



Rich Internet Applications

The original World Wide Web was a platform for accessing static or dynamic content encoded in hypertext markup language. User interaction was limited to navigating links and entering data in forms. This thin-client architecture was simple and universal (no client installation required) but severely limited the quality of the applications that could be delivered over the Internet. Early attempts at extending interface functionality (such as Java applets and client-side scripting) enriched HTML-based navigation with interactive objects, animated presentation effects, and input validation. However, these features' diffusion was limited by standardization issues (for example, proliferation of client-side scripting languages) and architectural issues (such as firewall incompatibility).

The intervening years have seen much progress in increasing the functionality of tasks performed through the Web, with a corresponding increase in the complexity of their creation. Modern Web solutions resemble desktop applications, enabling sophisticated user interactions, client-side processing, asynchronous communications, and

multimedia. The “network as platform computing” idea, strengthened by Web 2.0's emergence, has accentuated HTML/HTTP's limits.

Web 2.0 is built around user-centric applications (for instance, social networks or user-generated content management solutions) that demand a high degree of usability and powerful interactions. A pure HTTP/HTML architecture fails to support the required capabilities in several respects: presentation, where only the interaction widgets and containers predefined in HTML are available; communication, which supports only the synchronous interaction native in HTTP and requires that callback mechanisms for asynchronous behavior either be simulated on top of the HTTP client-response cycle or implemented at a low level in TCP/IP; business logic, which occurs mostly at the server side; and data management, where only limited client-side data storage capabilities are available (such as using cookies). More and more, post-HTML/HTTP technologies are shaping the Web, among which *rich Internet applications* (RIAs) play a prominent role.

Piero Fraternali
Politecnico di Milano

Gustavo Rossi
Universidad Nacional de La Plata

Fernando Sánchez-Figueroa
Universidad de Extremadura

RIAs

The term RIA refers to a heterogeneous family of solutions, characterized by a common goal of adding new capabilities to the conventional hypertext-based Web. RIAs combine the Web's lightweight distribution architecture with desktop applications' interface interactivity and computation power, and the resulting combination improves all the elements of a Web application (data, business logic, communication, and presentation).

The original Web kept data (state) on a server, and the client explicitly downloaded information when needed. RIA technologies support client-side storage in a way that depends on the specific technology and device. For example, clients can locally store the shopping cart in an e-commerce application or an appointment calendar while users are manipulating this data. In the original Web, the server also performed business logic. RIA technologies enable moving part of the computation to the client. Offloading computation to the client allows quicker response and optimizes communication costs. For example, in a shopping cart or calendar solution, users can navigate, filter, and manipulate the data using complex operations before sending it to the server. The original Web was a request-response machine: the server sent information only in response to a client request. In RIA, both the client and server can initiate communication; program elements in the client stand ready to receive and execute asynchronous server commands. This bidirectionality eliminates many unnecessary server roundtrips typical of thin-client applications.

Most of all, RIAs' popularity owes much to their powerful presentation and interaction capabilities. A traditional Web application interface consists of multiple pages, refreshed at each user's interaction. In RIAs, the interface can be a single page comprising subpages that manages all the user's interactions, like in a desktop application. This paradigm avoids full-page refreshes at each interaction and permits applications to independently load, display, and update individual page elements. Applications can also extend the interface dynamically by loading parts of the presentation logic (for example, JavaScript code) and interface elements at runtime. Developers can define extended interaction events (such as event listeners and handlers) and temporal behaviors

(for instance, animations based on temporal relationships among interface components) to further improve the user experience. Because an application can download new behaviors dynamically, it isn't restricted to a small set of HTML-predefined standard controls and containers. For example, an RIA application can dynamically customize existing widgets, adapting them to specific usage contexts — such as to the display device's rendering capacity.

Many applications can reap RIAs' benefits: product/service selection, configuration and customization, workflow-driven applications, education, entertainment, collaborative work, and more. At present, Web applications such as Google Maps, YouTube, and Flickr exploit the large spectrum of possibilities RIAs offer.

