Journal of Web Engineering, Vol. 10, No.3 (2011) 197-233
© Rinton Press

A NEW END-USER COMPOSITION MODEL TO EMPOWER KNOWLEDGE WORKERS TO DEVELOP RICH INTERNET APPLICATIONS

DAVID LIZCANO FERNANDO ALONSO JAVIER SORIANO GENOVEVA LÓPEZ

School of Computing, Universidad Politécnica de Madrid, Madrid, Spain

dlizcano@fi.upm.es falonso@fi.upm.es jsoriano@fi.upm.es glopez@fi.upm.es

Received July 15, 2011
Revised September 11, 2011

Enabling real end-user programming development is the next logical stage in the evolution of Internet-wide service-based applications. Even so, the vision of end users programming their own web-based solutions has not yet materialized. This will continue to be so unless both industry and the research community rise to the ambitious challenge of devising an end-to-end compositional model for developing a new age of end-user web application development tools. This paper describes a new composition model designed to empower programming-illiterate end users to create and share their own off-the-shelf rich Internet applications in a fully visual fashion. This paper presents the main insights and outcomes of our research and development efforts as part of a number of successful European Union research projects. A framework implementing this model was developed as part of the European Seventh Framework Programme FAST Project and the Spanish EzWeb Project and allowed us to validate the rationale behind our approach.

Key words: end-user programming; user-centred service-oriented architectures; rich internet application.
Communicated by: M. Gaedke & A. Ginige

1 Introduction

Since the year 2000, enterprises and private customers have been refusing to pay for standardized or commodity-type software products, and both the business-to-business (B2B) and business-to-customer (B2C) information technology economies turned to software as a service-based (SaaS) model [1] through service-oriented architectures (SOA) [2]. SOAs increase asset reuse, reduce integration expenses and improve the rate at which businesses can respond to new demands. The main idea is to allow enterprises and even customers to access, exploit and customize their own software solutions by composing and orchestrating heterogeneous Internet services within the future Internet of Services (IoS) [3]. SOA, however, was conceived in an enterprise world, fostering B2B aspects, like business governance or automatic mediation but overlooking human-computer interaction (HCI) issues, like user-service interaction, flexibility or legibility. Therefore, the application of SOA technologies partially covers enterprise demands, but does not meet user requirements and expectations. Not all end users are capable of orchestrating the services that they are obliged to use, and the development of highly usable applications that offer a similar user experience to their desktop counterparts and exploit these services is way beyond their capabilities. Users are looking for rich, effective applications fed by the services of the emerging IoS that are, whenever possible, also tailorable to their needs [4].

In response to these weaknesses, rich internet applications (RIAs) are swarming on to the Internet market. RIAs are more efficient applications that have a better response time and offer a better user experience than traditional web applications. They also provide users with a means of managing some of the services that they need [5]. This remedies a number of the shortcomings identified by users who increasingly view the web as a new platform on which to run many of the software applications they use every day [6]. On the downside, most RIAs are inflexible and are not end-user tailorable, confining end users to the role of mere application consumers. The main aim of Web 2.0 is to empower end users as the ultimate motivation, goal and motor of the Internet as a whole, and it is here that RIAs have still not achieved their full potential [7]. Web 2.0 should have evolved RIAs through the new concept of "prosumers": consumers of web services and applications that are at the same time developers capable of customizing their own products [6]. This idea would implement real end-user development [8], enabling users to tailor RIAs to their needs. Janner et al. [4] and Garrigos et al. [9] discussed this concept, indicating the importance of service front-ends for bridging the gap between the service technical layer and end users.

There are key proposals offering DIY (do-it-yourself) [10] guidance on evolving RIAs to meet end-user demands and requirements through mashups of interoperable and highly configurable visual interface elements that invoke back-end services from such user-centred interfaces as iGoogle (http://www.google.com/ig), Yahoo! Pipes (http://pipes.yahoo.com/), Yahoo! Dapper (http://open.dapper.net), Netvibes (http://www.netvibes.com), JackBe Presto Cloud (http://prestocloud.jackbe.com/), Microsoft Popfly (http://www.popfly.com), OpenKapow and Kapow Katalyst (www.kapowsoftware.com), AMICO (http://amico.sourceforge.net/), Marmite (http://www.cs.cmu.edu/~jasonh/projects/marmite/) or EzWeb (http://ezweb.morfeo-project.org/). These solutions encourage the practice of mashing up resource front-ends to simplify the exploitation and invocation by end users of all sorts of web services tailored to their context and knowledge [11].

The end-user development (EUD) model would be a compositional development model, where programming-illiterate users do the composing [8]. But this model has yet to be formalized and structured. The increasingly globalized and fierce competition within an ecosystem of EUD solutions has driven developers (Google, Yahoo!, Microsoft, Amazon, Apple, Sun, IBM, etc.) to develop their own tools without formalizing either their underlying component model or the common composition processes for their development. This has led to two problems. First, there is no common component model enabling interoperability between building blocks from different manufacturers. Second, there is no development model, or at least a set of heuristics, to guide end users through the process of building and/or tailoring RIAs. These failings, plus the fact that current approaches for supporting end-user web developments rely on heavyweight engineering skills [12], prevent programming-illiterate end users from being able to satisfactorily exploit existing EUD tools [3]. This gap can be filled by designing a whole new compositional model. This model should employ reusable and composable components focusing on and meeting end-user requirements and expectations [13] regarding the design of a RIA to deal with a real problem.

The purpose of the research reported here is to define a compositional model to support emerging end-user development. This model provides guidance for non-technical users on the composition of their own RIA based on user-centred resources. This compositional model brings together the success factors of existing solutions (such as web mashups, spreadsheets, and so on) and has been instantiated by an EUD tool. This tool, developed as part of two European R&D projects, has been used to check end-user satisfaction with the implemented compositional model and to conduct a preliminary empirical study on whether programming-illiterate users managed to build RIAs that met a series of specified requirements. If successful, this model would empower knowledge workers [14] to build their own RIAs, either individually or even as part of an open community of end users sharing their compositions [15, 16]. These RIAs would exploit end-user domain expertise in the performance and support of their daily knowledge work. By developing their own RIAs, programming-illiterate end users would be able to deal with more general routine problems and save problem-solving time and effort.

The remainder of the paper is structured as follows. Section 2 presents related work and the state of the art in EUD applied to RIAs. Section 3 explains the motivation and goals for building, formalizing and instantiating the new EUD compositional model as a particular tool. In Section 4 we discuss the compositional model requirements and several other properties that they should have. These requirements are tailored to the user-centred RIA development domain, stipulating specific requirements for generating the proposed compositional model. Next, we analyse the main parts of the proposed compositional model: Section 5 deals with the component model, Section 6 with the development model, and Section 7 with the composition languages. Section 8 then presents the results of a study that we conducted to test the EzWeb/FAST tool implementing the compositional model. This study was useful for evaluating how well the principles and ideas embodied in the compositional model lived up to real user expectations. Finally, Section 9 concludes this paper and presents a brief outline of future work.

