Journal of Web Engineering, Vol. 10, No.1 (2011) 070-086
© Rinton Press

STATE-OF-THE-ART AND TRENDS IN THE SYSTEMATIC DEVELOPMENT OF RICH INTERNET APPLICATIONS

GIOVANNI TOFFETTI
Faculty of Informatics, University of Lugano
V. Buffi 13 – 6900 Lugano, Switzerland
toffettg@usi.ch

SARA COMAI
Dipartimento di Elettronica, Politecnico di Milano
P.zza L. da Vinci, 32. I20133 – Milano, Italy
comai@elet.polimi.it

JUAN CARLOS PRECIADO; MARINO LINAJE
Quercus Software Engineering Group.
Universidad de Extremadura. 10071 - Cáceres, Spain
jcpreciado@unex.es; mlinaje@unex.es

Rich Internet Applications (RIAs) are widely adopted Web applications that add the richer interaction, presentation, and client-side computation capabilities of desktop applications to the Web. However, the evolution from Web applications towards RIAs comes at the cost of increased complexity in their development. For this reason, a wide variety of tools and technologies have been proposed in order to streamline their development effort. This paper investigates the current state of the art of the RIA development approaches. The review shows that the current industrial development practice lacks a comprehensive approach to RIA development, supporting all the development steps from the design to implementation, test and maintenance, and helping identifying correct design choices. This is in part due to the severe fragmentation of current RIA technologies that prevents the adoption of a commonly recognized set of best practices resulting in ad-hoc development processes. These aspects are in part treated by research methodologies and some innovative industrial solutions, but also these approaches present some limitations. The paper identifies future research directions for RIAs to fully support their development process and to support their design in a more comprehensive and systematic way, from both industrial and research perspectives.

Key words: Rich Internet Applications, Systematic Development, Web Engineering
Communicated by: (to be filled by the JWE editorial)

1 Introduction

In the last decade, Web applications have continuously evolved, from static Web pages, to pages whose content is generated dynamically, to applications offering sophisticated User Interfaces (UIs). One of the last steps of this evolution is represented by Rich Internet Applications (RIAs) and related technologies that provide capabilities currently adopted by a growing number of Web 2.0 applications, such as multimedia support, sophisticated interactivity and presentation, the possibility to store and process data at the client side, collaborative work, and flexible communication paradigms (like pull, push, disconnected functioning) [17].

Attractiveness and usability have become key issues in current Web development. In these terms, the main reasons behind RIA adoption are user experience improvement and the increased number of capabilities compared to traditional Web applications. However, these benefits come with some drawbacks since RIAs are much more complex to design and develop than their Web 1.0 counterparts. Indeed, while in traditional applications what mattered most in terms of user experience was Web server response-time (and the interaction paradigm left very few alternatives), in a RIA the UI responsiveness is a direct consequence of a much wider range of design choices (e.g., adopted technologies, client-side vs. server-side logic and data, client-server communication, pre-fetching policies, presentation logic, and so on). When also RIA-specific functionalities are considered (e.g., server-push, disconnected functioning, or collaborative work) the range of design decisions that have to be evaluated and combined into a coherent system grows considerably.

Considering both, server-side and client-side components, RIAs can be developed and deployed for different technological platforms (e.g., Silverlight [DL-18], Flex [DL-4], OpenLaszlo [DL-22], and so on). While these technologies are reaching maturity, wide-ranging methodologies for the systematic development of RIAs are lagging behind. The current development practice relies on a set of application development frameworks for client-side code with little or no provision for the management of the complete application life-cycle from design to deployment and evolution, making it difficult to maintain a big picture perspective. Notwithstanding, an alternative solution is promoted by model-driven approaches, proposing methodologies relying on visual languages for the conceptual specification of the application and automatic code generation. They provide a more systematic approach to the development of RIAs, but are scarcely adopted by developers.

The main goal of this paper is to represent the current RIA development reality, classify and compare the development approaches, identify their principles, limitations, and highlight research challenges.

This paper is structured as follows: in Section 2 we provide an overview of RIA features and technologies; in Section 3 we explain the rationale behind the choice of the analysed approaches and the evaluation criteria used for comparison. Sections 4 and 5 compare code-generation frameworks and model-driven development approaches, respectively. Finally, Section 6 discusses the open issues and devises possible future trends and research directions.

2 An overview of RIAs: features, applications, and technologies

Compared to traditional Web applications, RIAs improve the user experience through novel **features** [20], facing four main aspects of the application development:

- *Rich Presentation:* RIAs offer client-side event-handling and widgets similar to those of desktop UIs, allow partial page updates, support interaction with visual data representations and multimedia content (e.g., audio, video).
- *Client Data Storage:* In RIAs it is possible to store data on the client-side with different levels of persistence (temporarily while the application is running or persistently).
- *Client (and Distributed) Business Logic:* In RIAs it is possible to carry out complex operations directly on the client, like for example, data navigation/filtering/sorting with multiple criteria, domain-specific operations and local validation of data. It is also possible to distribute the

Business Logic between the client- and the server-side, (e.g., to validate some form fields on the client and others on the server).

- *Client-Server Communication:* RIAs support (a)synchronous communication between client and server to distribute domain objects, data, and computation, and provide server-push (e.g., in collaborative / monitoring applications).

Depending on the application's intended functionalities, the above features can be combined to obtain for instance standalone applications (with client-side persistent data and logic), collaborative applications (with client-server communication and the server acting as proxy), or simply more appealing UIs for existing Web applications (through rich presentation). In terms of growing number of features and development complexity a RIA typically falls in one of the following **application types**[a] (that can be possibly combined to obtain complex RIAs):

- *Traditional Web applications with RIA-makeover*: where simple isolated RIA capabilities (usually for partial page updates) are added to a traditional Web application (e.g., Facebook).
- *Rich UIs*: Web applications with widget-based UIs, where the client-side logic is an extension layer over the browser, superseding core browser responsibilities, such as handling events and managing states and the rich user interfaces components work in a coordinate way (e.g., Gmail).
- *Standalone RIAs*: Web applications capable of running both inside and/or outside the browser in a connected and/or disconnected fashion (e.g., SlideRocket).
- *Distributed RIAs*: where the application data and logic are (sometimes dynamically) distributed across client and server, on-line collaboration is supported, client-server communication is used to fill the gap between objects and events living across the application components (e.g., Google Docs).

