.

The chapter is organized as follows: we start describing the role of RIAs in the context of Web 2.0; then we summarize some background information about RIAs and about WebML. Subsequently, we move to the core part of the chapter, describing the new conceptual modelling primitives for supporting RIAs; hence we describe the WebRatio architecture and the extensions needed for RIAs, and an industrial case study where the approach has been applied; finally, we draw some conclusions on the work.

THE ROLE OF RICH INTERNET APPLICATIONS IN WEB 2.0

RIAs represent one of the mainstream evolutions of Web applications that are recently taking place. Together with other evolution aspects, they contribute to the innovation of the Web in a subtle but radical way. Among these aspects, we can cite the success of the Web 2.0 applications, whose main characteristic is the deep involvement of end-users in the success of applications, based on community behaviour, on continuous and pervasive user interaction, and on contents mainly provided by end users. The novelty of these trends does not stand only in technical innovations introduced in the Web applications, but on the new ways of using the existing technologies for achieving different objectives. Among the other phenomena that we are witnessing, rich interfaces are not usually considered as a mainstream innovation of Web 2.0. However, Web 2.0 sites very often feature a rich, user-friendly interface based on AJAX or similar rich media. In some sense, RIAs can be seen as one enabling technique for strictly speaking Web 2.0 applications. Indeed, a lot of community and interaction features rely on user friendly interfaces. Without them, many user activities involved in Web 2.0 applications (although technically feasible) would be so complex and boring that many users would probably simply give up interacting.

BACKGROUND

Our proposal of extension towards AJAX and RIAs of a well known and established CASE tool for Web application design is positioned in a quickly changing scenario of technologies and tools. We now examine the current state of the art in the field, considering contributions from the different classes of: AJAX toolkits and libraries, AJAX comprehensive IDE tools and frameworks, and conceptual model proposals for AJAX applications. We also briefly describe the WebML models that have been extended in the context of this work.

AJAX libraries and toolkits

Following the exceptional growth of the RIA interfaces, several application frameworks have been proposed in the context of AJAX development. Among them, several opensource and commercial projects provide libraries for rich internet application development. Among them, we can cite the most established ones: Dojo (2008), Ext (2008), Google Web Toolkit (2008), jQuery (2008), MooTools (2008), Prototype (2008) and Scriptaculous (2008), and Yahoo (2008) User Interface Library, but there are hundreds more that are flourishing. Among the advantages of AJAX libraries, we mention the fact that often developers get full access to the source code, even if not released as opensource, thanks to the fact that libraries are almost entirely developed in JavaScript, which is normally visible to developers. The most successful toolkits are probably Dojo, currently supported by IBM and Sun; Ext, a fast-growing toolkit offering both opensource and commercial licenses; and the Google's GWT library, built on Java. To further enrich the scenario, there exist also hybrid versions now that mix and match several of the major projects, like GWT-Ext (Google, 2008, 2) and MyGWT (2008), that mix GWT and Ext, and Tatami (2008), that mixes GWT and Dojo. Other hybrid approaches are positioned between the scripting libraries and the browser desktop-like function libraries, such as XUL (Mozilla, 2008), that are often provided with appropriate development approaches (Brent, 2007).

In the WebRatio runtime framework Prototype and Scriptaculous have been adopted, and thus the code generator of WebRatio produces Javscript code that exploits these libraries. Prototype provides a simple approach to manipulating a Web page, with a relatively light layer that offers both shorthand versions for popular functions and a good amount of cross-browser abstraction. Scriptaculous is a set of special effects and simple widgets built on top of Prototype. Among the various options, the choice of Prototype is motivated by the simplicity of the library, on the widespread usage, and on the good quality and reliability of the results.

AJAX development tools

Besides the various AJAX libraries, a smaller set of development tools for AJAX exist. The four leading tools are Backbase (2008), Bindows, JackBe (2008) NQ Suite, and Tibco (2008) General Interface. All of them offer broad widget collections, rich tools, sophisticated debuggers, and development platforms that rival any of the IDEs for traditional languages. Differently from the toolkits described in the previous sections, these are full frameworks that function best when one builds the entire application on top of their structure.

All of these systems are built around collections of widgets that are joined with a central backbone of events and server calls that link the widgets in a cohesive set of panes. Events flow across a central bus to all parts of the system, with an approach that is closer to desktop application than to Web page design. The major differences among these packages lie not in the capabilities, but in the server support and in the finer details of their approach. Although it may be easy to find one widget or design structure in each of the packages that outshines the others, the cores of the four packages are similar, and they're all built around a core set of UI widgets. All these tools manipulate the DOM tree. The main drawback of these tools is that the resulting presentation tends to be rather boilerplate, not fully showing the interaction quality that might be expected from JavaScript applications. For the interaction with the server, some of the tools expect the data to be packaged in Web services, while others include extensive server frameworks that integrate the client application with backend databases.

With respect to our proposal, these tools are positioned in the IDE field, supporting the developers for writing the code. They don't provide any high-level, model-based design facility.

Model-driven approaches to RIA design

Several researches have tried to highlight the capabilities and features of RIAs. The paper by Preciado et al. (2007) discusses how RIAs extend the behaviour of Web application at different levels: data, business logic, presentation, and communication.

- In the data layer, the client can be exploited for storing non persistent data;
- At the business logic level, some operations (data filtering, numeric operations, and so on) can be delegated to the client;
- At the presentation level, new user events can be managed and new interaction paradigms are allowed (drag and drop, modal windows, and so on)
- At the communication level, new synchronous and asynchronous communication patterns can be exploited, allowing pushing of information from the server and partial page refresh.

The approach presented in this chapter is a pragmatic extension of WebML and WebRatio for supporting those features. These extensions are defined in terms of new properties of WebML components.

Other more comprehensive works, like Bozzon (2006), aim at proposing a new structured framework for event management and interaction design. Specific works address the client-server communication issues (Toffetti, 2007). Other approaches (Brambilla, 2008) exploit workflow modeling for achieving a larger separation of concerns regarding (i) the data and business logic distribution, (ii) the interface behaviour, (iii) the hypertext navigation, and (iv) the presentation aspects. The proposal by Kadri (2007) exploits the UML notation and hierarchical design of components for specifying complex (possibly distributed) Web applications including rich interface behaviours. The proposal is provided with a design and code generation tool. However, the focus is more on the application structure than on the GUI behaviour.

The RUX-Method (RUXProject, 2008) proposes a more clear separation of the presentation at design-time through a stack of models specifically designed for RIAs.

Other recent proposals in the Web Engineering field represent the RIA foundations (e.g., Urbieta, 2007) by extending existing Web engineering approaches. Urbieta et al. (2007) suggests a design approach that extends OOHDM with a good separation of concerns and a UML-like notation. Some works offer insights and experiences on the migration of traditional desktop or client-server applications to Web based application that exploit rich interfaces. Among them, Samir et al. (2007) proposes an approach to translate Java Swing applications to XUL applications.

WebML and WebRatio background

The WebML language and methodology is a high-level notation for data-, service-, and process-centric Web applications. It allows specifying the data model of a Web application and one or more hypertext models that can be based on business process specifications and can exploit Web service invocation, custom backend logic, and rich Web interfaces. The WebML approach to the development of Web applications consists of different phases. Inspired by Boehm's spiral model, the WebML process is applied in an iterative and incremental manner, in which the various phases are repeated and refined until results meet the application requirements.

The WebML language is a Domain Specific Language (DSL) for designing Web applications. This section summarizes the basic WebML concepts, with particular attention to data model and hypertext model.

The WebML methodology and language are fully supported by the CASE tool WebRatio 5.0 (2008), an Eclipse plugin representing a new generation of model driven development (MDD) and engineering (MDE) tools for Web applications. Besides taking advantage of the Eclipse features, WebRatio provides advanced capabilities in terms of support of model extensions, model checking, code generation, project documentation, and collaborative work.

WebML Data Model. For the specification of the underlying data of the Web application, WebML exploits the existing Entity-Relationship data model, or the equivalent subset of UML class diagram primitives. The data model can also include the specification of calculated data. Calculated attributes, entities, and relationships are called *derived* and their computation rule can be specified as a logical expression written using declarative languages like OQL or OCL.

WebML Hypertext Model. The hypertext model enables the definition of the front-end interface of the Web application. It enables the definition of pages and their internal organization in terms of components (called content units) for displaying content. It also supports the definition of links between pages and content units that support information location and browsing. Components can also specify operations, such as content management or user's login/logout procedures (called operation units).

A *site view* is a particular hypertext, designed to address a specific set of requirements. It consists of *areas*, which are the main sections of the hypertext and comprises recursively other sub-areas or pages. *Pages* are the actual containers of information delivered to the user. Several site views can be defined on top of the same data schema, for serving different user roles or devices.

Pages inside an area or site view can be of three types: the home page ("H") is the default address of the site view; the default page ("D") is the one presented by default when its enclosing area is accessed; a landmark page ("L") is reachable from all the other pages or areas within its enclosing module.