2 Background

In their capacity of knowledge workers of a domain with which they are at ease and familiar [10], end users are perfectly clear about the requirements of the problem that they have to solve [14]. At present, however, they have to convey this knowledge/expertise to someone with programming skills that can create an application that meets the requirements. Consequently, there is often a gap between the target and real application, called the paradox of expertise [17]. Web 2.0 spawned the user-centred DIY philosophy, claiming that users should be able to be the providers and/or developers of the web applications that they consume. This approach proposed centring existing services and resources on end users so that they could do the development work, thereby cutting the development cost and time of their final solutions [14]. On top of this, Web 2.0 also offered new enabling technologies, such as AJAX, to lay the foundations for more usable applications that were more appealing to end users, like RIAs [18].

This whole philosophy had been coined years earlier as end-user development [19]. This time, however, it was directed at the web applications domain. There have been several key applications that were very well accepted by end users outside the web world, such as spreadsheets or email filters [8]. But the web includes a novel aspect that has not yet been exploited in the EUD domain: all EUD solutions, for example, spreadsheets, are ultimately based on off-the-shelf back-end functions, such as mathematical functions on data, that can be visually and simply invoked by users. If an as yet non-existent function is required to solve the problem (for example, a function that is not implemented in Excel), end users will not be able to find a solution to their problem. Thanks to the web, though, end-user development components and back-end functions can be service wrappings available in a service-oriented architecture (SOA), for example. This way, today's Internet provides EUD with the unprecedented wealth of an ecosystem of services and resources that can serve, having been previously wrapped by means of a REST philosophy [20], as a back-end for the new user-centred RIAs [18]. This wealth of *functionality,* plus the possibility of creating visually appealing web applications, makes the above EUD constraints, which limited the end-user options and generated a high interest in academia and industry [3], a thing of the past.

Faced with Web 2.0 business opportunities, some companies started to focus on people, and hence RIAs, as the SOA entry point [16]. What they needed was a means to bridge the gap between people and services. It was then that they came up against the traditional shortcomings of RIAs. Consequently, a number of user-centred RIA business frameworks are now beginning to proliferate. Worthy of note are IBM's solution, named SOA for people [21], and SAP's proposal, called SOA-People (http://www.soapeople.com). They focus on a portal framework acting as a SOA front-end to maximize people's productivity and collaboration. The increasing interest in this approach is indicative of the current importance of user-centred SOA in the business world. However, existing approaches focus on employing particular Web 2.0-based technologies to deliver a front-end to SOA rather than lending attention to the EUD process and user-friendly component modelling [22].

Other companies are focusing on a different approach, stressing the mashup and EUD ideas, but overlooking the exploitation of real back-end business logic. This is the case of the aforementioned iGoogle, Yahoo! Pipes, Yahoo! Dapper, OpenKapow, Kapow Katalyst Platform, JackBe Presto Cloud, Netvibes, Microsoft Popfly, Marmite [23], AMICO [24] and so on.

These solutions somehow empower programming-illiterate end users to develop, within a very short time to market, their own RIAs by visually connecting components with varying abstraction levels to create a GUI for accessing and using services and resources of different kinds. These approaches foster composition, loose coupling and reuse on the front-end layer, and aim to advance towards a *user-centred service conception* [19]. The EUD movement is based on the same premises [8]: offer users visual compositional programming interfaces based on closed, off-the-shelf components (ranging from web forms, through data operators, visual elements, like buttons, fillable fields, to mathematical functions, etc.) available in some sort of repository. Users will be able to integrate these components into their target design often using drag-and-drop techniques. Each component is a node within a dataflow. This way, the data, sourced from either the user at runtime or a some service invocation, pass from one component to another and generate a workflow, where each component solves part of the problem at hand. All these premises are set out in several technical reports generated by the Open Mashup Alliance (OMA) [25], which has been trying to join the efforts of the different web mashup approaches since 2009. An end-user applications development model should implement these premises based on a number of success factors examined in [8], of which the following should be highlighted:

- The components behave like black boxes that perform a function on input data and generate output data.
- Users instantiate these components based on a catalogue organized at different levels of abstraction. Catalogues include natural language descriptions of each component and its functionality.

- The data are associated with some sort of semantics or typing whereby the EUD tool can infer whether a design component is interoperable with another and, at design time, recommend links between components or suggest which components might be included in a partial design because they contain input or output data that are compatible with existing components.

Most of these factors were elicited from the most successful EUD solution to date: spreadsheets. Also Wu et al. [26], Jones et al. [27] and Lizcano et al. [28] conducted studies analysing and categorizing such factors.

Even though current mashup solutions take up the idea of creating data mashups, they are of no use for creating service mashups or even managing real services in an open IoS [29]. The major pitfalls of these solutions are that they are unable to create web applications with front-ends (as is the case of Yahoo! Pipes or Yahoo! Dapper), do not enable intercommunication and dataflow (as is the case of iGoogle or Netvibes) or require programming expertise (as is the case of Kapow), etc.

In any case, each EUD tool and/or approach has its pros and cons, and offers particular functionalities in the EUD field for creating end-user solutions. To date, however, no detailed study of these tools, their success factors and the generated products had been performed.

As mentioned earlier, this paper formalizes a new compositional model [30] focusing on the EUD of RIAs. Its aim is not only to reshape web engineering [31] to meet the needs of EUD but also to define its components as problem-solving building blocks (instead of programs and software developments). This way, end users would be able to build RIAs based on elements akin to their expertise, view of the problem and understanding of the solution [32]. We have taken a user-centred approach based on enterprise RIAs to create this new compositional model that helps to standardize the composition process and improve application performance and reliability [33].

This compositional model abstracts and accounts for all the current EUD solutions (iGoogle, Kapow, Yahoo! Pipes, Yahoo! Dapper, and so on). Also it can be used to objectively compare the approaches, eliciting their advantages and disadvantages, improving their shortcomings. More importantly, it offers guidance for developing, enhancing and propagating more and better EUD tools in the future.

The compositional model is the groundwork for the design of the EzWeb/FAST tool. EzWeb/FAST implements the model and has a fair number of advantages over other EUD tools existing today. For a comparison of the EzWeb/FAST with other EUD tools and its contributions to the state of the art, see [34].

EzWeb/FAST relies on the European FP7 Fast and Advanced Storyboard Tools (FAST) Project and the EzWeb R&D Project [34]. The wholly visual web IDEs (integrated development environments) developed as part of these projects enable customers without any programming skills to create RIAs by dragging and dropping resources and building blocks from a catalogue, and then tailoring and interconnecting these resources to meet their requirements in a very short time to market. These projects are part of NESSI [35], the Networked European Software and Services Initiative, which considers the need of end users to be able to develop and exploit RIAs in the future Internet. Other similar initiatives are beginning to proliferate in this research field. Of these, the NEXOF-RA project [36] deserves a special mention. We are collaborating and actively participating in this project, which aims to build the Reference Architecture for the NESSI Open Service Framework by leveraging research in the area of user-centred service-based systems, and to consolidate and trigger innovation in service-oriented economies where the end user can simply orchestrate, invoke and exploit services.