Table 1 shows the mapping between RIA features and common application types.

RIA Features / Type of Application	Rich Presentation	Client Data Storage	Client (and Distributed) Business Logic	Client-Server Communication
RIA-makeover	Limited: partial page update	No	No	Limited: pull for partial page update
Rich UIs	Yes	No	Limited: events and state management	Limited: pull for partial page update
Standalone RIAs	Yes	Yes	Yes	Limited: data synchronization
Distributed RIAs	Yes	Yes	Yes	Yes (push - pull)

Table 1: Rich Internet application types and supported features

[a] A similar classification for AJAX-based RIAs can be found in Gartner's report no. G00136945

Currently, RIAs capabilities can be implemented in a number of different client-side **technologies** that can be broadly classified in three categories, according to the runtime environment:

- *JavaScript-based*: the client-side logic is implemented in JavaScript (the approach is also known as "AJAX", Asynchronous JavaScript and XML) and UIs are based on a combination of HTML and CSS. The main advantage of this approach is that it relies on built-in browser JavaScript support and W3C standards. The main drawbacks are insufficient media support, browser sandboxes limiting, for instance, file system access or persistent storage, and inconsistent browser behaviour. Because of the latter aspect, a large number of libraries and frameworks have been proposed allowing developers to abstract from browser idiosyncrasies (e.g., Backbase [DL-7], Dojo [DL-8], Prototype [DL-23], GWT [DL-10], jQuery [DL-14], etc.). The current W3C's working draft HTML5 [DL-11] is trying to solve many of such limitations.
- *Plug-in-based*: advanced rendering and event processing are granted by browser's plug-ins interpreting specific scripting languages, XML or media files (e.g., Flash [DL-3], JavaFX [DL-12], Silverlight). An advantage of plug-ins (e.g., FlashPlayer [DL-9]) is that they generally support media interaction natively, allow client-side persistence, and provide better performances than interpreted JavaScript. Some plug-ins come already installed in the browsers, but others require user administrative action. However, in some cases they do not provide access to the Operating System (OS) services (e.g., file system).
- *Runtime environments*: applications are downloaded from the Web but are executed outside the browser using a desktop runtime environment (e.g., Java Web Start [DL-13], AIR [DL-2]). These solutions offer the most in terms of client-side capabilities and off-line usage with full access to the underlying operating system. However, they rely on a dedicated runtime environment, which requires users to install it (and might not be available on all platforms, e.g. mobile phones). Many RIA technologies can be used to develop applications for these runtimes (e.g., for Adobe AIR, Javascript-based and/or Flash-based development technologies can be used).

RIA Features / RIA Technologies	Rich Presentation	Client Data Storage	Client (and Distributed) Business Logic	Client-Server Communication
Javascript-based	Limited: no multimedia	Limited: no persistence	Yes	Yes
Plug-in based	Yes	Yes (by means of additional plug-ins)	only some plug-ins	Yes
Runtime environments	Yes	Yes	Yes	Yes

Table 2: RIA technologies and supported RIA features

Table 2 shows the mapping between RIA features and RIA technologies. Most technologies can be used to implement all RIA features. The main limitations are for JavaScript-based ones with respect to multimedia and client-side persistent storage (in browsers not fully compliant with the recent HTML5 draft). Notwithstanding, among the above technologies, the current RIA development practice sees the adoption of the JavaScript-based ones (i.e., AJAX) as the most common choice [14]. The current limitations are typically solved using Flash extensions for video rendering (and Google Gears or Flash Shared Objects when client-side persistent storage is required). The main reasons behind this trend are to be found in the fact that: a) AJAX is felt by many developers as the most open and standard set of technologies (and closer to the incoming HTML5 specification); b) it does not require administrative actions (e.g., installing software) from users; c) it can be easily combined with plug-ins that are built to overcome its limitations.

3 Survey Methodology

Over 150 approaches have been proposed by software vendors [14] and academic researchers [16] to support the new implementation of RIAs or the migration from legacy applications to RIAs.

To better represent the state of the practice, we can fit existing development approaches into three categories, exhibiting homogeneous features:

1) *Code-based* development, where developers choose the set of client- and server-side technologies and code directly in the technology-specific programming language(s).

2) *Framework-based* development, where developers leverage on a set of specific primitives, tools, libraries, and/or code generation techniques that provide some automation of the most repetitive programming tasks for the client- and/or server-side (e.g., Ajax, Adobe Flex, Silverlight, etc.).

3) *Model-driven* development methodologies, where the application designer specifies the application behaviour by creating models consisting of a set of high-level primitives, abstracting from implementation details. Code-generators transform models into executable code for the server and/or the client-side of a RIA (see Section 5).

The order of presentation of the different categories reflects the increasing level of abstraction of concepts and tool support to the structured development process of RIAs.

Approaches not using frameworks are very rare in practice: a developing framework is generally picked by considering it against the required features for the developed application. However, as application complexity increases, more systematic development approaches are needed to improve the whole development process, from its design, to its deployment and maintenance.

In our analysis we will focus only on systematic development approaches, and therefore on framework-based and model-driven development solutions.

Our evaluation is based on a set of criteria organized into three groups (*technology*, *language*, and *process-related* features) fitting the key features identified in Section 2 and answering the following questions:

Technology:

Q1: Is the proposed approach tied to a specific set of (client- or server-side) technologies?

Language(s)/ Development environment:

Q2: What is the programming style (e.g., imperative or declarative) of the supported languages?

Q3: What abstract primitives are provided to support a higher-level specification of the application?

Q4: What is the language scope? Does it address only client or also server features? Does it cover all the RIA features and, as a consequence, can it be applied to all RIA application types?

Development process:

Q5: Which tools support the development process?

Q6: What are the required development phases / steps?

Q7: Who are the actors involved in the process?

These questions should help in understanding how the development process of RIAs is supported (in particular, how much a developer needs to be concerned with the low-level details of implementation like, e.g., programming style and abstractions, as well as how the typical development process unfolds) and in identifying the possible weaknesses and inadequacies in the current state of the art.