Pages are composed of *content units*, which are the elementary components that publish pieces of information within pages. In particular, *data units* represent some of the attributes of a given entity instance; *multidata units* represent some of the attributes of a set of entity instances; *index units* present a list of descriptive keys of a set of entity instances and enable the selection of one of them; *scroller units* enable the browsing of an ordered set of objects; *entry units* allow to publish forms for collecting input values from the user. Units are characterized by a *source* (the entity from which the unit's content is retrieved) and a *selector* (a restriction predicate on the result set of the contents).

Units and pages are interconnected by *links*, thus forming a hypertext. Links between units are called *contextual*, because they carry some information from the *source unit* to the *destination unit*. In contrast, links between pages are called *non-contextual*. Different link behaviours can be specified: an *automatic link* (marked as "A"), is automatically "navigated" in the absence of a user's interaction when the page is accessed. A *transport link* (dashed arrow), is used only for passing context information from one unit to another and thus is not rendered as an anchor.

Parameters can be set as globally available to all the pages of the site view. This is possible through *global parameters*, which abstract the implementation-level notion of session-persistent data. Parameters can be set through the *Set unit* and consumed within a page through a *Get unit*.

WebML also supports the specification of content management, custom business logic, and service invocation. WebML offers additional primitives for expressing built-in update operations, such as creating, deleting or modifying an instance of an entity (represented through the *create*, *delete* and *modify* units, respectively), or adding or dropping a relationship between two instances (represented through the *connect* and *disconnect* unit, respectively). Other utility operations extend the previous set. Operation units do not publish the content to be displayed to the user, but execute some business logics as a side effect of the navigation of a link. Each operation can have two types of output links: the *OK link* is followed when the operation succeeds; the *KO link* when the operation fails. Like content units, operations may have a source object (either an entity or a relationship) and selectors, and may have multiple incoming contextual links, which provide the parameters necessary for executing the operation. Two or more operations can be linked to form a chain, which is activated by firing the first operation.

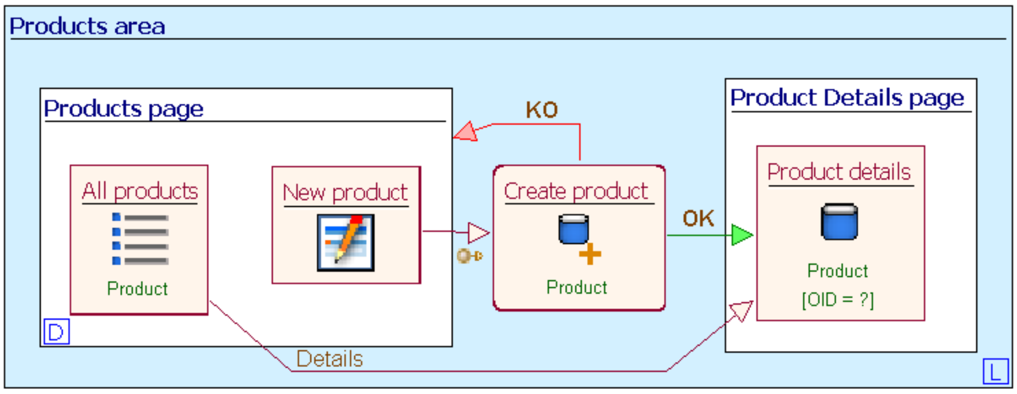


Figure 1. The WebML model for uploading a file with asynchronous behaviour using AJAX.

Figure 1 shows the WebML model representing a Web site area (*Product area*) marked as Landmark (L), comprising two pages: *Products page* (default page of the area) contains the index of *All products* available in the database and a form. By clicking on the index, the used follows the *Details* link to the Product Details page, where the information about the selected product is shown. By submitting information through the *New product* form, the user triggers the execution of the Create product unit, which instantiates a new Product tuple in the database. In case of success, the OK link is followed and the Product details of the newly created element are shown; in case of failure, the KO link is followed toward the Product page.

MODELING PRIMITIVES FOR RIAS

As previously mentioned, RIAs enable a wide set of user interaction patterns that mimic the behaviour of desktop application interfaces. In this section we examine the most common interactions and describe how they can be specified with the WebML conceptual modeling language. A comprehensive specification of the possible interaction patterns can be found in Preciado (2007). In this chapter, the discussion will highlight how the conceptual modeling of the new RIA behaviours differs from traditional web application design.

The main areas where AJAX mechanisms can be applied are the refinement of navigation of links, content publishing in the pages, and user input management. If any of these aspects is expected to be managed with AJAX, at the WebML modeling level the involved objects (pages, units, links) must be marked with the property "AJAX enabled".

Partial page refresh

Thanks to the new features introduced by AJAX, links can represent new kinds of interactions in the pages. In particular, it is possible to define partial refresh of pages and management of separate popup windows with respect to the main application interface.

The partial page refresh consists in the possibility of reloading only some portions of the page, thus making the user interaction with the application quicker and more effective. The (partial) refresh is usually caused by a user click on the interface, although it could be in principle activated by any kind of user event.

The behaviour of the partial page refresh is fundamental in the modeling of RIAs. Therefore, its representation at the modeling level must be straightforward. Webratio allows one to describe this feature simply by marking the link that triggers the page refresh as "AJAX". In this case, by clicking on the AJAX link the user will activate the refresh only of the target of the link (a page, subpage, or group of content units) that are affected by the navigation of the link.

The calculation of page contents, of dependencies among units, and of partial page refresh criteria is a complex task. The visual WebML model relies on a solid page computation algorithm that manages all these issues. The main aspect to be considered is the definition of which parts must be recalculated after a user interaction. WebML specifies calculation semantics (which is implemented in the WebRatio tool too) for its models that relies on the topology of the links: basically, any unit with an incoming link can be seen as dependent with respect to the source unit of the link. When a user submits or selects a value in a unit, all the (recursively) dependent unit must be recomputed. In a traditional Web 1.0 approach, the full page is refreshed anyway, while in a RIA application only the dependent units must be refreshed.

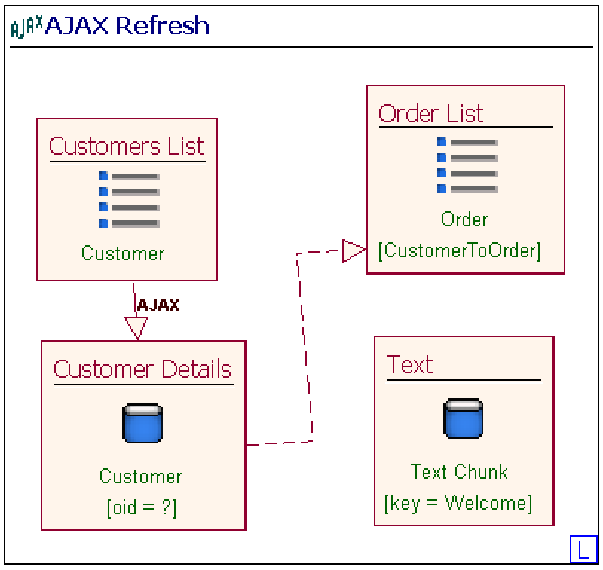


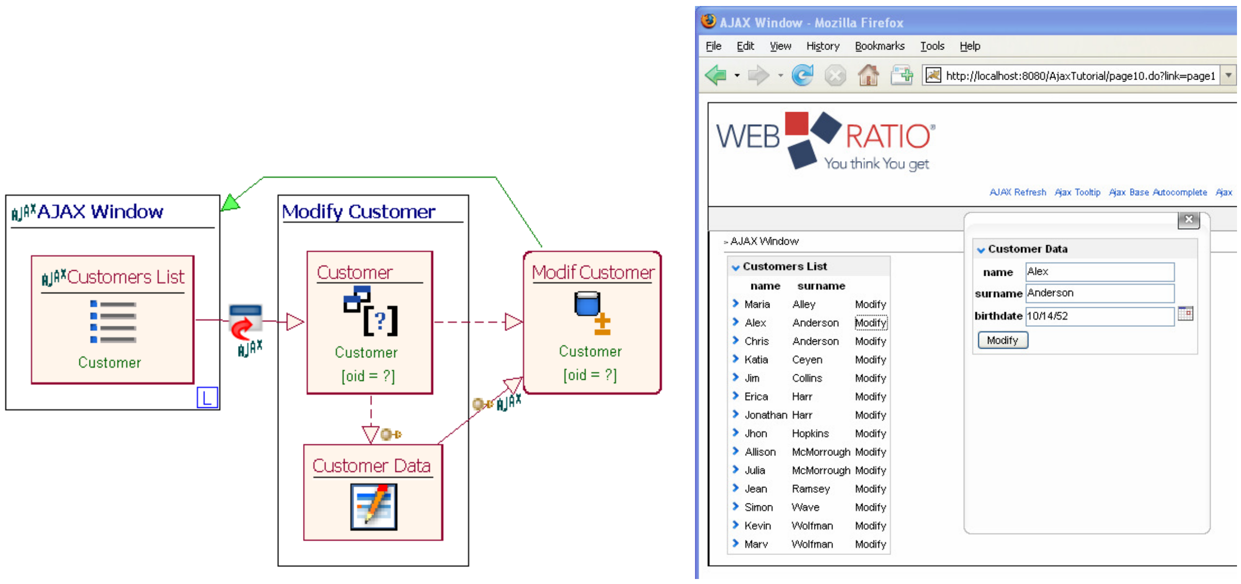
Figure 2. The WebML model for a partial refreshing page.