The 7th Framework Programme has funded other projects similar to FAST and EzWeb, as they embody strategic motivations and objectives on the European research and development agenda. ServFace is another STREP project funded under the same European Commission's 7th Framework Programme that bears some relation to the ideas presented in this paper. This initiative aims to add an integrated UI description and development approach to SOA concepts by introducing the notion of a correspondent user interface for services. This is a completely bottom-up EUD approach: the idea is to enrich web services and resources with UI descriptions (i.e., in UML) to build generic faces into the back-end, which the users can manage through such faces. We also considered this viewpoint in the definition of the compositional RIA end-user development model.

3 Motivation and Goals for Setting up and Formalizing the New EUD Compositional Model

Empirical research has pinpointed a major challenge facing the construction of user-centred RIAs: the creation of a visual service composition tool enabling programming-illiterate users to create and share their own off-the-shelf resources composition [2]. The proliferation of big budget international projects, such as FAST or ServFace, is evidence of this.

Nonetheless, the creation of a visual development tool is only the first step towards the solution of a broader problem. This toolkit should be supported by a formalization framework [32] that validates and standardizes the development environment, its implementation process and the produced resources [8]. The glut of mashup tools and user-centred web services has what might be termed a heterogeneous and contradictory rationale. The creation of a new end-user compositional model could help to create a consistent discourse in the field of user-centred RIAs [29]. It could also lead to a common conceptualization of the EUD process and its terminology, of how to create building blocks, and, finally, of how to manage resource interoperation and functionality in all existing solutions [24]. This would define the conditions to be met by EUD tool building blocks (iGoogle widgets, Yahoo!Pipes Pipes, Yahoo!Dapper Dapps, AMICO cells and adaptors, OpenKapow gadgets and robots, etc.) to assure flexibility, interconnection and tailoring to end-user needs [37, 28].

This formalization effort is the first step towards a hypothetical global standardization process that could enable, through proper interconnection, the joint management of resources created by different tools and IDEs, no matter what their source.

A user-centred compositional model improves the traceability of the RIA creation process [18], at the same time as it increases the visibility and intelligibility of the whole EUD process [4] (from the comprehension of the real problem, through the solution developed in the final release, to the exploitation of the self-service solution [6]). An end-user compositional model will simplify RIA end-user development, fostering the collaborative facet but also simultaneously improving security, reliability and trustworthiness [37].

The visibility and traceability provided by the proposed compositional model are able to achieve several strategic HCI goals. HCI is highly important branch of knowledge in the RIA and EUD field, improving the user feel and experience [13]. An outline of HCI-related motivations follows [37]:

- A user-centred compositional development model will consistently clarify the real options that knowledge workers will have when they use EUD solutions to create their RIAs.
- The model will improve the intelligibility of both the composition process and the resources aggregation process.
- The definition of a new compositional model will improve the visibility of the actions taken by end users and their effects on both the development tool and the created application.
- The model will foster both development environment and development process fault tolerance.

Another potential benefit of a compositional model would be to enable EUD frameworks to abstract and shift the functionality and rationale behind run-time released RIAs and components to the EUD software components by using the standardized vision and conceptualization offered by a formal component. This way, EUD frameworks could straightforwardly import any RIA to another framework, drawing, in all cases, on end-user effort and knowledge.

4 Requirements of the Compositional Model for the End-User RIA Development

This section will describe the requirements that we need to address in order to define the end-user compositional model for the development of RIAs. Some of these requirements are elicited from current EUD tools, like iGoogle, Yahoo! Pipes, OpenKapow and so on [8]. The procedure starts from the general requirements for a compositional model accounting for the solutions and success factors of EUD tools, and then details the specific requirements for achieving our objectives [38].

As repeatedly stated in the literature [39, 40], three key aspects need to be defined in order to describe a software compositional model or, more generally, design a component-based software architecture:

- *Component model*: a component is a concept serving to abstract an element, such as a software element, from the real world. These abstractions are modelled to represent program components (such as data sources, visual elements, web portals and sites, workflows, operators, and so on) that solve a real-world problem.
- *Compositional model and technique*: this model states how to build the component model and determines what mechanisms, workflows and development phases are suitable for this purpose. Furthermore, this task must define how components should be connected for intercommunication purposes. There are very interesting approaches to the application of EUD for managing the evolution of web-based workflow applications [41] and for integrating services and resources in web portals [42], outlining a RIA development model based on a dataflow-oriented workflow among components described in the component model.
- *Composition language*: this language indicates how to describe the architecture of any system conforming to the component model. Although composition languages usually define how the components are represented, dependence on the component model should be avoided in order to provide interoperability with different models in the composition process.

Note that, as our compositional model is designed with programming-illiterate end users in mind, only the first two models are of any use to our target users. They define the elements that users will handle and how they will handle them. An EUD framework or tool (such as EzWeb/FAST or the existing iGoogle, Yahoo! Pipes, etc.) will offer end users a set of components that conform to the described component model, as well as support and guidance for applying the composition techniques and building component-based web applications. End users should, however, not manage the language implementing the components and the compositional system, which programming-proficient users will use to code and implement components enabling EUD.

Earlier studies on compositional models [39] suggest that there are a number of requirements to be met before building the compositional model:

- The component model must taxonomize composition interfaces with declared hooks (inputs and outputs), omitting merely visual elements. The composition programs produce the functional interfaces, resulting in efficient systems without any superfluous GUI elements.
- The composition techniques must consider the component hooks so that composition programs are dataflow-based program transformers and generators.
- User composition languages shall be visual, where each type of model component shall have different basic representations. These visual languages shall be implemented transparently for the user as conventional code, where the use of standard object-oriented languages is an option.

However, more detailed requirements need to be defined for specific application domains. In our case, we need to focus on enabling the user-centred composition of applications from services and other back-end resources to empower knowledge workers to materialize their expertise and knowledge as ad-hoc RIAs that they themselves create to support their routine knowledge work [3, 1]. As our aim is to create service-based RIAs, there follows a brief study of the different aspects addressed by such applications [5]. Note that all of these aspects must be implemented by the end-user compositional model:

- *Service discovery*. Service, resource and component localization is a necessary part of RIA construction. It should gather all the required pieces: the actual services, and tailoring and integration resources. To do this, several approaches have been proposed for the specific case of web services. They are based on both metadata and a global catalogue or register. Some have not succeeded due to poor semantics [43] or failure to gain a critical mass [44].
- *Service invocation mechanism*. At some point, the RIA will have to invoke services. The challenge related to this aspect is to define its inputs in terms of the respective service and then translate back the results to provide the RIA dataflow. This issue cannot be overlooked since it has an impact on the ability to combine services from diverse sources or even completely different services.
- *Service orchestration and choreography* [45]. One of the core principles of SOA is that it relies on loosely coupled services that do not call each other directly. Processes can then be built on top of the services to provide coordination by orchestration or choreography. Central coordination, as in an orchestration engine, makes more sense for RIAs since the application does the orchestration.
- *User interface*. The application interacts with users at all times through a set of interface elements (such as input, output and navigation elements). Yet the UI elements are not fixed, and they evolve depending on the results of the service invocations.
- *Presentation logic*. Presentation logic is all the UI-related logic that exploits context information for tailoring and customization purposes. Also, presentation logic should address some additional RIA problems, such as UI harmonization and multiple sources of context information.