4 Framework-driven development

The most adopted development approaches in the Web industry are framework-based. Table 3 gives an overview of the most prominent (according to [14]) frameworks that focus on the client-side code of Rich Internet Applications. The technology, language, and process dimensions are considered. In this table frameworks are ordered in terms of growing number of supported RIA features.

In general, most frameworks share a set of common characteristics:

- They focus only on the client-side aspects, where they are very tied to specific RIA technologies; they are *independent of server-side technology* for accessing server-side *data source*s, since they leverage on SOA or REST paradigms.
- *Advanced communication* behaviours typical of RIAs (e.g., based on push) are instead bound to the usage of a specific server-side support (e.g., streaming in Silverlight, RPC in GWT).
- Considering the adopted languages, the UI implementation is quite similar to standard-based Web UIs: a *declarative language* is used *for UI* definition, and *scripting for event handling* (e.g., HTML with DOM and JavaScript, XML and scripting in MXML, LZX, and XAML). This solution sets up the ground for the separation of designers and developers roles so that specific tools and IDEs can be provided to each role (this is the case, for example, of Adobe FlashBuilder and Catalyst).

Considering the development process, most approaches are *hybrid* in the sense that they are strongly influenced by target developers' backgrounds providing programming metaphors from either *desktop development* (e.g., GWT and JavaFX using Java) or *traditional Web development* (e.g., AJAX or Flash). For both approaches, extensions are provided (typically through APIs) to target and exploit the out-of-the-browser capabilities. Currently, the latter is a key aspect for all RIA frameworks: the struggle is clearly to provide developers with a unified way to produce both native-Web and Web-enabled desktop applications. This is especially relevant for mobile development where Web applications must be wrapped as native applications in many cases.

The main issues with the framework-based development practice in RIAs concern the lack of support for:

1) the *complete application development* (server & client-side business logic, client/server communication, and interaction),
2) the *complete application lifecycle* (no design and testing phase, limited support for maintenance and evolution),
3) the *integration/collaboration among roles* concurring in designing and implementing the final application (typically, using different tools and IDEs).

	Technology		**Language**				**Process**	
				Scope				
	Server-side tech dependence	**Client-side tech dependence**	**Style**	**Server**	**Client**	**Abstractions: Data, Logic, Presentation, Communications**	**Tool support**	**Typical development process**
AJAX libraries (e.g., Dojo, jQuery, Backbase, YUI toolkit)	Independent of server-side technology (usually LAMP)	AJAX libraries (JavaScript enabled)	Declarative UI (HTML), Imperative logic (JavaScript)	-	✓	**D**: virtual datasets at client side **L**.: JavaScript libraries **P**: cross-browser DOM and widgets **C**: cross-browser XMLHTTPRequest	Aptana IDE. Some have lib-specific IDEs, some are IDE-neutral	UI design and coding, no provision for server-side development
AJAX code-generators (e.g., Google Web Toolkit GWT)	Independent of server-side technology	AJAX libraries (JavaScript enabled)	Declarative UI in some cases (e.g., GWT using UiBinder) + Imperative (Java)	-	✓	**D**: application domain objects **L**.: GWT Java library **P**: Java GWT UI components and events C: cross-browser XHR abstraction, RPC	Java IDEs, IDE-neutral	Developers specify UIs and business logic as they would in Java. JS code is generated for the client-side including support for RPC on the server
OpenLaszlo	Independent of server-side technology for data sources. Own application server is mandatory when LZX runtime compilation is required	Both AJAX or Flash plug-in	Declarative UI / Imperative logic (LZX:XML + JavaScript)	-	✓	**D**: datasets **L**: OpenLaszlo JavaScript library **P**: UI description language, UI components and events C: SOA, XML-RPC, and Java Data Transfer Object	IDE4Laszlo (IBM), IDE-neutral	Definition of UI components and behaviour, definition and connection to data services, SWF or AJAX code generation. Server-side development not-covered.
Adobe Flex	Independent of server-side technology. Initially shipped with LCDS a J2EE integration application	Browser Flash plug-in. AIR runtime, for desktop RIAs	Declarative UI / Imperative logic (MXML: XML + ActionScript).	✓[b]	✓	**D**: datamodel **L**: Actionscript libraries **P**: UI description language, UI components and events **C**: AMF, Real Time Message Protocol, HTTP Services, WebServices (SOA/REST) and Remote Objets. More capabilities with LCDS	Flex Builder, FlashDevelop, FlexBean, Amethyst... Many IDEs are also available to pack AJAX and Flash into AIR (e.g, Aptana)	Definition of UI components and behaviour, definition and connection to data services, SWF generation. Server-side development not-covered.
Silverlight	Independent of server-side technology, Ms IIS for advanced features (e.g., WMS streaming)	Silverlight plug-in. WPF (.NET) for desktop RIAs	Declarative UI (XAML) / Imperative logic (multiple programming languages)	-	✓	**D**: data objects **L**: dependent of the language used **P**: UI description language, UI components and events C: SOA/REST, Windows Communication Foundation	Microsoft Expression Studio, Blend and Visual Studio. Eclipse4SL	Three roles collaborate, each using its specific tools: a UI designer, an integrator, and a developer. The developer has responsibility over data retrieval from local or remote sources. Server-side development not-covered.
JavaFX	In principle independent of server-side technology, clear tie with Java servers	JVM with JavaFX support	Declarative UI / Imperative logic (Java)	-	✓	**D**: data objects **L**: Java libraries **P**: UI components and events **C**: JNI, SOA, REST, RMI EXEC, TCP	Netbeans or Eclipse	A mix between Java traditional desktop application and Java Web Start

Table 3: Most adopted RIA development frameworks

b When using Adobe LiveCycle Data Services (LCDS) [DL-5]

The last aspect is recognized and addressed especially by runtime environment vendors that propose tools and processes (e.g., Adobe Catalyst [DL-1], Ms Expression Blend [DL-17]) targeted to different roles involved in the Web development process (designers, developers) and supporting the integration of their work. Most frameworks are designed to automate or simplify the coding of the application on the client-side (rightfully considered the novelty in RIAs), but very little or no support is given to the design and implementation of the client-server communication, as well as the server-side implementation. This is somehow justified by the widespread adoption of SOA, thus on the assumption that the same services designed for other aims (e.g., service composition) are appropriate or can be easily adapted to RIA UI needs.