Figure 2 shows a page with an AJAX link that allows the user to select a customer from a list. The effect of clicking on the link is that the page is partially refreshed: the *Customer details* and *Order list* units are refreshed, because by clicking on the AJAX link the user triggers partial page refresh of all the dependent components. In the example, the content of *Customer details* and *Order list* units depend on the selection of the user. Indeed, they are both connected by (possibly indirect) links to the *Customer List* unit, where the user interaction took place. Notice that *Text* and *Customer list* units are not refreshed, since they are not affected by the user selection.

On-page popup and window management

Another important feature related to the page management is the possibility of defining AJAX Windows within the Web application. Windows can be opened as popups or message boxes in the interface upon user events. AJAX Windows can be defined as modal or not; modal windows do not allow one to interact with the other parts of the application while they are opened.

From an implementation point of view, AJAX windows are placed on a new layer above the page, thus automatically disabling the user interaction on the main page. When the popup window is closed, the content of the main page becomes active again and the user can browse the application.



AJAX Window - Mozilla Firefox

http://localhost:8080/AjaxTutorial/page10.do?link=page1

File Edit View History Bookmarks Tools Help
WEB RATIO
You think You get
AJAX Refresh Ajax Tooltip Ajax Base Autocomplete Ajax
AJAX Window

Customers List

name	surname	
Maria	Alley	Modify
Alex	Anderson	Modify
Chris	Anderson	Modify
Katia	Ceyen	Modify
Jim	Collins	Modify
Erica	Harr	Modify
Jonathan	Harr	Modify
Jhon	Hopkins	Modify
Allison	McMorrough	Modify
Julia	McMorrough	Modify
Jean	Ramsey	Modify
Simon	Wave	Modify
Kevin	Wolfman	Modify
Marv	Wolfman	Modify

Customer Data

name	Alex
surname	Anderson
birthdate	10/14/52

(a) *(b)*

Figure 3. The WebML model of a popup window for editing the customer information.

The WebML model of Figure 3.a represents the navigation from a list of customers (*AJAX Window* page) to a popup *Modify Customer* popup window that displays a form containing the data of the selected customer and allowing to modify the details. The popup window is opened by the AJAX link from the *Customers List* unit. Once opened, the *Modify Customer* popup page retrieves the current *Customer* value and shows a form for updating such values. The popup is closed once the user clicks on the outgoing AJAX link towards the *ModifCustomer* operation, that perform the database updates on the customer instance retrieved through the dashed transport link coming from the *Customer* unit. The rendering of the page is shown in Figure 3.b: the main page contains the *Customers List*, while the AJAX popup contains the form *Customer Data* for updating the information of the selected customer.

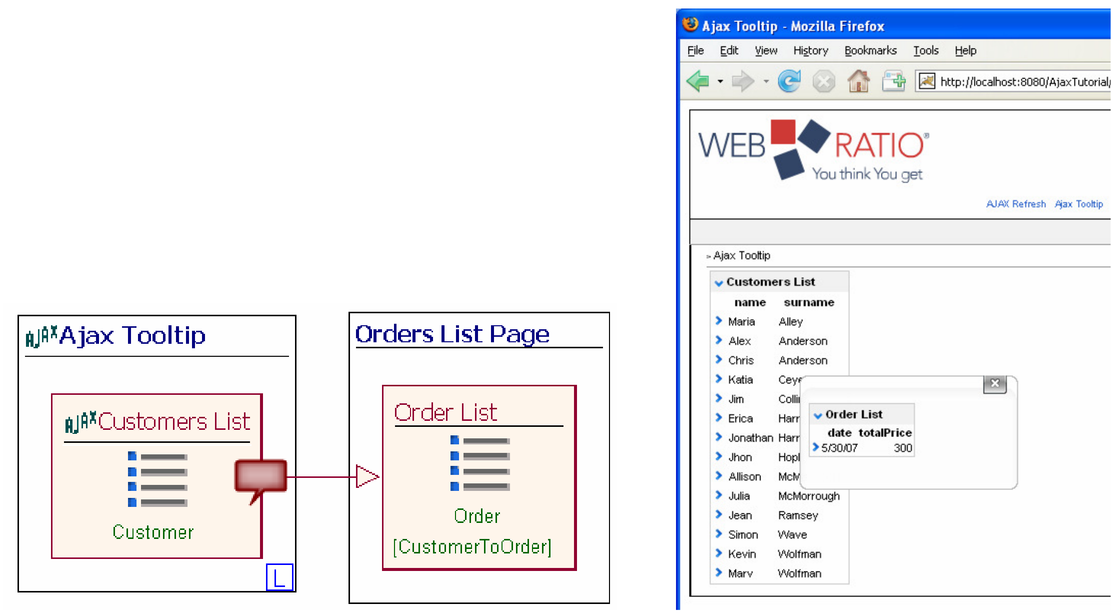
Dynamic tooltips on page data

The tooltip feature allows one to render some information in a dedicated area that is loaded when the user selects an element of the page, without the need to reload the whole page. Tooltips can be shown upon a user action over a page object.

At the modeling level, this can be specified with a simple notation: the tooltip behaviour is associated by a set of properties with the unit where the tooltip is expected to appear. Those properties include: the anchor of the tooltip, i.e., the position where the tooltip is activated (for instance, the link anchors or the attribute values shown in the unit); the triggering event (mouse click, mouse double click, mouseOver, …); the tooltip size (possibly dynamic on the size of the content); the options for drag and drop of the tooltip; and so on.

Once this has been specified, a link exiting the unit can be marked as a *tooltip link*, representing that the activation of the tooltip concretely consist of traversing that link. The actual tooltip content consists of a full WebML page and therefore can be dynamically extracted from a datasource by any set of WebML units. To make a page behave as a tooltip, one of its units must be the destination of the tooltip link.

Figure 4.a shows a simple example of tooltip usage: a page contains an index of *Customers*, which is enabled for tooltips. The outgoing link activates the tooltips upon event. The link leads to a page containing an index that fetches from the database the list of orders previously submitted by the current customer. If we specify that the triggering action is the OnMouseOver event on the attribute values of the index, the content of the *Order List* page will be dynamically shown once the user rolls over the values of the *Customers List* index unit. Figure 4.b shows the rendered page with the tooltip window opened for the current customer.


Ajax Tooltip
Customers List
Customer
L
Orders List Page
Order List
Order
[CustomerToOrder]
Ajax Tooltip - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/AjaxTutorial/
WEB RATIO
You think You get
AJAX Refresh Ajax Tooltip
» Ajax Tooltip

Customers List

name	surname
Maria	Alley
Alex	Anderson
Chris	Anderson
Katia	Ceye
Jim	Colli
Erica	Harr
Jonathan	Harr
Jhon	Hopl
Allison	McM
Julia	McMorrough
Jean	Ramsey
Simon	Wave
Kevin	Wolfman
Marv	Wolfman

Order List

date	totalPrice
5/30/07	300

(a) *(b)*

Figure 4. The WebML model for a dynamic tooltip on an index.

Drag and drop among content components

The Drag and Drop feature allows one to perform some operations in a web page by dragging some elements of the page on others. After data objects are selected and dragged, the drop action causes the execution of any associated side effect. This is a powerful interaction paradigm that can replace the traditional selection of objects in the page, making the interaction more intuitive in several contexts, such as adding a product to the cart, moving an element to a specific folder, and so on.

This behaviour can be modelled simply by specifying that a WebML unit is enabled for the dragging event. Then, the outgoing links marked as AJAX links will behave as drag&drop paths. A drag&drop path is defined as a link from a source unit to a side effect operation (or set of operations) that is performed when the drop event occurs.

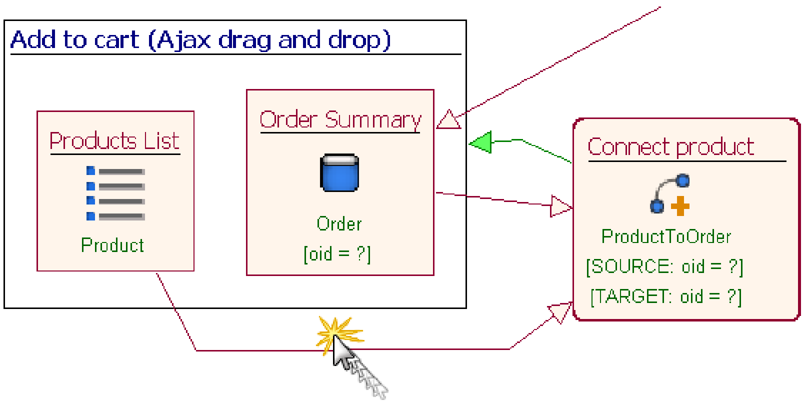


Figure 5. The WebML model for a drag and drop behaviour for an ecommerce cart.