We can implement RIAs with several different technologies. Most implementation technologies are invoked through a browser and might rely on the Web's inherent facilities or supplement conventional browsers with scripts or plug-ins. We can also execute an RIA outside a browser: with scripting-based technologies such as Ajax, Web pages can include programs (scripts) that reference presentation elements, respond to user-interface gestures, and have their own storage. Modern browsers understand these scripting languages and can interpret and effect these scripts. Plug-in solutions include Flash (www.adobe.com/products/flash), OpenLaszlo (www.openlaszlo.org), and Flex (www.adobe.com/products/flex). Plug-ins provide better performance than JavaScript because they run their own native code, allowing advanced rendering and event processing.

Browser-based approaches, such as Mozilla XUL (www.mozilla.org/projects/xul), support a rich interaction natively, without the need for proprietary extensions to the browser. However, this type of solution is hindered by its browser-dependent nature (for example, applications based on XUL might be inaccessible to users with other browsers). Finally, we can also execute RIAs outside the browser, using a specific runtime environment, as with AIR (www.adobe.com/products/air) and JavaFX (<http://sun.com/javafx>). Here, the user must install additional software, but the capabilities regarding client-side storage and offline use improve.

These technologies' success has accelerated RIA adoption. However, as in every fast-growing technological sector, development practices

run ahead of tools and methodologies. This gap provides research opportunities.

Research Issues

RIAs' advent creates an articulated research landscape with features that include the language and architectural standards used to develop RIAs, the software frameworks built on top of these standards that enhance development productivity and solution quality, and the development tools and methodologies backing the RIA life cycle's development activities.

At the language and architecture level, the W3C is considering many issues that RIA technologies have put forth for consolidation in the forthcoming HTML5 standard, now in a draft version (see <http://dev.w3.org/html5/spec/Overview.html>). Relevant research issues span such diverse aspects as the new language elements' syntax and semantics, backward compatibility, performance of parsing and rendering engines, compilation of declarative specifications into imperative languages, cache architectures, security, accessibility, and more. For HTML5 to assume a central role in RIA development, the research community must conduct a focused review effort to ensure that the complex set of features embodying the new language have internal coherence and meet RIA developers' requirements.

Developers use languages and architectures to build software frameworks, which, if well-constructed, help them adopt good design practices and enhance development quality. At present, RIA frameworks are a hot topic, attracting much effort. Examples include Backbone (www.backbase.com), Dojo (<http://dojotoolkit.org>), and Rico (<http://openrico.org>). These frameworks automate the most tedious programming tasks for the client or server side via specific primitives, libraries, or code-generation techniques, and provide reusable patterns for accelerating application design. A key research issue for such frameworks is the automatic and dynamic allocation of data and computation between the client and server, based not only on static application requirements but also current processing loads.

Much of the progress in building software systems comes from tools that automate the implementation of common behaviors. However, each additional automation potentially increases system complexity. Such complexity

is contained only through principled development methodologies. A bitter lesson from software engineering is that development without principled approaches becomes too complicated, expensive, and unmaintainable. Internet applications' strong interdisciplinary nature exacerbates these problems.¹ Systematic development methods improve productivity, reduce effort and costs, improve maintenance and evolution, and make applications less error-prone. This is particularly important when a new development paradigm emerges: the involved technologies' variety and complexity and the wide range of new capabilities demand a rethinking of the Web engineering process.

Recently, the Web engineering community has advocated adopting the model-driven development (MDD) paradigm for RIAs.² MDD refers to a family of development approaches based on using models as a primary artifact in the development life cycle. Engineers use models for a range of tasks: requirement specification, validation, code generation, testing, size estimations, and more. Researchers have extended numerous existing methodologies, originally conceived for traditional Web applications, to cope with the new modeling issues appearing in RIAs.³⁻⁶ However, this extension is far from trivial: RIAs incorporate many novel features, such as presentation behaviors, data and processing distribution, flexible event handling, and communication. Consequently, RIA models can quickly grow too complex for developers to understand or manage. Furthermore, pre-RIA Web engineering tools' model-checking and code-generation capabilities have yet to be fully demonstrated for RIAs. The research agenda at this level is broad and comprises such items as the mechanisms for designing expressive yet compact domain-specific languages and models for RIAs from orthogonal abstractions (for instance, aspect orientation), RIA semantics for model verification and support for testing, efficient and traceable model-transformation and code-generation techniques, and end-to-end methodologies for RIA development.