Albeit their limited industrial adoption, *model-driven methodologies* aim towards a more comprehensive approach, covering more application life-cycle phases (e.g., design, evolution) as well as more application layers (client, server, and communication) through abstractions. The next section provides an overview of model-driven solutions for RIA development.

5 Model-driven development methodologies

In [12] we recognized the need for systematic methodologies to develop RIAs. Since then, several proposals have been designed adopting model-driven development (MDD) methodologies for RIAs [16], both from academia and from software vendors.

Depending on their origins, these approaches can be categorized into:

1. Research contributions coming from the Web Engineering community, stemming as evolution of model-driven approaches conceived for the design and development of traditional Web applications: they include WebML-RIA [1][18], OOHDM for RIA [13], OOH4RIA [11][19], and UWE for RIA [5].
2. Systematic development approaches from the Human Computer Interaction (HCI) community: RIA design is the focus of the RUX-Method [8] and can be achieved also with the more general UsiXML approach [10].
3. Approaches mixing HCI and Web Engineering techniques: the UML-based approach presented in [3], OOWS for RIA [15].
4. Recent proposal from tool vendors adopting MDD: WebRatio [DL-27], Mendix [DL-16], Novulo [DL-19], RUX-Tool [DL-24] and Thinkwise [DL-25].

		Technology		Language				Process	
					Scope				
		Server-side tech dependence	Client-side tech dependence	Style	Server	Client	Abstractions for RIA features	Tool support	Typical development process
Web Engineering	WebML-RIA	Independent	Independent (technologies hidden in the code generator)	Declarative (Visual DSL)	✓	Architectural issues (client data, client business logic and C/S comm.)	Client/server annotations on data and business logic models; event handling high-level primitives	WebML Prototype on top of WebRatio. Design + automatic code generator (OpenLaszlo)	Back-to-front (data-oriented)
	OOHDM for RIA	Independent	Independent (technologies hidden in the code generator)	Declarative (Object-oriented notations + specific notation for UI objects)	✓	Presentation and behavioral issues	Extension of ADVCharts behavioral presentation model to capture rich interaction styles and event handling	Prototype Under development	Back-to-front (object-oriented)
	OOH4RIA	Independent	Platform specific presentation design (exemplified in GWT)	Declarative (UML-based notations)	✓	Presentation and behavioral issues	Introduction of a widget-based presentation model and of an orchestration model for event handling	OOH4RIA Prototype [DL-21]. Design + QVT model-to-model and model-to-text transformations (GWT)	Back-to-front (object-oriented)
	UWE for RIA	Independent	Independent (technologies hidden in the code generator)	Declarative (UML-based notations)	✓	Presentation and behavioral issues	UML state machines for modeling RIA patterns – independent of modeling language	MagicUWE [DL-15] Prototype under development	Back-to-front (object-oriented)
HCI	RUX-Method	N/A	Independent abstract models + platform specific concrete models	Declarative (Visual DSL)	-	Presentation issues	Abstract concepts for content and containers; visual options for layout, user interaction, event handling, temporal behaviors	RUX-Tool. Design + model-to-text transformations (OpenLaszlo, Ajax, Flex)	Back-to-front (component-oriented) + User-Centered Design
	UsiXML	N/A	Independent abstract models + platform specific concrete models	Declarative (XML Description language)	-	Presentation issues	Introduction of RIA widgets at the concrete level	Set of UsiXML tools. Design + XSLT model transformations (to XAML)	User-Centered Design
Web /HCI	UML-based	Independent	Independent	Declarative (UML-based notations)	-	Presentation and behavioral issues	None	None	User-Centered Design
	OOWS for RIA	Independent	Independent	Declarative (UML metamodel and CTT model)	-	Presentation and behavioral issues	OOWS [DL-20] RIA support Transformations and automatic code generator	Olivanova tool support. Transformations and automatic code generator (Flex)	User-Centered Design
	WebRatio	Independent	Currently applied only to AJAX	Declarative (Visual DSL)	✓	Presentation and behavioral issues	WebML RIA support	WebRatio. Design + automatic code generator (AJAX)	Back-to-front (data-oriented)
	RUX-Tool	Independent	Independent	Declarative (Visual DSL)	✓	Presentation issues	RUX-Method RIA support	RUX-Tool. Design + automatic code generator (for Open Laszlo, Flex, AJAX)	User-Centered Design (presentation components-oriented)
	Mendix	Independent	Dependent (AJAX) Model is interpreted, no code generation	Declarative (Visual DSL)	✓	Presentation issues	Rich Forms DSL	Mendix Business Modeler. Design + automatic code generator (AJAX)	Back-to-front (Service-Oriented Business Applications)
	Novulo	Dependent (based on .Net)	Dependent (AJAX)	Declarative (Visual DSL)	✓	Presentation issues	N/A	Novulo Architect and Application Server. Design + automatic code generator (AJAX)	User-Centered Design (workflows & forms-oriented)
	Thinkwise Software Factory	Independent	Independent	Declarative (Visual DSL)	✓	Presentation and behavioral issues	UI description language	Thinkwise Software Factory. Design + automatic code generator (different technologies, including AJAX)	User-Centered Design (object-oriented)
	Agile Platform	Independent	AJAX-based	Declarative (Visual DSL)	✓	Presentation and behavioral issues	AJAX-based patterns integrated in the visual models	ServiceStudio. Design + automatic code generator (AJAX)	Back-to-front (data + process + screen design)

Table 4. MDD methodologies for RIAs, clustered into Web Engineering, HCI approaches and commercial tools

The evaluation of the MDD approaches is summarized in Table 4. The table shows that they share the following characteristics:

1) From a technology point of view, the design of RIAs can abstract both from server-side and client-side technologies, thus reaching also *client-side technology independence*. In particular, the binding with the specific client-side technology is moved either to the code generation phase or to presentation models including technology-specific primitives.
2) Considering the language dimension, *visual declarative notations* based on Domain Specific Languages (DSLs) or on UML are used to model the structure and the behavior of the UI and possibly the back-end of the application.
3) From a process point of view, *code can be automatically generated* starting from design specifications.

The development process in MDD approaches is *incremental and iterative* with *back-to-front* design. The typical steps include: requirement analysis, design (from data to business logic to presentation and communication design), implementation, testing/evaluation, maintenance.