Figure 5 shows an example of drag and drop in a simple ecommerce application. The user may reach the *Add to cart* page by some other page in the site through the incoming arrow in the top-left angle of the figure. There, the user can see the list of available products (*Product List* unit) and can drag and drop them into his own cart, represented by the *Order summary* data unit. The drag link is represented by a symbol of moving mouse pointer, and the effect of the dragging is represented by the *Connect product* unit. The other link that reaches the *Connect product* unit is not meant to be navigated, but instead associates the recipient of the dragging (i.e., the *Order Summary* unit) to the dragging action. Therefore, notice that the drop event can happen only on the *Order summary* unit. In general, the allowed component for dropping objects can be identified because it is connected to the component that is the destination of the drag link.

The drag link in the example leads to the *Connect product* operation, that connects the dropped product to the Order. Once Order and Product are connected, the user is redirected to the page through the OK link exiting the *Connect product* operation.

Event management and dynamic dependencies in the page

While traditional Web applications only rely on the onClick event for the behaviour of the user interface, AJAX allows to handle a much wider set of events, thus making the user interface more usable. Thanks to AJAX events, the designer can specify the actions to be performed by the system after the occurrence of a specific event, usually associated to form fields.

The elementary events that can be managed are:

- onChange event: allowing one to define update policies among preloaded fields; the typical application is the refresh of the contents of a drop down list depending on the changed value of another field;
- onFocus event: allowing one to calculate and show some contents when the user moves the cursor to a specific field; this may be useful for showing some instructions or hints when the user enters a field;
- onBlur event: allowing one to execute some action when the cursor leaves a field; typically, this is used for applying validation checks on the input data.
- These options can be specified as field properties: each field can be marked as sensitive to a specific event and can be associated with an outgoing link that manages this event.

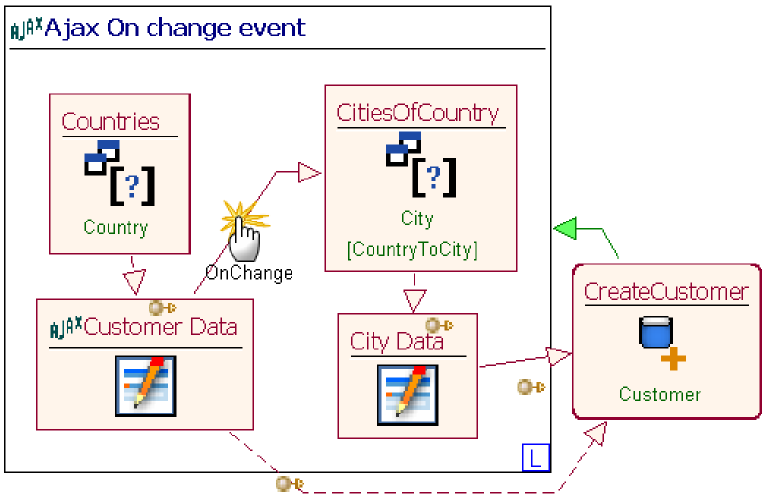


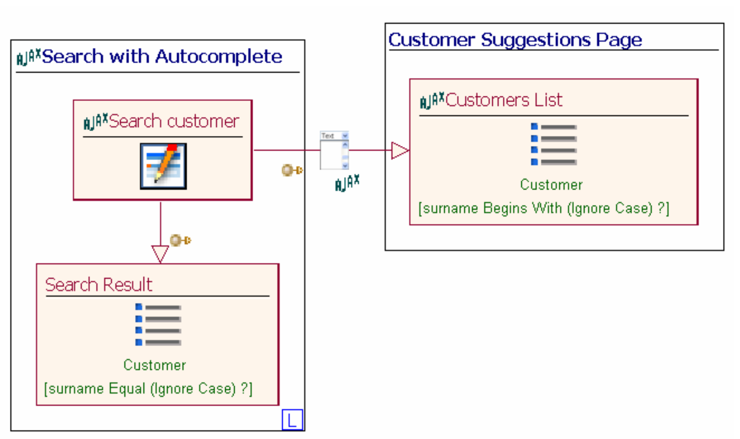
Figure 6. The WebML model for managing the OnChange event on the customer data submission.

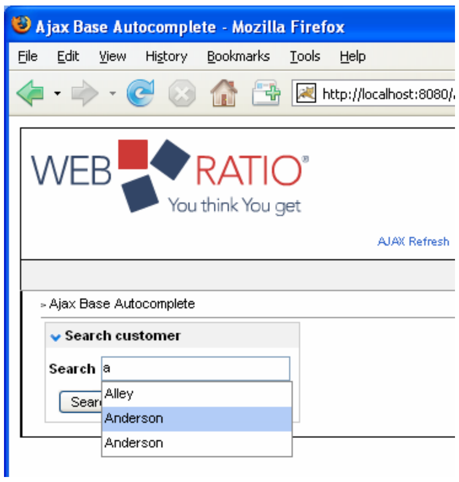
Figure 6 shows a simple WebML page aiming at adding a new customer in a database. In this page, the set of existing countries is immediately extracted (by the *Countries* query unit) from the database and used for populating a dropdown field in the *Customer Data* entry unit. The field is enabled for catching the *OnChange* event, associated to the highlighted link. When the user makes his choice for the country of the customer, the OnChange link is triggered and the list of available cities is recalculated based on the selection (by the *CitiesOfCountry* query unit). Therefore, the choice of the customer city is limited to those belonging to the country previously selected. Once the user finally selects a city, he triggers the *CreateCustomer* unit, which actually creates a new Customer instance in the database (including the city information). Analogous behaviours can be defined by exploiting the other kinds of events mentioned above.

Field content autocompletion

Autocompletion consists of a set of suggestions that is shown to the user while he types textual contents in some field of a form. In this way, the user can type only the initial letters of the word or of the value that has to be inserted in the field and the application shows different options among which the user can choose.

The AJAX autocomplete feature allows one to specify an information source that is used as suggestion. This can be specified by setting the AJAX Autocomplete property on the interesting fields of the form; then for each autocomplete field one outgoing link marked as AJAX Autocomplete identifies a page that is rendered within the autocomplete drop down menu.


AJAX Search with Autocomplete
AJAX Search customer
Search Result
Customer
[surname Equal (Ignore Case) ?]
L
AJAX
Customer Suggestions Page
AJAX Customers List
Customer
[surname Begins With (Ignore Case) ?]



(a) (b)

Figure 7. The WebML model enabling autocompletion on the customer surname field of a form.

Figure 7.a shows a page containing a search form for customers. The form has an outgoing link toward the *Search Result* index that displays all the customers whose surname matches the input of the user. Besides the traditional search behaviour, the form is equipped with the autocompletion facility represented by the Ajax link leading to the *Customer suggestion* page. This link associates a query that extracts the list of customers beginning with the letters typed by the user. This list is shown as a set of autocompletion options to the user.

This behaviour can be generalized by exploiting complex, possibly multiple, dependencies between fields. A typical scenario is autocompletion of geographical information. Once a country is specified in a field, the autocompletion of the city field will consider both the value of the country field and the partial text that the user is entering as city name. Figure 7.b shows the online application that displays the autocompletion suggestions when the user has typed the "a" character in the field.

Management of background file uploads

The AJAX file upload feature allows the user to upload files into a Web application with asynchronous behaviour, allowing the interaction with the application during the file upload. In standard Web applications this is not possible: the user has to wait until the upload operation is finished before continuing the navigation. To implement this particular feature there is a technological limitation in Javascript that must be solved: JavaScript does not allow access to the local file system resources for security issues. This means that no selection and upload of local files can be performed. The AJAX file upload can only be implemented by defining a separate page containing the upload field, which is then included as an iframe in the main page, so that the upload request is processed independently from the rest of the application.

To capture this behaviour, two different pages must be defined. The first page represents the iframe containing the upload field and the upload operation. The second page represents the main application page from which the user will start the file upload.

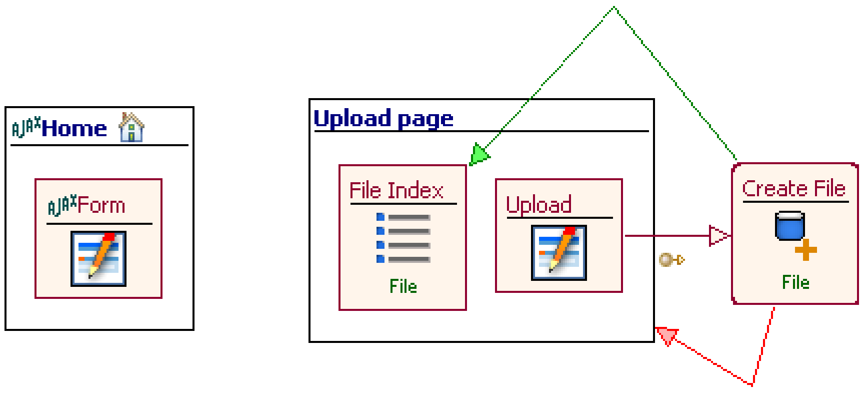


Figure 8. The WebML model for uploading a file with asynchronous behaviour using AJAX.