These aspects can be arranged into a very broad spectrum of component models, as discussed in [46, 47 and 33]. By adopting the maximum modularity principle to take advantage of the long tail of users, these options can be narrowed down. Most knowledge workers actually have trouble accessing or even understanding web services because they do not have a user interface. On the other hand, a smaller group of power knowledge workers is able to deal with interface and service abstractions as different entities. Finally, a tiny fraction of the knowledge workers will have programming skills. The compositional model to be created must satisfy all three types of users (programming-illiterate users, advanced users and programmers) so that they can all exploit the web services that will make up the back-end of their RIAs irrespective of their programming expertise.

As our contribution to the emerging EUD of RIAs, we implemented the above requirements in an end-user compositional model. We explain the details of this model in the following sections.

5 End-User Component Model for RIAs

The end-user component model defines what components are necessary to compose a RIA that solves a real problem and how they are organized.

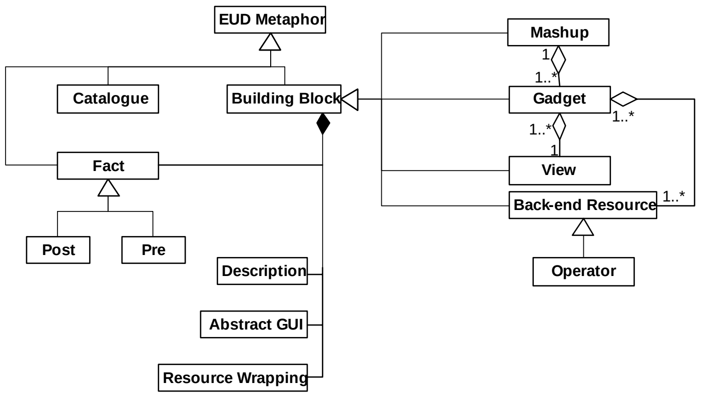


Figure 1 Component model of end-user RIAs

Figure 1 shows the EUD components for building RIAs and their relationships. This model aims to solve the problems described in Section 3 and, at the same time, exploit the key success factors of existing EUD applications [28], discussed in Section 2. Additionally, the model considers ideas suggested earlier in the state of the art by De Silva and Ginige [48] in an EUD support meta-model for business information systems, which is tailored for the development of RIAs. The key and innovative part of this approach lies in the back-end resources: these design elements will be the result of simply wrapping, as shown in Section 7.2, components already existing in EUD solutions, such as iGoogle or Kapow Katalyst, or new web services. They will be simply wrapped using a template that makes service invocation compatible with the pre- and postconditions proper to the above components model. This way, it is easy to convert the ecosystem of services located on the web into valid resources for the EUD of RIAs. Responsibility for wrapping falls to the service provider, motivated by what would be the success of connecting with the long tail of Internet users and benefiting from service use, or to programming-proficient users that detect the need for the service during the EUD of a RIA and decide to share their advances with other members of the end-user community.

In conclusion, the design elements of a RIA will be published in collaborative catalogues and will have a natural language description that knowledge workers can understand, a simple user-manageable interface for invoking functionality through a remote resource or accessing disperse heterogeneous data via the web. Finally, some design elements will be able to interact with others based on the same input and output data that they share and will be modelled as user-understandable pre- and postconditions.

The components defining the model and the motivation for their inclusion are detailed in the following.

5.1 Back-End Resource

In the context of RIAs, web services can be regarded as components that can and should be composed into larger systems [49]. However, web services are just a particular case of invocable resources for composition.

One of the main advantages of our proposed wrapping is the fact that tailoring is not tied to traditional SOAP-based web services. The model is open to all kinds of back-end services, such as databases, legacy systems and even REST-compliant resources. Thus, we are offering a new approach to enterprise application integration problems [50].

Bearing this in mind, the component model defines a back-end resource as the key component required to wrap or tailor services for subsequent composition. On the one hand, resources can be seen as an abstraction of an invocable method (i.e., one specific method for a web service, a POST method for a RESTFul service [20] or any other type of back-end resource conforming to this concept [51]). On the other hand, a standardized interface should enable a composition technique. We propose to model the inputs and outputs of the resources as pre- and postconditions composed of atomic assertions. They are modelled using semantic technologies and are called *facts*.

5.2 Operator

Operator is a subclass of back-end resources. Operators are meant to transform and/or modify data in piping processes. No constraints have been placed on operators in the proposed end-user compositional model, and they are invoked through a common interface as if they were simple tailored services.

The end-user compositional model must define several instances of general-purpose operators, such as aggregators, filters, selectors or iterators. Moreover, the development model will provide the option of extending operators on demand.

5.3 Fact

Scientifically speaking, a fact is an objective and verifiable observation representing assertions regarding a matter. From the end-user compositional model standpoint, facts are instances of domain concepts and are the basic information unit of RIAs. They enable assisted semantic composition to help users to enact processes.

To do this, they act as pre- or postconditions for the other building blocks. Preconditions form a set of constraints that restrict the execution of components, whereas postconditions represent changes as a result of component execution. In the next section we will see how pre- and postconditions play an important role in the EUD composition technique.

At runtime, facts can also be stored in a knowledge base representing the application state.

5.4 View

View can be seen as a generic graphical UI acting as a service front-end. It is responsible for establishing visual communication and the interaction mechanism with end users. In our proposed model, graphical elements contain both view and presentation logic (i.e., event management or rendering operations). Views are considered as black-box components and can be developed in any (web) technology. These developments are supported in the current state of the art, where conceptual UI patterns have been identified and specified in business management applications [52]. Note, however, that they should be designed as generically as possible to promote their reusability across different application domains.

Remember that views will be the interface that end users will see and interact with, and they have to offer the best user experience. But this is not an easy thing to do with just a generic interface. This is where component parameterization comes in. Parameterization is useful for delivering customized forms (i.e., enabling internationalization or tailoring of general-purpose interfaces for a specific domain). In some cases, a customized generic view will not meet user requirements either. To solve this problem, the model supports domain-specific views (like forms, specific GUIs, and so on).

As mentioned above, views are regarded as black-box components. To support interaction with the other model components, it is necessary to define their public interface. To ensure composition modularity, the forms offer an interface based on the above pre- and postconditions.

5.5 Gadget

Gadgets are probably the most important component of our model. They are the minimal functional blocks that can be executed independently. They include business logic and graphical user interfaces interconnected with each other. Like the above components, gadgets have a fact-based interface. This interface will play a key role in their composition to create mashups, as discussed in Section 6. Bearing these constraints in mind, gadgets can be created in two different ways: (a) by linking several resources, operators and a view together in compliance with the end-user development model, or (b) by developing a monolithic and ad-hoc piece of code on the condition that it conforms to the gadget interface.

5.6 Mashup

The last key component of our component model is the mashup. A mashup is a top-level RIA component, and it is literally made up of a set of gadgets. Specifically, a mashup is a meaningful aggregation of gadgets vested with business logic. Business logic comes from the combination of each gadget's inner logic, plus the composition logic.