Methods concentrating on UI issues are adopting *user-centered design* (UCD), which is still based on iterative design principles. The main focus is on users, directly involved in the process, and user tasks, to guarantee usable and effective artifacts. The UI can be designed first, so that user feedbacks can be incorporated in the earliest phases.

The high-level design of MDD approaches combined with the possibility of automatically generating the code presents several advantages:

1) Models can be used in all the phases of the project as (always updated) documentation,
2) Agile processes with rapid prototyping can be easily supported,
3) Developed components implementing high-level primitives can be reused across different applications.

Nevertheless, MDD solutions also present some drawbacks:

1. The independence of RIA technologies implies that in general they *do not cover all the features* offered by RIA technologies. This implies for instance that if an application requires a *functionality that is not directly expressible* through an MDD approach primitive, the designer has to go through the extension of the MDD language and code generator: this has the advantage of preserving the virtuous circle of high level-modeling and code generation through the whole application lifecycle, but presents the shortcoming of requiring *additional efforts and knowledge* with respect to simply writing executable code. Figure 1 illustrates the coverage of the different MDD proposals with respect to the four main categories of RIA features introduced in Section 2.
2. When the RIA interface to generate is particularly complex (e.g., cannot be expressed using the primitives of the modelling language, contains animations, etc.), although generally

discouraged in MDD, the case may require generated code to be manually manipulated. This can cause a lack of synchronization between models and code.

3. MDD, like frameworks, does not solve the issue of collaboration among designer and developer roles.

The primitives introduced by the MDD approaches to capture the RIA features can be specified by means of different models and at different levels of abstraction, which are worth being briefly described:

***Data* features** are addressed only by the WebML proposal: the specification consists of a data model represented as an Entity-Relationship model refined with annotations on the entities and relationships to express where data must be located (on the client or on the server) and its persistence level (temporary or persistent). By combining these two aspects, it is possible to specify which data are stored for disconnected functioning, which are temporarily stored on the client for client-side manipulation, which data remain only on the server and so on.

***Business logic* features** are addressed in two different ways:

- By explicitly specifying the *computation distribution between the server and the client* in the business logic model. This is the solution adopted in the WebML proposal, where the model expressing the business logic of a traditional Web application is enriched with annotations denoting the layer of computation of each primitive (an area of the page, an entire component, part of the query performed by a component, etc. can be executed entirely or partially on the server or on the client [1]). By annotating business logic operations as "client", it is possible to specify that navigation/filtering/sorting/validation of data are performed on the client; by denoting primitives as "server", the designer specifies which parts of the application are computed by the server.
- By *superimposing the UI model* (representing the client) *to the business logic model* (representing the server), and by associating UI elements with business logic elements to *trigger* server-side operations. This is the solution adopted for example in OOHDM, OOH4RIA, and RUX-Method.

***(Rich) presentation* features** are the focal aspect of most of the proposals. According to [2] user interface development is one of the most time-consuming parts of application development, testing, and maintenance: due to the nature of RIAs, this issue becomes more complex than in traditional Web applications. It involves several aspects:

- *Partial page refresh* behaviors: this aspect is explicitly treated in WebML, OOH4RIA, and RUX-Method where a *dynamic model* is introduced to address the computation of the Web page, where the parts to be refreshed/reloaded are specified.
- *Client-side interaction and RIA widgets*: this aspect is treated by the different proposals with different levels of abstraction as explained next.

UWE specifies *patterns* by means of UML state machines modeling the interaction, functionality, and presentation of typical RIA widgets (e.g., auto-completion of fields, periodic/dynamic refresh). Patterns are independent of the technology and also of the adopted model; they can be integrated into existing models by means of transformation rules that add the desired behavior to the model elements.

OOHDM abstracts from the technology, by extending the presentation model of a traditional Web application with Statecharts that express *UI transformations as the result of user interaction*; they allow the specification of sophisticated navigational behaviors (e.g., which objects are shown or hidden in the interface, how information expands or collapses, etc).