Figure 8 shows the WebML model representing a background upload, supposing to have one entity called File in the data model, with a BLOB attribute. The *AJAX Home page* is the main Web site page, which embeds the upload iframe, while the *Upload page* models the iframe containing the actual uploading.

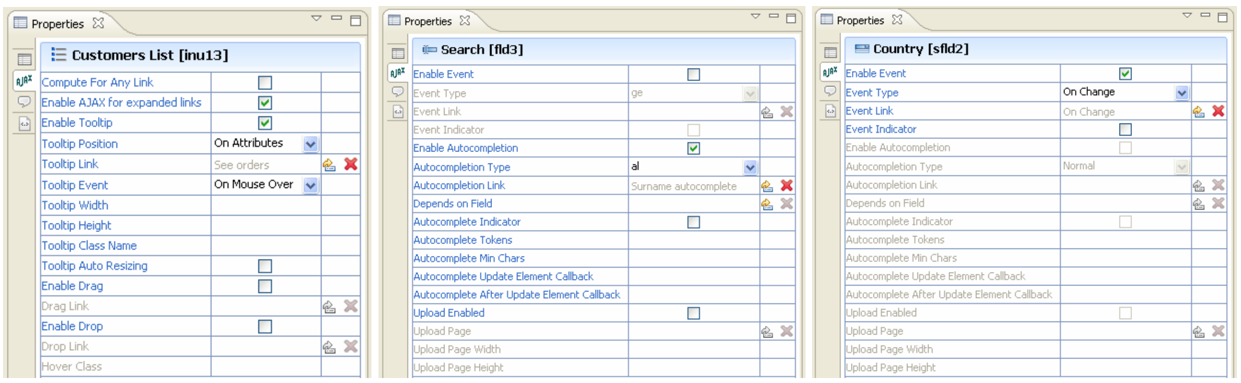
The Home page contains an AJAX form with a single field. This form is associated through a property to the *Upload page* iframe, that will be properly rendered in the page. The Upload page contains the *Upload* form with the actual upload field, which triggers the *Create file* operation. The same page contains also the list of files uploaded up to now (*File Index* unit). Once the user clicks on the Submit button in the *Upload* form, the link is followed and the *Create File* operation is performed.

WEBRATIO DESIGN TOOL AND RUNTIME ARCHITECTURE

WebRatio 5 (2008) is an Eclipse plug-ins that fully supports design and development of Web applications based on the WebML language and methodology. The tool provides a set of visual editors for the WebML models, some model checking and design facilities (wizards, property panels, and so on) for the developer, and a set of code generators for producing the running application. The full description of the tool is available in Acerbis (2008).

Design facilities for AJAX properties

At *design-time*, the WebML editors allow one to model the application and to save it as an XML project. The WebRatio Eclipse perspective comprises several panels: visual model editors, advanced text editors, form-based editors components and properties, wizards, and documentation editors.



Properties

Customers List [inu13]

Compute For Any Link	☐
Enable AJAX for expanded links	✓
Enable Tooltip	✓
Tooltip Position	On Attributes
Tooltip Link	See orders
Tooltip Event	On Mouse Over
Tooltip Width	
Tooltip Height	
Tooltip Class Name	
Tooltip Auto Resizing	☐
Enable Drag	☐
Drag Link	
Enable Drop	☐
Drop Link	
Hover Class	

Properties

Search [fld3]

Enable Event	☐
Event Type	ge
Event Link	
Event Indicator	☐
Enable Autocompletion	✓
Autocompletion Type	al
Autocompletion Link	Surname autocomplete
Depends on Field	
Autocomplete Indicator	☐
Autocomplete Tokens	
Autocomplete Min Chars	
Autocomplete Update Element Callback	
Autocomplete After Update Element Callback	
Upload Enabled	☐
Upload Page	
Upload Page Width	
Upload Page Height	

Properties

Country [sfld2]

Enable Event	✓
Event Type	On Change
Event Link	On Change
Event Indicator	☐
Enable Autocompletion	☐
Autocompletion Type	Normal
Autocompletion Link	
Depends on Field	
Autocomplete Indicator	☐
Autocomplete Tokens	
Autocomplete Min Chars	
Autocomplete Update Element Callback	
Autocomplete After Update Element Callback	
Upload Enabled	☐
Upload Page	
Upload Page Width	
Upload Page Height	

(a) *(b)* *(c)*

Figure 9. AJAX property panels for describing various RIA behaviours within WebML components.

The editing of the AJAX features can be performed mainly through the property panels of the WebML visual components. Figure 9 shows three examples of property panels for setting AJAX features. Figure 9.a sets the tooltip behaviour of a unit: in this example, the event is OnMouseOver; the active position is set on the attributes; and the link to be followed is the *See orders* link. This means that when the user moves the mouse over any content attribute shown by the unit in the page, a tooltip appears displaying the contents of the page reached by the *See orders* link. Figure 9.b, referring to the example in Figure 7, enables the autocompletion of a field and defines the link to the page containing the autocomplete hints (*Surname Autocomplete*). Finally, Figure 9.c, referring to the example in Figure 6, activates the OnChange event on the Country field and specifies the Link to be followed when the event is triggered. The WebML models are enriched by these specifications and can hence be used as starting point for automatic AJAX code generation.

Runtime architecture

The *run-time* WebRatio framework exploits a set of off-the-shelf object-oriented components for organizing the business tier, as shown in Figure 10.

For every page, there is one main JSP template in charge of displaying the full page interface with the needed contents, and one auxiliary JSP template, which contains the data elements to be retrieved by AJAX. Every kind of WebML unit is associated to one service class. At runtime a single service class is deployed for each type of unit and one runtime XML descriptor is generated for each unit instance used in the application design. For instance, every *Data unit* in the project will be executed by the same *Data unit component*, which will be configured by several descriptors (one for each actual *Data unit* instance), to provide the respective functionalities.

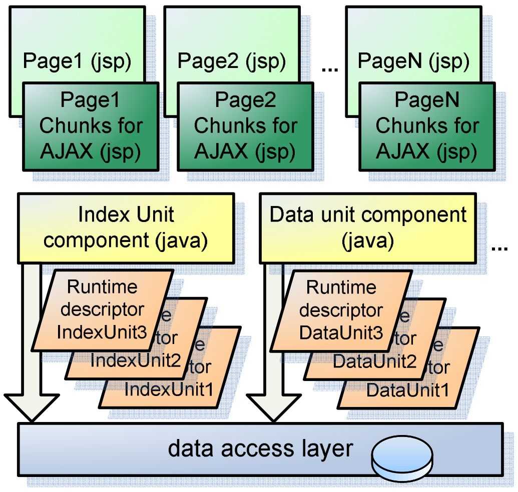

Page1 (jsp)
Page2 (jsp)
...
PageN (jsp)
Page1 Chunks for AJAX (jsp)
Page2 Chunks for AJAX (jsp)
PageN Chunks for AJAX (jsp)
Index Unit component (java)
Data unit component (java)
...
Runtime descriptor IndexUnit3
IndexUnit2
IndexUnit1
Runtime descriptor DataUnit3
DataUnit2
DataUnit1
data access layer

Figure 10. Runtime view of Java components and XML descriptors for WebRatio units.

AJAX Templates organization

Code generation and runtime support of AJAX features have been developed by means of a mix of opensource technologies, extensively exploiting the XMLhttpRequest method. Namely, the AJAX toolkits of choice are Prototype (2008) and Scriptaculous (2008). Their characteristics and the comparison with other solutions are presented in the Background Section.

The adopted runtime architectural solution consists of implementing each conceptual hypertext page with two dynamic pages that interact through XMLhttpRequest for providing the rich interface features: the first page is a back-end JSP page that stores the chunks of page that could be requested by an AJAX request; the second page is the front-end JSP page (including the required JavaScript needed for event management) that is shown to the user. The latter invokes extraction of pieces from the back-end JSP page according to the user behaviour, and shows the results and the interface options to the user. Figure 11 pictorially represents this behaviour.

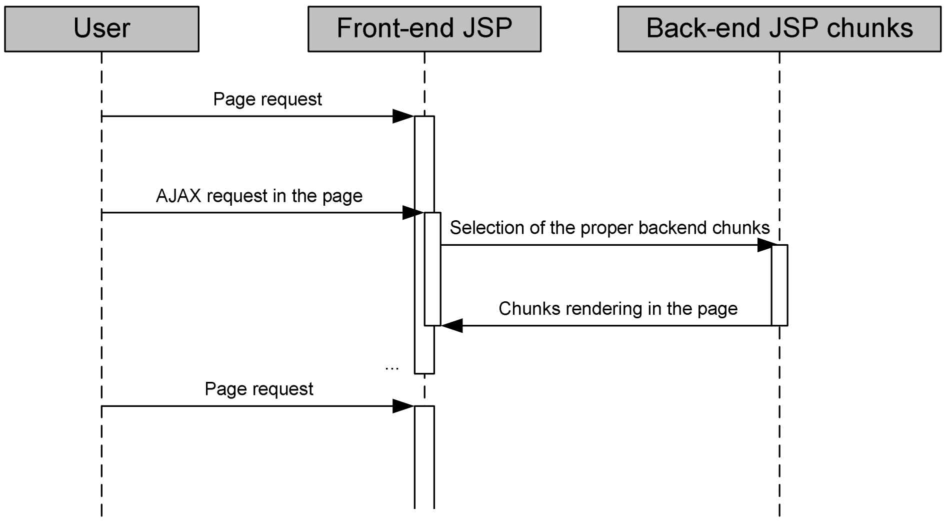


Figure 11. Sequence diagram describing the manipulation of page contents on AJAX events.