There is a fact-driven approach to flow between gadgets. This is equivalent to the famously successful dataflow between spreadsheet cells. Both pre- and postconditions are used to drive the dataflow during the mashup execution. This technique will be explained in Section 6.

Finally, note that mashups can also be composed to create bigger ones. This is possible because they also share the same fact-based interface as the other EUD components.

6 End-User Development Model for RIAs

The aim of the end-user development model is to define how the components described in Section 5 can actually be composed to build real RIAs.

The development model proposed in this paper is as follows: end users give a natural language description of how to solve familiar problems systematically using distributed information sources, dataflows among these sources, by accessing remote resources, etc. [16]. For example, if users repeatedly have to copy and paste data from one web portal that wraps public administration services or a consumer service in the commercial sphere to another, they would be able to give a natural language description of the solution that they are looking for. This description will contain nouns that can be used in a keyword search of a components catalogue, where these components are tagged, applied by their providers and other users to find preliminary parts on which to build the solution [18]. When they have identified a valid component for the RIA under development (like a wrapped service, a component for searching data on a web site, etc.), users will build it into their composition. At this point, the framework implementing the EUD composition will infer recommendations of new components based, first, on the natural language description given by the user and, second, on the semantics of the data of the components already in use [29]. This approach based on directed searches and supported by folksonomies [53] (taxonomies of components inferred from lightweight semantic annotations, like tags, imposed by the long tail itself) and on social IT support technologies like blogs, wikis, etc. [6], appear in numerous EUD approaches like iGoogle, Yahoo!Pipes, OpenKapow or RoboMaker [28].

Using this method end users will be able to find an exact (or a similar) solution in a components catalogue published by a software provider or an end user that has already wrestled with a similar problem. In this case, the user will simply have to instantiate and parameterize this solution. More often than not, though, users will have to build their own solutions by mashing up several components recommended on the basis of their natural language description of the target solution. Generally, end users are known not to be good at decomposing problems into subproblems; therefore, users may find it difficult to create a RIA from the ground up. Nonetheless, we describe a primarily bottom-up process, since the EUD framework will recommend the off-the-shelf elements that users will mostly use. These elements will then be equated with parts of the problem as it is conceived by the users (for example, wrapped services that are equivalent to web portals that users use on a daily basis, etc.), and end-user development will be based on these elements. After instantiating these off-the-shelf elements, the users will add more and more functionality to their RIA based on the elements already at their disposal and recommendations based on lightweight semantics. Based on other earlier user actions and the data processed by the available elements, the framework will infer recommendations of other elements that users will be able to build into their application. Users will take a top-down approach to search the catalogue first for more general-purpose elements to meet their needs. Catalogues are not the most straightforward way of locating components, and the complications [54], such as the difficulty in finding functional elements that perform the target operations and procedures, problems in understanding an element's functionality and the complexity of finding different elements that are compatible with each other and can cooperate on involved tasks, have been under investigation for many years. Our approach has dealt with all these problems using the lightweight semantics that accompanies each element and can be edited, improved and supervised by each end user that uses the component in question. A user can tag a component, and a folksonomy [53] of elements will be generated based on the tags used by the element provider and users enabling keyword-based element searching. Similarly, each element will include natural language descriptions to explain the functionality and possible uses of the component. Finally, the component inputs and outputs will be specified by the tags for each input and/or output data item. This way, a compositional tool can match an element to the data present in a partial design and check whether it is compatible with that design or recommend parts that are interoperable with the ones in use. The building blocks hierarchy present in the EUD catalogues, classified in decreasing order of generality, is as follows: mashups, gadgets, views and back-end resources. Each mashup is composed of multiple gadgets. Gadgets offer users a visual interface for managing a wrapped resource. Thanks to the support of software providers, these are the most populous elements in today's EUD catalogues. Through visual and semantically-driven wiring, end users will be able to build these elements into their mashups and create dataflows between them. These flows will help to convey the knowledge workers' systematic knowledge, explicitly exploiting their problem expertise. If the catalogue does not contain the gadget that the end users need, they will have to use finer-grained EUD components —visual items or back-end resources (operators or services invocation)—, to compose this gadget.

Note that if the back-end resource that users require is not available, the entire EUD process will end without the RIA required by the users being built. This way, the whole RIA-oriented end-user composition model described is based on two premises. First, the widespread use of current and future EUD tools will end up populating a range of components catalogues with parts enough to be able to increasingly satisfy the more general needs of the end users. Second, the end-user composition model can never supplant traditional programming processes, as they are still absolutely necessary for building these back-end resources. The compositional model enables end users to compose and parameterize the RIAs based on existing components, releasing them from tedious routine tasks and boosting their productivity [6].

This EUD model contains two key composition levels: the wiring of different gadgets with each other to build a mashup, and the piping of different back-end resources and visual elements to build a gadget. Both levels are supported by a composition technique based on pre- and postcondition mechanisms.

In the following, we present the two composition levels for gadgets and mashups, and the fact-based pre-/postcondition mechanism.

6.1 PRE/POST Mechanism

As discussed earlier, all the components in the EUD composition model have a common interface whose inputs and outputs are defined in terms of facts. Facts are data accompanied by lightweight semantics useful for ascertaining their type and the ontological concept that they represent for association with other facts referencing the same semantics. Components can use preconditions (PRE) to constrain their execution. A component that needs a certain data type in order to execute properly will wait until it receives that information. Once the component has received all the data that it requires, it will execute.

Components usually generate some output data belonging to a specific data type and a domain class at run time. These are postconditions (POST). Once generated, a postcondition is usually stored in a knowledge base. In gadget composition, the postcondition will be propagated along semantic pipes to the next component in the composition chain, whereas, in mashup composition, an inference engine will propagate dataflows.

Both the components of the model and the PRE/POST mechanism play a role in the parameterization and tailoring of properties. First, components should be parameterized. To do this, a component is instantiated by filling its internal attributes with specific values (see Figure 2a). This parameterization can range from the component having one interface or another (based, for example, on different skins) to the component being set up with user-specific context data (like search preferences, email account or associated web services like social network profiles configurations, etc.).

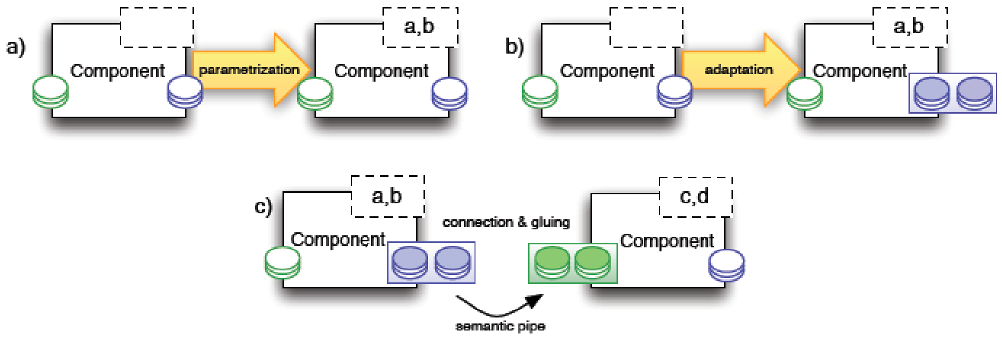


Figure 2 Component parameterization and tailoring

Then, the component has to be tailored to the execution context through its interfaces, operators, etc., to output the target postconditions, taking into account that the precondition may also be modified (see Figure 2b). Thanks to this step, called tailoring, the data of different components that a priori do not have the same syntax or typing will be able to cooperate with each other, provided their semantics denotes that both data reference a common meaning, formalized in an ontology, for example. Suppose, for example, we have a design where the precondition of the component is a place and the purpose of the component is to indicate that place on a map, whereas the postcondition of another component is a data item that is a priori a character string. If the tags associated with the character string indicate that its meaning is a place where an event is to take place, that postcondition has to be tailored to indicate that the generated fact is a location, and, semantically, matches the precondition of the map component. Finally, facts are connected by creating semantic pipes thus enabling gluing (see Figure 2c).