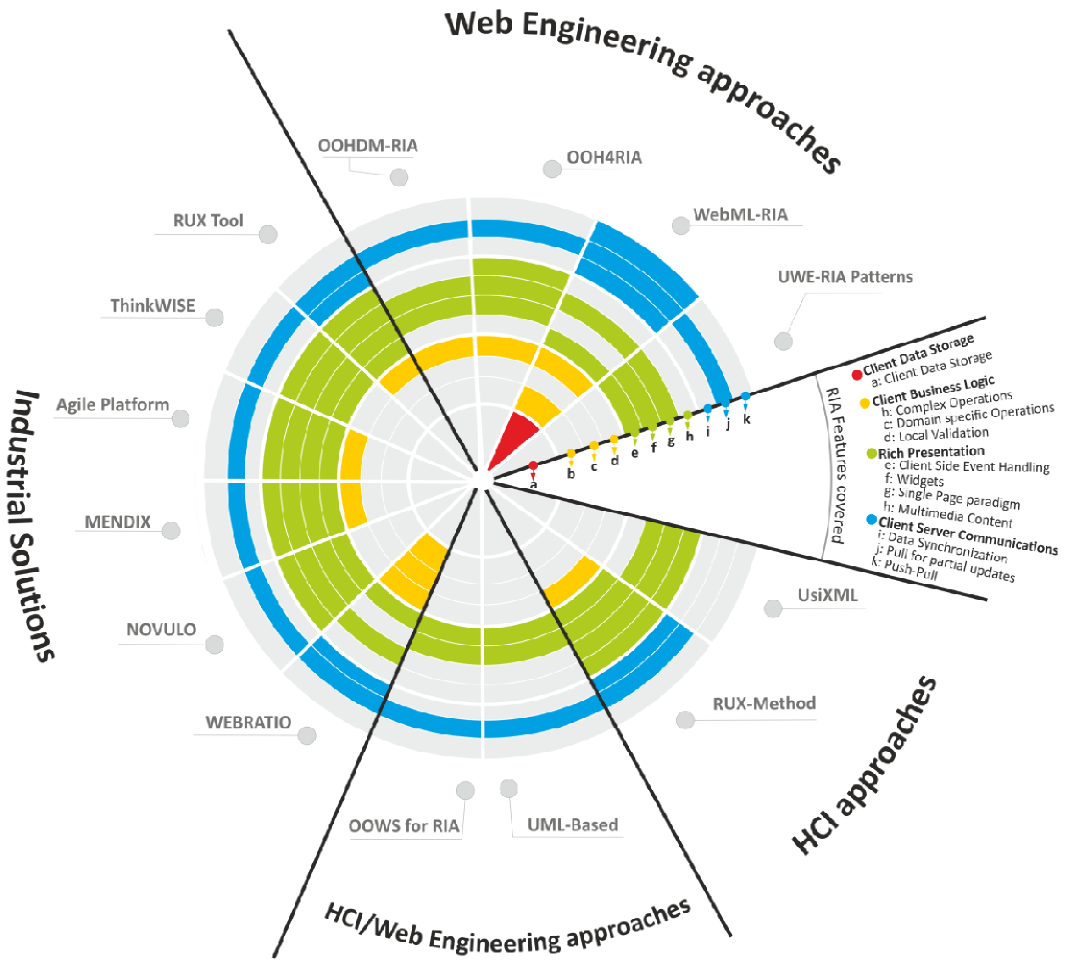


Figure 1: MDD approaches and coverage of RIA features

OOH4RIA proposes a new presentation meta-model at a lower level of abstraction, where the central elements are represented by the *widgets* provided by a specific platform (in this case, Google Web Toolkit): this meta-model allows the specification of the structural aspects of a RIA. Widgets can be combined, extended, customized, and bound to the other models. Work is ongoing to define a more abstract meta-model to represent generic widgets for generating any RIA framework, and not only GWT.

UsiXML applies to general UIs, not only to Web applications: it is an *XML-compliant markup language* that describes the UI for multiple contexts of use, including also RIA widgets. However, only the presentation aspects can be described and cannot be bound to business logic or data aspects.

The RUX-Method operates at three different levels of presentation (Abstract, Concrete and Final presentation levels), which include also *temporal and interaction behaviors*. Following Event-Condition-Action rules, operations provided by other models (e.g. UWE, WebML) can be triggered. RUX-Method specifies a component library able to specify the relationships between *widgets* of different complexity levels (e.g., mashups, windows and so on) and to ensure their right composition in the UI.

The approaches mixing HCI and Web Engineering techniques focus on users tasks and on user *interaction*, modeled as interaction patterns (both at an abstract and at a concrete level) or as UML-Statecharts, respectively.

Commercial tools provide simple solutions for RIA presentation. Novulo defines the UI by means of its IDE; Thinkwise uses a UI description language; Mendix supports a DSL for the specification of rich forms, and the Agile Platform [DL-6] contains a set of rich-Web usability patterns (e.g., user input validation, patterns involving asynchronous HTTP requests). In WebRatio, the interface elements like links, form fields, and pages can be associated with properties defining rich behaviors, like partial page refresh, drag&drops, dynamic tooltips, field content autocompletion, etc.: the corresponding code is automatically derived from the model enriched with such properties. Finally, RUX-Tool implements the specification of the RUX-Method.

Communication **features** are captured by means of *event-handling*, which may occur either

- at the business logic level, to generate, synchronously or asynchronously notify, and detect *events between the client and the server* (e.g., in WebML) or
- at the presentation level, where *UI or system events* can be *associated with actions* to be executed on the server-side (e.g., in OOHDM and OOH4RIA) and where the designer can decide between synchronous and asynchronous event handling (e.g., in the RUX-Method).

Figure 1 graphically summarizes all such aspects: currently, none of the proposed methodologies covers all the RIA features. However, it can be noticed that almost all the typical features of RIAs can be specified using conceptual models and, therefore, the different MDD methodologies can however support the development of the different classes of applications introduced in Section 2:

- A RIA makeover can be easily done with all the approaches focusing on presentation.
- Rich UIs can be designed with RUX-Method, OOHDM, and OOH4RIA.

- The main features of distributed and standalone RIA applications can be specified with the WebML methodology.

More complex RIAs covering all the categories can be obtained by combining different proposals: for example, in [7] the combination of WebRatio and RUX-Tool is illustrated.

6 Discussion and conclusions

Both framework-based and model-driven approaches present some limitations. Figure 2 visually compares the RIA development frameworks of Section 4 with the MDD development approaches of Section 5.

The current industrial development practice and MDD solutions lack a *comprehensive approach to RIA development*, covering all development steps and application layers (server and client), easing the process, and helping identifying correct design choices.

The severe fragmentation of RIA technologies encourages non-structured ad-hoc development, preventing the adoption of a common set of guidelines and best practices. Also, MDD approaches allow to declaratively model at a high-level of abstraction several RIA features and in many cases to automatically generate the final code, but each methodology focuses only on specific aspects.

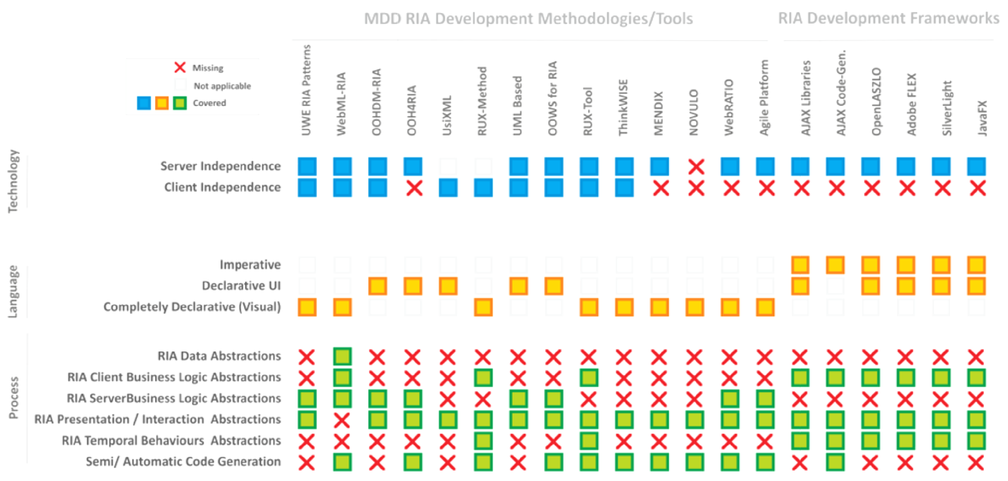
Given these shortcomings, we identified the following main future research directions for RIAs:

1. **Life-cycle:** RIAs impact on the *whole life-cycle* of the development process, but both frameworks and model-driven approaches have focused their attention mainly on the design and implementation phases. Important phases like requirement analysis and testing need to be revised w.r.t. traditional Web applications. RIAs pose new challenges both concerning functional (dynamic UIs, mark-up and scripting, browser idiosyncrasies) and non-functional testing (how and where to measure UI responsiveness [9]).
2. **Design methodology:** A *complete design methodology* covering all RIA dimensions (data, business logic, presentation, communication) is still missing. Such a methodology should be distilled as a combination of the best practices in the industrial world and the choices identified at an abstract level in the design of MDD approaches. MDD solutions cover complementary aspects and should help in the early identification of the possible design alternatives and the impact of a choice on the different aspects of the application. For instance, both *server- and client sides* should be seamlessly *co-designed*: code-based and framework-driven developments induce a fracture between client-side and server-side development[c].
3. **Roles:** In RIAs the UI is the main responsible for the user experience and therefore it plays a pivotal role. Specific methodologies such as *User-Centered Design* become important; however, they still need to be adapted to the RIA development process by focusing on the inclusion and integration of the different tasks and professional figures involved in producing the final application user experience (designers, client- and server-side developers, system administrators). Also, accessibility guidelines such as WAI-ARIA[d] should be supported by

c An exception is GWT providing means for the generation of (at least part of) the server-side application logic.
d http://www.w3.org/WAI/intro/aria

development approaches. Currently, only some efforts have been proposed by some MDD approaches such as [6].

4. **Tool support:** current tools can be improved to reach technology independence, to cover the complete application life-cycle, to leverage on high level primitives and code-generation. Furthermore, they should natively ease the integration of different roles in the development process.



✗ Missing
Not applicable
🟦 🟨 🟩 Covered

		MDD RIA Development Methodologies/Tools														RIA Development Frameworks					
		UWE RIA Patterns	WebML-RIA	OOHDM-RIA	OOH4RIA	UsiXML	RUX-Method	UML Based	OOWS for RIA	RUX-Tool	ThinkWISE	MENDIX	NOVULO	WebRATIO	Agile Platform	AJAX Libraries	AJAX Code-Gen.	OpenLASZLO	Adobe FLEX	SilverLight	JavaFX
Technology	Server Independence	🟦	🟦	🟦	🟦			🟦	🟦	🟦	🟦	🟦	✗	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
	Client Independence	🟦	🟦	🟦	✗	🟦	🟦	🟦	🟦	🟦	🟦	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Language	Imperative															🟨	🟨	🟨	🟨	🟨	🟨
	Declarative UI			🟨	🟨	🟨		🟨	🟨							🟨		🟨	🟨	🟨	🟨
	Completely Declarative (Visual)	🟨	🟨				🟨			🟨	🟨	🟨	🟨	🟨	🟨						
Process	RIA Data Abstractions	✗	🟩	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	RIA Client Business Logic Abstractions	✗	🟩	✗	✗	✗	🟩	✗	✗	🟩	✗	✗	✗	✗	✗	🟩	🟩	🟩	🟩	🟩	🟩
	RIA ServerBusiness Logic Abstractions	🟩	🟩	🟩	🟩	✗	✗	🟩	🟩	✗	✗	✗	✗	🟩	🟩	✗	✗	✗	✗	✗	✗
	RIA Presentation / Interaction Abstractions	🟩	✗	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
	RIA Temporal Behaviours Abstractions	✗	✗	✗	✗	✗	🟩	✗	✗	🟩	✗	✗	✗	✗	✗	🟩	🟩	🟩	🟩	🟩	🟩
	Semi/ Automatic Code Generation	✗	🟩	✗	🟩	✗	🟩	✗	🟩	🟩	🟩	🟩	🟩	🟩	🟩	✗	🟩	✗	✗	✗	✗

Figure 2: Graphical summary of the aspects covered by RIA development approaches

We can affirm that MDD approaches can play a major role in challenging the complexity of RIA development. Indeed, current trends in the software and Web engineering fields show an increasing adoption of MDD and MDA [4] (e.g., Eclipse has joined MDA, Visual Studio partially adopts MDD) with the aim of streamlining the development process and improving final software quality.

The *XMLHTTPRequest* [DL-26] has got the W3C recommendation status in 2010, meanwhile the HTML5 specification - expected to be completed by the end of 2011 - will have a major impact on RIA development in terms of technologies and, therefore, processes. Browsers supporting HTML5 will natively provide most of the features that now require plug-ins (namely multimedia and client-side storage support); in addition, more homogeneous JavaScript support across browsers is expected. We believe this will result in less effort to be invested in technology-specific issues (cross-browser JS frameworks, plug-in-specific coding) and more into final application and development process quality. For these reasons, the research directions we identified above remain valid, as they concentrate on high-level primitives, models, and processes rather than implementation technologies.

Acknowledgements.

The authors wish to acknowledge the collaborative funding support from the Spanish projects: PDT09A015 granted by Junta de Extremadura and FEDER and TIN2008-02985 granted by Ministerio de Ciencia e Innovación and FEDER.

References

[1]. A. Bozzon, S. Comai, P. Fraternali, G. Toffetti Carughi, Conceptual Modeling and Code Generation for Rich Internet Applications, in: International Conference on Web Engineering (ICWE), ACM 263, Palo Alto, USA, 2006, pp. 353-360.

[2]. F. Daniel, J. Yu, B. Benatallah, F. Casati, M. Matera, R. Saint-Paul, Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities, IEEE Internet Computing 11 (3) (2007) 59-66.

[3]. P. Dolog, J. Stage, Designing Interaction Spaces for Rich Internet Applications with UML, in: International Conference on Web Engineering (ICWE), Springer LNCS 4607, 2007, Como, Italy, pp. 358-363.

[4]. B. Hailpern, P. Tarr, Model-driven development: the good, the bad, and the ugly, IBM Systems Journal 45 (3) (2006) 451-461.