As described in the modeling section, special attention must be devoted to asynchronous file upload. Indeed, it requires two distinct dynamic templates. The main template contains a form with the reference to the subpage for the actual upload, which is hence included. The two pages involved in the AJAX upload are generated separately by the code generator. The subpage is generated as a page without any menu and header, and then is included as an iframe in the main page. This is possible simply by specifying that the page graphical style is the Empty page layout, which prints only the page content.

Automatic code generation

The WebML models are transformed into running code by the WebRatio automatic code generation modules. WebRatio code generators produce J2EE Web applications starting from the WebML models. They are developed using the ANT, XSLT, and Groovy technologies. Groovy is a light weight language using a Java-like syntax and fully integrated in the Java Platform, since it is actually translated to Java programs before being executed. It provides many features and facilities that are inspired by scripting languages, but also allows one to exploit all the existing Java libraries.

The generated code includes all the XML descriptors of the units, the front-end JSP pages including the Javascript for the AJAX features, and the back-end pages used as a source of data to be shown upon AJAX events.

Besides existing WebML primitives, it is possible to specify new components (i.e., new WebML units), that can in turn be AJAX-based, and include them in the application design and code generation framework. A custom WebML unit consists of a component made of:

- a Java class that implements the service of the component; and
- a set of XML descriptors, defining the component interface in terms of inputs and outputs.

RIA INDUSTRIAL CASE: EKRP

The presented approach has been applied in the development of several industrial applications that required flexible and user friendly interface. We present now one of these applications, called *eKRP* (electronic Knowledge Repository Process), to demonstrate the feasibility and effectiveness of the proposal. The eKRP application has been developed for a primary Italian textile enterprise, well known at European level for its home textile production, which includes curtains, linens, towels, and bathrobes.

Requirements

The eKRP Web application aims at providing the research and development division of the company a tool for:

- the technological and social analysis of the state of the art and of the user requirements, based on market analysis, on competitor and internal products, and on user's opinions;
- the management of the creative process of invention and of concept definition.

The application is therefore a facility for *(i)* identifying the emerging trends in the reference market and in other related markets; *(ii)* capturing and structuring the data resulting from these trends; *(iii)* interpreting and sharing the findings; and *(iv)* feed the new results to the creative people for the definition of new ideas. Eight main use cases are identified:

- User profile development: the administrator can dynamically manage and organize user profiles and categories, and apply different access rights to different profiles;
- Stimuli profiling: users can dynamically define the profiles for the various stimuli, i.e., inputs coming from the market, that can feed the creativity process;
- Data entry: users must be able to access content management interfaces for the defined stimuli;
- Cluster development: users can cluster the stimuli, thus aggregating inputs based on the expertise and sensitivity of the specific persons, by means of visual diagrams and interactive graph editing. This leads to a first mapping of the interesting areas and trends. The management of the clusters

must be provided at two levels: a view “in the large” allows one to visualize and edit the position of the clusters, while a view “in the small” allows one to manage the internal structure of a cluster, in terms of trends and stimuli that pertain to the cluster;

- Polarization development: users can create and manage the polarization of the clusters that are considered strategic for the company;
- Orientation and design direction: users can manage and organize the processes of creative development of new ideas;
- Backoffice management: users can access the usage history of the eKRP system and can check the system performances in terms of measurable advantages on the resulting products.

Design

The whole eKRP application has been developed using WebRatio. All the interface requirements have been fulfilled, thanks to a deep use of AJAX features throughout the project. The design and development followed the steps specified by the best practices of Web engineering, according to the WebML methodology (Ceri, 2002).

The resulting design consists of two siteviews, one devoted to the company user and one for the administrator: the overall application is organized in 79 pages and 1265 WebML units. Around 60 pages incorporate some AJAX feature. Some new custom components (units) have been developed too: the ErrorsCheckUnit (for checking the correctness of inputs coming from multiple and possibly alternative forms), the ThumbnailCreateUnit (for automatically generating thumbnails from uploaded images), and the WGetUnit (for downloading and storing a Web page, including all its resources, such as css, javascript, images, and so on).

The peculiar aspect of the developed application is the sophisticated interface of most pages. This reflects a typical trend of RIAs: pages tend to be more and more complex, while their number decreases (because several functions are gathered into one page).