6.2 Gadget Composition

The bottom level of the end-user development model refers to the composition of existing resources, operators and views to create gadgets. We will illustrate how this composition technique works by means of a login gadget, illustrated in Figure 3.

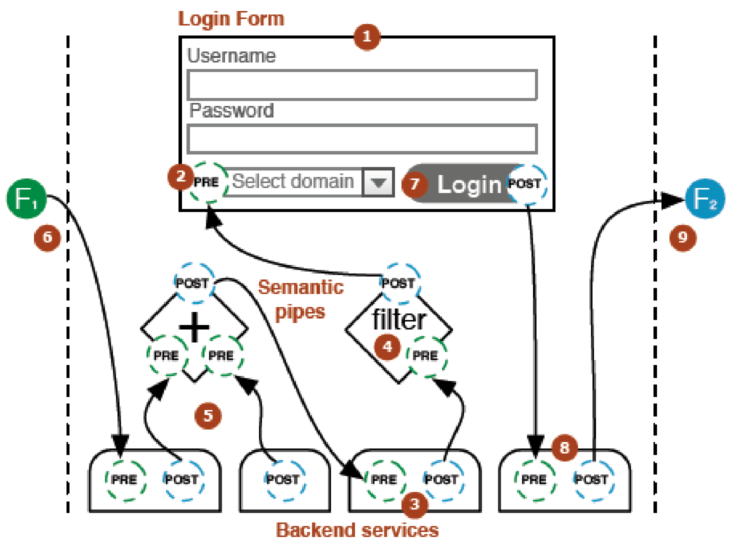


Figure 3 End-user gadget development process

To get the gadget development started, end users usually start by selecting a view that meets their needs (1). In this first step, existing approaches can be applied to set out user requirements in complex user interfaces that meet such requirements, as expounded by Díaz et al. in 2001 [55]. In our approach, visual development is completely bottom up and the users will be able to generate the interface based on general-purpose visual elements, such as forms with editable fields. The selected view may have to satisfy a precondition (2). In this case, end users will need to look for a back-end resource whose postcondition matches the above precondition (3). If end users are unable to find exactly the right resource, but there is another resource whose output is quite like what they are looking for, they might want to use an operator to adapt the data to the specific precondition format (4). Semantic pipes will link back-end resources, operators and the view. These pipes will use semantic matching to guarantee the validity of the data and their data type.

Thanks to the PRE/POST mechanism, back-end resources have their own preconditions. These preconditions also have to be satisfied in order to get the gadget working. If they can be satisfied by means of other resource postconditions, the user will connect these resources (5). If not, the unresolved precondition will be one of the gadget preconditions (6), thus having to be solved.

When the view preconditions have been satisfied, a GUI event could, depending on the type of form that the user has selected (interactive or otherwise), be necessary for the view to execute (7). View execution should create some output data (in the shape of a postcondition). Depending on the gadget business logic, the postcondition could be propagated to a back-end resource to validate its data (8). Finally, if the execution is error free, the gadget postcondition will generate the new fact (9).

6.3 Mashup Composition

Mashup composition is the top level of the end-user development model and generates a fully functional RIA. As expected, every gadget to be composed has a set of attached pre- and postconditions that will be used to drive the dataflow from one to another through a set of facts output during the mashup execution. This way, a gadget has two possible states: reachable and unreachable. If all the preconditions of a gadget are met by the facts output during the mashup execution, the gadget will be reachable. Otherwise, it will be unreachable. In this case, users will be told that the data required for the gadget to work are not being produced by the dataflow, and they will be instructed as follows:

- The compositional environment shall recommend new components that contain the necessary data in their postcondition, suggesting in the first place those components whose preconditions are also absolutely or almost completely met.
- Users receive recommendations on data tailoring actually generated by the dataflow that might possibly share the semantics of the unsatisfied precondition. To do this, the framework will have to use support ontologies to check the lightweight semantics associated with each data item, recommending data that are likely to be interrelated and enabling users to complete the necessary tailoring for the generated data item to be syntactically consumable by the unsatisfied precondition.

Note that end users do not explicitly define any dataflow between gadgets at execution time. End-user IDEs will be able to define these dataflows at development time, through the PRE/POST mechanism. Thanks to the PRE/POST mechanism, there should be no obstacle to adding a gadget whose preconditions are satisfied by the current facts present in the mashup.

7 End-User Development Languages for RIAs

End users do not manage these languages, as they use the components as black boxes through a framework supporting compositional development. These low-level languages will be useful for user-centred component or EUD solution programmers and providers to be able to implement frameworks that instantiate the described compositional model.

In order to meet all the development language requirements [34], we envision three different representations or languages with different aims:

- End-user visual development language (EUDL): a visual language enabling end and technical users to intuitively and productively compose their own software solutions.
- EUD modelling language (EUDML): a markup language used for intermediate storage of compositions and development. This representation is not intended for use directly by users. In actual fact, it is designed for processing and transmission.
- Execution language: RIAs have to be compiled to executable languages for deployment on different execution platforms.

The RIA design process will include a set of model transformations [56] from the visual language to the execution language. However, these transformations will be done automatically, and end users will only deal with the visual language. In the following we detail these languages and the transformations.

7.1 End-User Visual Development Language

End-user RIA development will require a language for visually composing the different EUD components. There are lots of visual languages in the literature [57, 58], and some even describe how services are composed [33, 46]. These visual languages often suffer from scalability problems and, therefore, alternatives, such as EUD support hypermedia tools have emerged [59]. This end-user RIA development approach was conceived by researching and observing such tools. Finally, there are several languages designed from a user-centred perspective. For this reason, we have not developed a new language, and have simply adapted visual languages to the components and development model presented in this paper. A number of visual languages and tools have been successfully used for many different purposes (e.g., programming, user interaction and visualization) [47, 60]. Visual languages attempt to provide an effective, graphical, non-linear representation that has been successfully applied to modelling (e.g., UML), parallel computing, laboratory simulation, image processing, workflow description, hypertext design, and even object-oriented programming [61]. Following our EUD approach, software composition is potentially a good application domain for a graph-based, visual notation. Instead of focusing on typical composition issues like how the "spatial" architecture of a software system can be specified in terms of components and connectors, however, we describe how services should be composed in "time" [62]. Apart from describing the dataflow structure of the interaction between different services, we have also included a separate description of their controlflow dependencies [62]. A limitation of a controlflow-based visual language applied to service composition, however, is that there is no visual notation for specifying adaptations between mismatching service interfaces [64].