In this Issue

This special issue features two articles that address some of the research challenges we've highlighted. One focuses on language and architecture issues, whereas the other deals with the methodological principles at the base of an MDD approach to RIA development.

In “Compiler Transformations to Enable Synchronous Execution in an RIA Runtime,” Anantharaman P. Narayana Iyer, Arijit Chatterjee, and Jyoti Kishnani describe a technique based on compiler transformations that enables synchronous execution on an asynchronous RIA programming model. This solution simplifies the complexity of synchronizing multiple pieces of program logic when using asynchronous communication. The proposed method exploits a compiler to modify the RIA code using a block-chaining algorithm, wherein asynchronous functions are transformed into a sequence of synchronously executing blocks. The authors have implemented their solution using the Adobe Flash Player and Flex framework.

In “Architectural and Technological Variability in Rich Internet Applications,” Santiago Meliá, Jaime Gómez, Sandy Pérez, and Óscar Díaz advocate introducing architecture and technological concerns at an early stage of model-driven RIA development. To this end, they propose an RIA-specific architecture model that developers can use to represent an RIA architectural configuration. Their model enables model transformations that accelerate RIA implementation. This technology allows developers to choose, during the modeling phase, the right options from alternative RIA technological and architectural configurations. The authors have used their approach to extend OOH4RIA,⁴ an MDD methodology for RIAs.

The articles in this special issue touch on some of the many open issues that we must solve to make RIA engineering a fully industrial process. As their increasing diffusion demonstrates, RIAs are here to stay. It is our task to continue investigating their features and propose novel methods and tools to make their development more efficient. □


References

1. J.C. Preciado et al., “Necessity of Methodologies to Model Rich Internet Applications,” *Proc. IEEE Int’l Symp. Web Site Evolution*, IEEE CS Press, 2005, pp. 7–13.
2. N. Moreno, J.R. Romero, and A. Vallecillo, “An Overview of Model-Driven Web Engineering and the MDA,” *Web Engineering: Modelling and Implementing Web Applications*, G. Rossi et al., eds., Springer-Verlag, 2007, pp. 353–382.
3. A. Bozzon et al., “Conceptual Modeling and Code Generation for Rich Internet Applications,” *Proc. Int’l Conf. Web Engineering (ICWE 06)*, ACM Press, 2006, pp. 353–360.
4. S. Meliá et al., “A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA,” *Proc. Int’l Conf. Web Engineering (ICWE 08)*, IEEE CS Press, 2008, pp. 13–23.
5. M. Linaje, J.C. Preciado, and F. Sánchez-Figueroa, “Engineering Rich Internet Application User Interfaces over Legacy Web Models,” *IEEE Internet Computing*, vol. 11, no. 6, 2007, pp. 53–59.
6. M. Urbieto et al., “Designing the Interface of Rich Internet Applications,” *Proc. Latin-American Web Congress (LA-WEB 07)*, 2007, pp. 144–153.

Piero Fraternali is a full professor in the Dipartimento di Elettronica e Informazione at Politecnico di Milano. His main research interests concern database integrity, rule-based languages, methodologies and tools for Web application development, multimedia information retrieval, and search-based applications. Fraternali is a co-inventor of WebML (www.webml.org), a model for the conceptual design of Web applications, and cofounder of Web Models (www.webratio.com), a start-up focused on innovative Web development tools. Contact him at piero.fraternali@polimi.it.

Gustavo Rossi is full professor at Facultad de Informatica, Universidad Nacional de La Plata, Argentina, and director of LIFIA (the Research Laboratory in Advanced Informatics). He’s also a researcher at CONICET. His research includes agile and model-driven development of Web applications, Web model refactoring, and rich Internet applications. Rossi has a PhD in computer science from PUC-Rio, Brazil. He’s an editor of *Web Engineering: Modeling and Implementing Web Applications* (Springer, 2007). Contact him at gustavo@lifia.info.unlp.edu.ar.

Fernando Sánchez-Figueroa is a full professor in the Departamento de Ingeniería de Sistemas Informáticos y Telemáticos at Universidad de Extremadura. His main research interests concern rich Internet applications and user interfaces for people with disabilities. He’s cofounder of Homeria Open Solutions (www.homeria.com), a spin-off focused on novel and original Web development tools such as RUX-Tool, aimed for the design of RIA user interfaces. Contact him at fernando@unex.es.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.