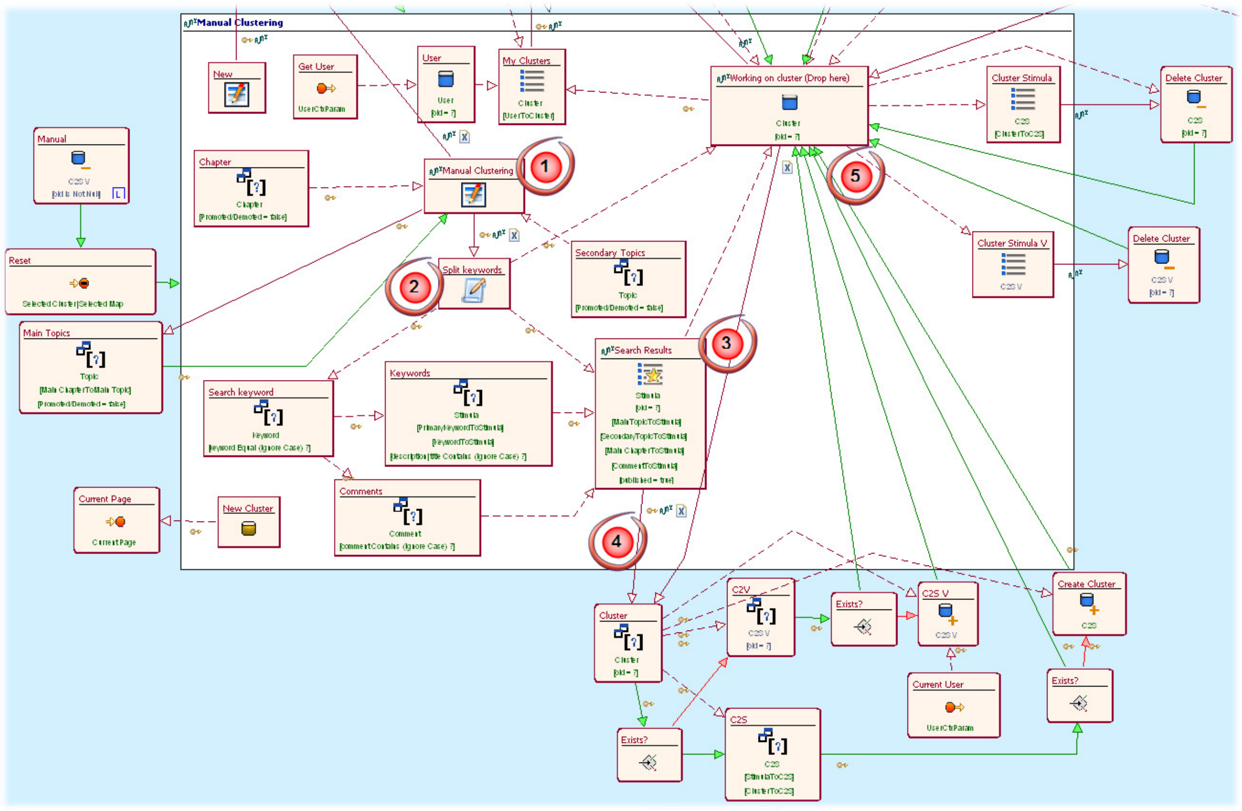


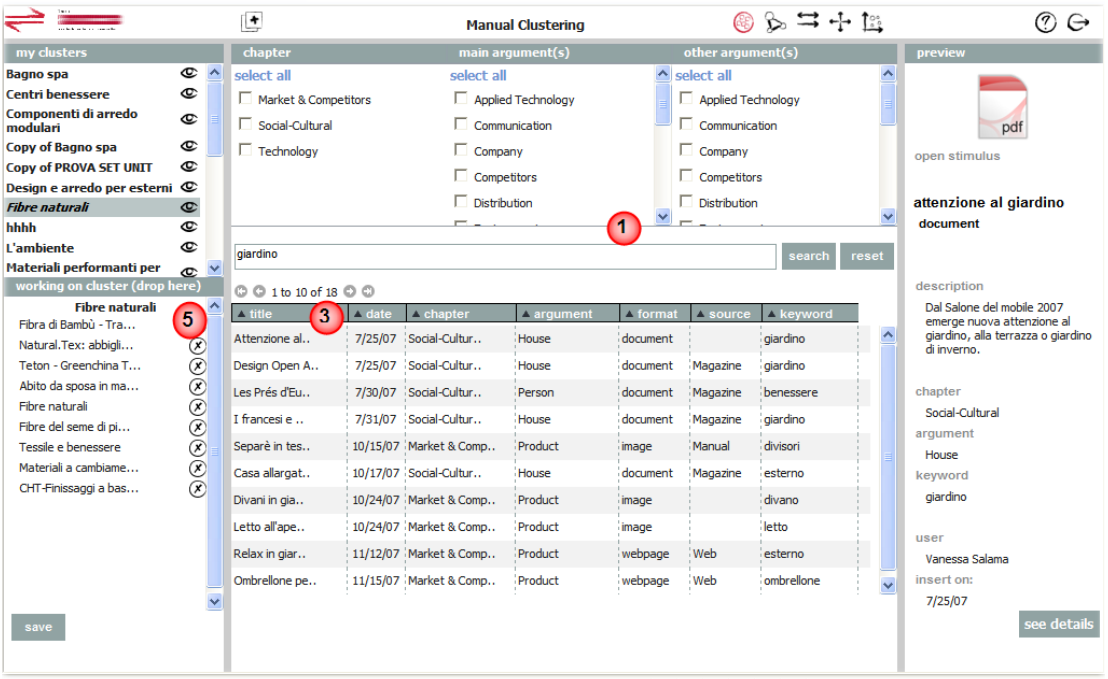
Figure 12. WebML model of the Manual Clustering page in the eKRP application.

For space reasons, the design cannot be reported entirely in this chapter. To give a flavour of the features that can be obtained with the AJAX components, we show in Figure 12 the WebML model of the *Manual Clustering* page, one of the most complex of the eKRP application.

The page allows to manually create and modify clusters of stimuli. The main function of the page is provided through the search for available stimuli within the system, according to some criteria (Chapter, Argument, and Keyword). The search is implemented by the *Manual Clustering* entry unit, the component (1) in Figure 12, where the user submits the search criteria. The form is equipped with full-fledged AJAX validation and comprises many dynamic dependencies among fields. The submitted keywords are split into separated strings by the *Split keywords* unit (2) and are fed to the parametric query, together with the other criteria. The results of the search are displayed in the *Search results* unit (3), that shows all the stimuli matching the criteria. The user can define (or redefine) a cluster by dragging one or more results to the *Working on cluster* data unit (5). This is obtained through the link (4), which is associated with the drag event. When the user drops an element in the data unit (5), the side effects defined for the link (4) are performed.

Implementation

The implementation of the application completely relies on the code generation features of WebRatio: a visual style has been defined for satisfying the requests of the customers, and has been applied by the code generators. The only handwritten code is the one needed for the ad hoc business logic of new custom units.


Manual Clustering

my clusters
Bagno spa
Centri benessere
Componenti di arredo modulari
Copy of Bagno spa
Copy of PROVA SET UNIT
Design e arredo per esterni
Fibre naturali
hhhh
L'ambiente
Materiali performanti per

working on cluster (drop here)
Fibre naturali
Fibra di Bambù - Tra...
Natural.Tex: abbigli...
Teton - Greenchina T...
Abito da sposa in ma...
Fibre naturali
Fibre del seme di pi...
Tessile e benessere
Materiali a cambiame...
CHT-Finissaggi a bas...

chapter	main argument(s)	other argument(s)
select all	select all	select all
Market & Competitors	Applied Technology	Applied Technology
Social-Cultural	Communication	Communication
Technology	Company	Company
	Competitors	Competitors
	Distribution	Distribution

giardino — search — reset

1 to 10 of 18

title	date	chapter	argument	format	source	keyword
Attenzione al..	7/25/07	Social-Cultur..	House	document		giardino
Design Open A..	7/25/07	Social-Cultur..	House	document	Magazine	giardino
Les Prés d'Eu..	7/30/07	Social-Cultur..	Person	document	Magazine	benessere
I francesi e ..	7/31/07	Social-Cultur..	House	document	Magazine	giardino
Separè in tes..	10/15/07	Market & Comp..	Product	image	Manual	divisori
Casa allargat..	10/17/07	Social-Cultur..	House	document	Magazine	esterno
Divani in gia..	10/24/07	Market & Comp..	Product	image		divano
Letto all'ape..	10/24/07	Market & Comp..	Product	image		letto
Relax in giar..	11/12/07	Market & Comp..	Product	webpage	Web	esterno
Ombrellone pe..	11/15/07	Market & Comp..	Product	webpage	Web	ombrellone

save

preview

open stimulus

attenzione al giardino
document

description
Dal Salone del mobile 2007 emerge nuova attenzione al giardino, alla terrazza o giardino di inverno.

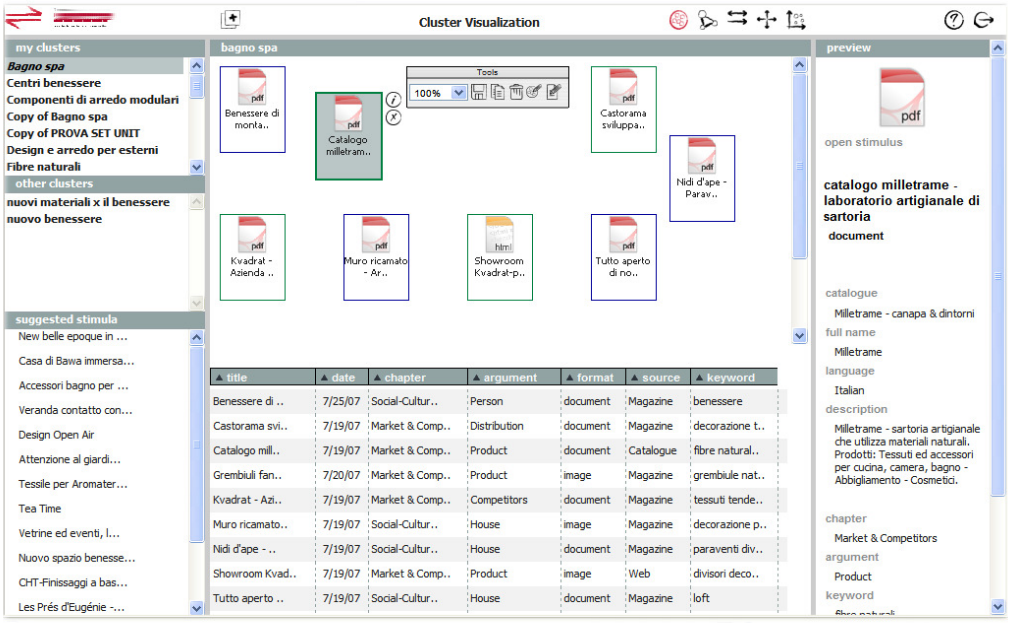
chapter
Social-Cultural
argument
House
keyword
giardino

user
Vanessa Salama
insert on:
7/25/07

see details

Figure 13. Snapshot of the Manual Clustering page interface in the eKRP application.

Figure 13 shows the Manual Clustering page interface, corresponding to the model shown in Figure 12. The top central area (1) of the page represents the form that collects the search criteria (in terms of Chapters, Main arguments, and Other arguments). When the user clicks on the search button, the query is performed and the resulting stimuli are shown in the subpage (3). Stimuli can be dragged from the subpage (3) to the *Working on cluster* panel on the left (5), thus redefining the members of the cluster. Other pages are then devoted to the visualization of clusters. For instance, Figure 14 shows a page with a graph of clusters in the top central part of the screen. This graph is implemented by a custom AJAX-enabled unit that allows one to show, edit, and save the set of clusters.


Cluster Visualization

my clusters
Bagno spa
Centri benessere
Componenti di arredo modulari
Copy of Bagno spa
Copy of PROVA SET UNIT
Design e arredo per esterni
Fibre naturali

other clusters
nuovi materiali x il benessere
nuovo benessere

suggested stimula
New belle epoque in ...
Casa di Bawa immersa...
Accessori bagno per ...
Veranda contatto con...
Design Open Air
Attenzione al giardi...
Tessile per Aromater...
Tea Time
Vetrine ed eventi, l...
Nuovo spazio benesse...
CHT-Finissaggi a bas...
Les Prés d'Eugénie -...

bagno spa

title	date	chapter	argument	format	source	keyword
Benessere di ..	7/25/07	Social-Cultur..	Person	document	Magazine	benessere
Castorama svi..	7/19/07	Market & Comp..	Distribution	document	Magazine	decorazione t..
Catalogo mill..	7/19/07	Market & Comp..	Product	document	Catalogue	fibre natural..
Grembiuli fan..	7/20/07	Market & Comp..	Product	image	Magazine	grembiule nat..
Kvadrat - Azi..	7/19/07	Market & Comp..	Competitors	document	Magazine	tessuti tende..
Muro ricamato..	7/19/07	Social-Cultur..	House	image	Magazine	decorazione p..
Nidi d'ape - ..	7/19/07	Social-Cultur..	House	document	Magazine	paraventi div..
Showroom Kvad..	7/19/07	Market & Comp..	Product	image	Web	divisori deco..
Tutto aperto ..	7/19/07	Social-Cultur..	House	document	Magazine	loft

preview

open stimulus

catalogo milletrame - laboratorio artigianale di sartoria
document

catalogue
Milletrame - canapa & dintorni
full name
Milletrame
language
Italian
description
Milletrame - sartoria artigianale che utilizza materiali naturali. Prodotti: Tessuti ed accessori per cucina, camera, bagno - Abbigliamento - Cosmetici.

chapter
Market & Competitors
argument
Product
keyword

Figure 14. Snapshot of the Cluster Visualization page interface in the eKRP application.