[5]. N. Koch, M. Pigerl, G. Zhang, T. Morozova, Patterns for the Model-Based Development of RIAs, in: International Conference on Web Engineering (ICWE), Springer LNCS 5648, San Sebastián, Spain, 2009, pp. 283-291.

[6]. M. Linaje, A. Lozano-Tello, J.C. Preciado, F. Sanchez-Figueroa, R. Rodríguez, Obtaining accessible RIA UIs by combining RUX-Method and SAW, in: Automated Specification and Verification of Web Systems, Hagenberg, Austria, 2009, pp. 85-98.

[7]. M. Linaje, J.C. Preciado, R. Morales-Chaparro, R. Rodríguez-Echeverría, F. Sanchez-Figueroa, Automatic Generation of RIAs Using RUX-Tool and WebRatio, in: International Conference on Web Engineering (ICWE), Springer LNCS 5648, San Sebastián, Spain, 2009, pp. 501-504.

[8]. M. Linaje, J.C. Preciado, F. Sanchez-Figueroa, Engineering Rich Internet Application User Interfaces over Legacy Web Models, IEEE Internet Computing 11 (6) (2007), 53-59.

[9]. C. Loosley, Rich Internet Applications: Design, Measurement, and Management Challenges, white paper, Keynote Systems (2006). <http://www.keynote.com/docs/whitepapers/RichInternet_5.pdf>

[10]. F. Martinez-Ruiz, J. Muñoz Arteaga, J. Vanderdonckt, J. Gonzalez-Calleros, R. Mendoza, A first draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications, in: Latin American Web Congress (LA-Web), Puebla, Mexico, 2006, pp. 32-38.

[11]. S. Meliá, J. Gómez, S. Pérez, O. Díaz, A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA, in: International Conference on Web Engineering (ICWE), New York, USA, IEEE Computer Society, 2008, pp. 13-23.

[12]. J.C. Preciado, M. Linaje, F. Sanchez, S. Comai, Necessity of methodologies to model rich Internet applications, in: International Symposium on Web Site Evolution (WSE), IEEE Computer Society, Budapest, Hungary, 2007, pp. 7-13.

[13]. M. Urbieta, G. Rossi, J. Ginzburg, D. Schwabe, Designing the Interface of Rich Internet Applications, in: Latin American Web Congress (LA-Web), Santiago de Chile, Chile, 2007, pp. 144-153.

[14]. R. Valdes, E. Knipp, D. Mitchell Smith, G. Phifer, M. Driver, Market Scope for Ajax Technologies and Rich Internet Application Platforms, Gartner RAS Core Research Note G00173751, 2009. <http://www.adobe.com/enterprise/pdfs/gartner-ajax-ria.pdf>

[15]. F. Valverde, O. Pastor,,P. Valderas, V. Pelechano, A Model-Driven Engineering Approach for Defining Rich Internet Applications: A Web 2.0 Case Study, Handbook of research on Web 2.0, 3.0 and X.0 Technologies, Business, and Social Applications, IGI Global, 2009, pp. 40-58.

[16]. J. Wright, J. Dietrich, Survey of existing languages to model interactive web applications, in: Asia-Pacific Conference on Conceptual Modelling vol. 79, Wollongong, Australia, 2008, pp. 113-123.

[17]. J. Wright, J. Dietrich, Requirements for Rich Internet Application Design Methodologies, in: International Conference on Web Information Systems Engineering (WISE), Auckland, New Zealand, 2008, pp. 106-119.

[18]. P. Fraternali, S. Comai, A. Bozzon, G. Toffetti. Engineering rich internet applications with a model-driven approach. ACM Trans. Web 4, 2, Article 7 (April 2010), 47 pages.

[19]. S. Meliá, J. Gómez, S. Pérez, O. Díaz. Architectural and Technological Variability in Rich Internet Applications. IEEE Internet Computing 14, 3 (May 2010), 24-32.

[20]. P. Fraternali, G. Rossi, F. Sánchez-Figueroa, "Rich Internet Applications," IEEE Internet Computing, vol. 14, no. 3, (May 2010) pp. 9-12.

Appendix 1: DL-Development Links

[DL-1]	Adobe Catalyst:	http://labs.adobe.com/technologies/flashcatalyst/
[DL-2]	Adobe AIR:	http://www.adobe.com/products/air/
[DL-3]	Adobe FlashPro:	http://www.adobe.com/products/flash/flashpro/
[DL-4]	Adobe Flex:	http://www.adobe.com/products/flex/
[DL-5]	Adobe LiveCycle Data Services:	http://www.adobe.com/devnet/livecycle/dataservices.html
[DL-6]	AgilePlatform:	http://www.outsystems.com/
[DL-7]	Backbase:	http://www.backbase.com/
[DL-8]	Dojo Toolkit:	http://dojotoolkit.org/
[DL-9]	Flash Player:	http://www.adobe.com/support/flashplayer/
[DL-10]	GWT:	http://code.google.com/webtoolkit/
[DL-11]	HTML5:	http://www.w3.org/TR/html5/
[DL-12]	JavaFX	http://sun.com/javafx/
[DL-13]	Java Web Start:	http://java.sun.com/products/javawebstart/
[DL-14]	JQuery:	http://jquery.com/
[DL-15]	MagicUWE:	http://uwe.pst.ifi.lmu.de/toolMagicUWE.html
[DL-16]	Mendix:	http://www.mendix.com/
[DL-17]	Ms Blend:	http://www.microsoft.com/expression/products/Blend_Overview.aspx
[DL-18]	Ms Silverlight:	http://silverlight.net/
[DL-19]	Novulo:	http://www.novulo.com/
[DL-20]	Olivanova:	http://www.care-t.com/products/index.asp
[DL-21]	OOH4RIA:	http://www.dlsi.ua.es/~santi/ooh4ria
[DL-22]	OpenLaszlo:	http://www.openlaszlo.org/
[DL-23]	Prototype:	http://prototypejs.org/
[DL-24]	RUX-Tool:	http://www.homeria.com/
[DL-25]	Thinkwise:	http://www.thinkwisesoftware.com/
[DL-26]	XMLHttpRequest:	http://www.w3.org/TR/XMLHttpRequest/
[DL-27]	WebRatio:	http://www.webratio.com/