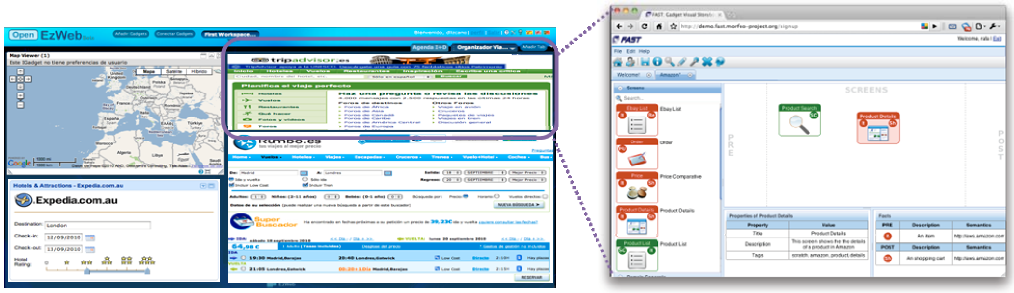


Figure 4 Example of the visual language used in the EzWeb/FAST framework that implements the end-user compositional model

Figure 4 shows an example of the visual language used in the EzWeb/FAST framework that implements the described compositional model. The EzWeb tool is illustrated on the left, where the different tabs are different workspaces, and each one contains a set of gadgets (a map built from Google Maps, a hotel management system created by Expedia, a gadget with tourist information that invokes off-the-shelf Amazon products, and an airline/train/coach ticket search engine). These gadgets will cooperate with each other through their pre-/postconditions. Users can set up a dataflow between the gadgets as if each component were a spreadsheet cell, and data input in one cell were to trigger the dataflow and service invocations in the others. FAST, a tool for composing each gadget based on its building blocks, is shown on the right. This tool is useful for designing the interface, data use, services invocations, operating on data, etc. FAST is a design time development environment, whereas EzWeb is a RIA development environment at development time and a technological execution platform at run time. The explanation of the visual language used in each tool is irrelevant in the field of EUD research generally, as a multitude of languages and options would satisfy the needs of this end-user compositional model.

7.2 EUD Modelling Language (EUDML)

Visual development languages are intended to be used by human beings through graphical representations that must be serialized for computer processing. Therefore, the EUD frameworks need to deal with a machine-friendly representation of the building blocks created during design. The Open Mashup Alliance has published an Enterprise Mashup Markup Language (EMML) [65] containing support for serialized representations of composite applications and components. EMML was a seed for the EUDML proposed here. Apart from the serialized representation of building blocks, EUDML must meet the following requirements.

First of all, EUDML must be expressive enough to losslessly represent the building block model being designed. Not only must it ensure model persistence but it must also support social sharing mechanisms. Resulting artefacts must conform to an unambiguous representation capturing all the modelling information and, at the same time, make provision for their translation into executable objects. Unlike the end-user development language, the focus shifts from user-friendliness to computability and execution performance, raising different design constraints. This is why the behaviour of the EUDML needs to be defined.

Whenever a user visually adds a new building block to a composition or creates a new connection (i.e., pipes or wires) between two components, this visual manipulation has an effect on the model that has a counterpart in the EUDML representation. Both representations must be kept consistent throughout all the development steps, since there is a definite correspondence which is manageable by the tool.

EUDML-modelled documents should represent the following groups of information about an EUD component design:

- *Non-functional properties*. These properties describe the building block under design and include component meta-data, such as author, creation date, semantically enriched tags and so on. Their purpose is to better identify and describe the building block, and they are kept separate from the behaviour that is covered in more detail and more formally in the remainder of the document.
- *Building blocks library*. EUDML-described building blocks should be defined in terms of lower level building blocks. When a user visually adds a new building block, the library should be updated with the URI identifying the block and any instantiation metadata related to parameterization, and render information for further editing.
- *Relationships*. The document must represent how the building blocks relate to each other. For instance, piping or wiring will lead to the creation of new components in the document describing the related entities and the type of relation.
- *Referenced executable objects*. Bottom-level building blocks are executable black boxes that should also be described by EUDML documents. In such documents, the building blocks library and the relationship information should be replaced by a reference to the actual implementation.

- *Pre- and postconditions*. The boundaries of the component as a whole should be defined in the form of pre- and postconditions whose formalization relies on, for example, a subset of SPARQL expressions [66], plus a human-readable description. This kind of representation will enable semantic-based recommendations during design and component semantic matching.

At the syntactical level, EUDML documents could be represented in any hierarchical object-oriented syntax. This way, the semantics described in EUDML will be instantiated in an object-oriented structure and implemented by means of object-oriented languages, for which purpose assorted markup languages, such as XML or RDF are suitable. But, JSON object notation [67] stands out on interoperability and client-server round-trip communication. An illustrative code snippet in Figure 5 depicts a gadget modelled in EUDML using JSON, which is responsible for displaying Amazon commercial products.

The EUDML representation of a gadget, plus the linked representation of the composed building blocks and the executable objects in the bottom-level blocks, contain all the information required to build the RIA. The next section discusses that process.

7.3 Execution Language

The main objective of the presented end-user compositional model is to produce an executable RIA that is able to run on existing EUD web platforms. Therefore, the end-user design process outcomes must be transformed into an executable object. This object will be tailored to the available underlying technologies supported by the RIAs (i.e., JavaScript, HTML and CSS). These well-known technologies are the groundwork of the EUD Execution Language and are common place in web development. The interesting points related to the architecture and actual execution of the building blocks are discussed here.

EUDML compilation to the final executable code includes several steps. Remarkably, though, the compilation process is done differently depending on the type of building block, mashups or gadgets, being created. In any case, the outcome is an executable object.

A compiled mashup, which is the basis of a RIA, is made up of the following components:

- *A rule-based engine.* This engine manages the execution flow of the different gadgets. It consists of a knowledge base and a set of rules. The knowledge base represents the current status of an executing mashup and contains all the instances of concepts produced at a given time. Each of the rules represents an executable gadget or a connector between two gadgets. The engine relies on the well-known RETE algorithm [68], which is not described here. The rule engine is largely mashup-platform agnostic, but some key and optional functionalities are confined to the platform-dependent module. Depending on the target platform, a suitable module will be plugged in.
- *Platform-dependent library*. This library conforms to a predefined API, providing common features such as AJAX calls or user preference management. Mashup platforms implement the above functionality in a non-standard fashion, calling for target platform-specific libraries.
- *Library of executable objects*. Pre-compiled code for all the gadgets and other building blocks that constitute the mashup are compiled into this library. The gadget compilation process will be described below.