CONCLUSIONS

This chapter presented the features available in the WebML language for supporting RIA behaviour of Web applications, based on AJAX extensions. The primitives have been implemented in the WebRatio design tool, which now features visual modeling of AJAX characteristics and automatic generation of the code.

The proposed approach is very pragmatic and relies on relatively simple design abstractions, if compared to more comprehensive design proposals. The reason of this choice stands on the need of a practical and efficient way for describing the typical RIA interactions. In addition, the approach based on simply extending the WebML features with a set of properties of the standard objects allows full backward compatibility with respect to traditional WebML models and do not require significant effort for the WebML designers to become accustomed to the new features.

Another need covered by the approach is the relatively easy and quick implementation of the new features, both at the design level (in terms of new panels, properties, and primitives in the hypertext model editor) and at the code generation level (in terms of updates to the code generator).

Thanks to the model-driven design approach adopted in our proposal, good separation of concerns between conceptual components and implementation details is achieved. Indeed, the conceptual components specified in WebML and the corresponding interaction paradigms presented in Section 4 are not bound to the AJAX technology. They represent general RIA behaviours that could be implemented in any technology. Therefore, components (and existing application models) could be reused without any change in case of technology switching. It would be enough to reimplement the runtime classes of the components in the new platform of choice (i.e., other AJAX libraries or different paradigms, like Laszlo, Flex, or XUL) and to regenerate automatically the existing applications. The result would be an application implemented in the new technology.

Currently, the approach is already adopted for the implementation of real enterprise Web applications, as shown in the discussed case study. The percentage of developed applications that include AJAX features is continuously increasing since the first day of availability within the tool. This is a clear symptom of the success of the RIA interfaces among the users, which definitely calls in the Web Engineering field to develop comprehensive design facilities for this kind of applications.

REFERENCES

[1] Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., & Fraternali, P. (2008). Web applications design and development with WebML and WebRatio 5.0. In *Proceedings of TOOLS Europe 2008*, LNBIP Vol. 11 (pp. 392–411): Springer.

[2] Backbase (2008). http://www.backbase.com/

[3] Bindows (2008). http://www.bindows.net/

[4] Bozzon, A., Comai, S., Fraternali, P., & Toffetti Carughi G. (2006). Conceptual Modeling and Code Generation for Rich Internet Applications. In *Proceedings of ICWE 2006, International Conference on Web Engineering*, (pp. 353-360): ACM Press.

[5] Brambilla, M., Preciado, J.C., Linaje, M., & Sanchez-Figueroa, F. (2008). Business Process -based Conceptual Design of Rich Internet Applications. In *Proceedings of ICWE 2008*. Yorktown Heights, USA: IEEE Press.

[6] Brent, S. (2007). XULRunner: A New Approach for Developing Rich Internet Applications. *IEEE Internet Computing*, 11 (3), 67-73.

[7] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., & Matera, M. (2002). *Designing Data-Intensive Web Applications*, San Francisco, CA, USA: Morgan Kauffmann.

[8] Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., & Saint-Paul, R. (2007). Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. *IEEE Internet Computing*, vol. 11(3), 59-66.
[9] Dojo (2008). http://dojotoolkit.org/
[10] Ext (2008). http://extjs.com/
[11] Google (2008). Google Web Toolkit: http://code.google.com/webtoolkit/
[12] Google (2008). GWT-Ext: http://code.google.com/p/gwt-ext/
[13] JackBe (2008). http://www.jackbe.com/
[14] jQuery (2008). http://jquery.com/
[15] Kadri, R., Tibermacine, C., & Le Gloahec, V. (2007). Building the Presentation-Tier of Rich Web Applications with Hierarchical Components. In *Proceedings of WISE 2007, Web Information Systems Engineering*, LNCS Volume 4831/2007, pp. 123-134, ISSN 0302-9743, ISBN 978-3-540-76992-7: Springer.
[16] Linaje, M., Preciado, J.C., & Sánchez-Figueroa, F. (2007). Engineering Rich Internet Application User Interfaces over Legacy Web Models. *IEEE Internet Computing*, 11 (6), 53-59.
[17] Preciado, J.C., Linaje, M., Comai, S., & Sanchez-Figueroa, F. (2007). Designing Rich Internet Applications with Web Engineering Methodologies. In *Proceedings of International Symposium on Web Site Evolution*, pp. 23-30: IEEE Press.
[18] Preciado, J.C., Linaje, M., & Sánchez-Figueroa, F. (2007). An approach to support the Web User Interfaces evolution. In *ICWE Workshop on Adaptation and Evolution in Web Systems Engineering*, pp. 94-100: Springer.
[19] Preciado, J.C., Linaje, M., & Sánchez-Figueroa, F., & S. Comai (2005). Necessity of methodologies to model Rich Internet Applications. In *Proceedings of International Symposium on Web Site Evolution.* Pp. 7-13: IEEE Press.
[20] MooTools (2008). http://mootools.net/
[21] MyGWT (2008). http://mygwt.net/
[22] Prototype (2008). http://www.prototypejs.org/
[23] RUXProject (2008). http://www.ruxproject.org/
[24] Samir, H., Stroulia, E., & Kamel A. (2007). Swing2Script: Migration of Java-Swing applications to Ajax Web applications. In *Working Conference on Reverse Engineering 2007 (WCRE 2007).* pp. 179-188, ISSN: 1095-1350, ISBN: 978-0-7695-3034-5.
[25] Schwabe, D., Rossi, G., & Barbosa, S. (1996). Systematic Hypermedia Design with OOHDM. In *7th ACM International Conference on Hypertext*. Pp. 116-128. Washington: ACM Press.
[26] Scriptaculous (2008). http://script.aculo.us/
[27] Tibco (2008). Tibco General Interface. http://gi.tibco.com/
[28] Tatami (2008). http://code.google.com/p/tatami/
[29] Toffetti Carughi, G., Comai, S., Bozzon, A., & Fraternali, P. (2007). Modeling Distributed Events in Data-Intensive Rich Internet Applications. In *Proceedings of International Conference on Web Information Systems Engineering*. pp. 593-602.
[30] Urbieta, M., Rossi, G., Ginzburg, J., & Schwabe, D. (2007). Designing the Interface of Rich Internet Applications. In *Proceedings of Latin-American Conference on the WWW*. Pp. 144-153: IEEE Press.
[31] WebRatio (2008). http://www.webratio.com/
[32] Mozilla (2008). XUL. http://www.mozilla.org/projects/xul/
[33] Yahoo (2008). Yahoo User Interface Library. http://developer.yahoo.com/yui/

ADDITIONAL READING

Curl (2008), RIA Knowledge Center, *http://www.curl.com/knowledge-center/*
Macromedia (2002). Requirements for Rich Internet Applications, *http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf*
W3C (2007). XMLHttpRequest, *http://www.w3.org/TR/2007/WD-XMLHttpRequest-20070227/*

W3C (2008). Rich Web Clients Activity, *http://www.w3.org/2006/rwc/Activity.html*
Adobe, Flex Developer Center, *http://www.adobe.com/devnet/flex/*
W3Schools (2008), AJAX Tutorial, *http://www.w3schools.com/Ajax/Default.Asp*

KEY TERMS & DEFINITIONS

Web engineering: scientific discipline studying models, methodologies, tools, techniques, and guidelines for the design, development, evolution, and evaluation of Web applications.

Model Driven Development: software development approach based on the systematic use of models and their transformations throughout the engineering lifecycle of a software system.

Rich Internet Application (RIA): Web application that implement sophisticated user interfaces, and advanced user interaction patterns with respect to traditional Web applications, including partial page refresh, client-side calculation and data storage, drag&drop, and other features.

Partial page refresh: possibility of refreshing single pieces of a Web page upon the occurrence of an event.

Web Modeling Language (WebML): conceptual model and methodology for the visual design of data-intensive, process-intensive, and service-intensive Web applications.

Asynchronous JavaScript and XML (AJAX): set of Web technologies and development techniques used for developing RIAs. Asynchronous client-server interactions are achieved thanks to the XMLHttpRequest object.

Automatic code generation: software engineering technique that allows to automatically generate application code starting from (platform independent) conceptual models.

WebRatio: CASE (Computer Aided Software Engineering) tool for the specification of Web applications according to the WebML modeling language and for the automatic code generation.