- *Runtime metadata*. The metadata include all the information necessary to instantiate the different building blocks in the mashup and an executable rule set used by the rule engine. None of the non-functional properties are needed at this stage, however. Rules include static mediation information and, thus, are independent of a full-blown semantic reasoner.

```
gadget = {
   "uri": "http://FASTCatalogue/screens/35",
   "name":"...",
   "creationDate":"2010-02-20T18:00:00+0200",
   "description":{
      "en-gb":"...."
   },
   "creator":"admin",
   "tags":[
         {
            "label":{
               "en-gb":"amazon"
            },
            "means":"http://dbpedia.org/page/Amazon.com"
         }
         ...
   ],
   "preconditions":[
     { "pattern":
       "?x http://www.w3.org/1999/02/22-rdf-syntax-ns#type
       http://aws.amazon.com/ECommerceService#Item",
      "label": {
         "en-gb": "Item"
      },
      "id": "Item_1"
     }
    ],
    ...
```

```
 ...
"definition":{
   "buildingblocks":[
      {
         "id":"amazonListForm1",
         "uri":"http://FASTCatalogue/forms/16",
         "position": {
            "x": 132,
            "y": 200
         },
         "params": {
            "pagination": true,
            "titleCaption": "..."
         },
      },
      ...
   ],
   "pipes":[
      {
         "from":{
           "buildingblock":"amazonSearchService1",
           "condition":"list"
         },
         "to":{
           "buildingblock":"amazonListForm1",
           "condition":"list",
           "action":"showTable"
         }
      },
      ...
}
```

Figure 5 Example of EUDML representation of a gadget component

On the other hand, gadget compilation takes an EUDML-based gadget document and outputs the executable object for integration into a building block library. To do this, the following components must be compiled:

- *Interface characterization*. This is relevant meta-information that will be exploited during mashup compilation and execution. It states the gadget pre- and postconditions.
- *Building block runtime dependencies*. List of references to building block components needed for gadget execution that can be recursively traversed in order to output a transitive dependency list. The combined list of dependencies is output as part of the mashup compilation and comprises the view conveying the UI, back-end resources providing both data and functionality, and operators that transform the data.
- *Instantiation metadata*. Executable components to be instantiated at runtime can be appropriately parameterized with user information kept as part of the instantiation metadata.
- *Piping information*. A pipe states that the output (postcondition) of a building block is connected to an input (precondition) of another one. Therefore, the execution of a given building block is constrained to the reception of every precondition through their respective pipes. This will be translated into statically resolved pairs that inform the respective

connected precondition of a change when a postcondition is triggered (for instance, if a service has output data). If all the preconditions of a building block are satisfied, the associated action is launched.

The gadget compilation process will end up making a new gadget available in the catalogue. This gadget can be consumed to create new mashups.

8 Empirical Study of the End-User Development of RIAs

Having defined the emerging end-user compositional model to create RIAs and its development model, our research focused on the construction of a framework that would instantiate the defined model, exploiting the success factors present in other EUD tools for creating mashups, RIAs and composite applications (examined previously in [28] and [38]). The result was the EzWeb/FAST framework (see [69] and [70] respectively), an open-source product developed as part of research by two international R&D project consortiums. Using this framework, developed in partnership with major IT companies, like Telefónica I+D, SAP, Deri Galway or Cyntelix, and higher education institutions, including the Universität Kassel, we have been able to statistically evaluate whether programming-illiterate end users are really able to tackle certain real problems and compose a RIA based on the off-the-shelf components that major companies like Google, Amazon, Yahoo!, IBM, Sun or Oracle have made available to the long tail over today's Internet. EzWeb/FAST instantiates the end-user compositional model (see [28] and [38]). Also, it is a good test bench for checking that the created component achieves its goal: give end users access to the tools that they need to create RIAs to support or boost their knowledge work, irrespective of their programming expertise.

The conducted study was designed to answer three research questions:

(1) RQ1: How well does EzWeb/FAST, which instantiates the proposed end-user compositional model, adapt to the needs and capabilities of programming-illiterate knowledge workers?

(2) RQ2: Is the time it takes to develop RIAs applying the end-user compositional model viable and practicable?

(3) RQ3: Do the RIAs created by surveyed users meet the set requirements and how do they fare in terms of quality, robustness, reliability and completeness?

The results of this survey using the framework that instantiates the model do not validate the underlying compositional model (such validation would require the use of different frameworks implementing the model and a long-term evaluation of such frameworks), but it does check whether the components, development process and coding languages described here really do empower programming-illiterate users to compose a RIA that satisfies a given real problem.

We ran the statistical study on a significant sample of 100 users characterized in Table 1. Half of the sample were programming-illiterate users and the other half were users with some technical expertise (previous experience using BPEL, notions of JavaScript, mashup platforms, HTML, CSS), some (a minority) even having knowledge of a programming language (Java, J2EE, Php, ASP, AJAX, object-oriented programming patterns, C, C++, C#, Haskell, Prolog, and so on).

Table 1 Sample characterization

Characterization	End users (50)	Power or advanced users (50)	Total (100)
Gender			
Male	26	25	51
Female	24	25	49
Age			
< 20 years	9	10	19
20-34 years	12	11	23
35-49 years	11	12	23
50-64 years	10	10	20
> 65 years	8	7	15
Educational Attainment			
Secondary School	12	12	24
Vocational Training	13	13	26
Bachelor's Degree	12	13	25
Master's Degree	13	12	25
Employment			
Student	13	15	28
Researcher	14	18	32
Employee	23	17	40
Experience and previous knowledge			
Mashup Platforms	3	42	45
Web Services (SOAP, ESB, BPEL, etc.)	2	25	27
JavaScript, HTML, CSS, AJAX	0	41	41
Java, J2EE	0	25	25
Php, ASP	0	26	26
OO Programming	0	31	31
C, C++, C#	0	6	6
Haskell, Prolog	0	3	3

Users were asked to develop a composite application to plan business trips. They had to create a web application that searched for and booked means of transport, hotels, tourist information and localized destinations listed on a personal agenda. This solution also had to control the financial costs against a spreadsheet that included a budget. A complete description of the proposed problem can be found at [71] (see Problem 0). To solve this problem, they had to use EzWeb/FAST and an abundant set of design elements conforming to the end-user compositional model principles. For more details on the components populating EzWeb/FAST, how users found, used and tailored components, see [72] and [73], respectively.

To find out which components the sample had access to, their description, consumed and generated data, etc., register and use the free EzWeb/FAST tool, available at [74] and [75]

The survey was conducted as follows. Users were trained, and they then conducted a requirements study and analysis, and the final development. The end-user compositional model training was confined to a 20-minute oral presentation of EzWeb/FAST and its interface, and a 10-minute viewing of multimedia material serving as the developed tool user's manual and illustrating the development process to be enacted (see [76]). These demos likewise show the use of the catalogue and the semantic data channels among components. They make it clear how users used the catalogue and managed to discover useful elements to compose the required RIA. This catalogue enables end users to discover and exploit the elements it contains. It conforms to a number of premises described in [34] and uses

Web 2.0 technologies, such as wikis, lightweight semantic tagging and folksonomies, described in more detail in [77] to do this.

Note, first, that all users managed to build this solution as a mashup-type composite web application based on the composition of catalogued design elements. These elements were provided by software distributors like Amazon, Trip Advisor, Google or Yahoo! These distributors have already started to publish services as programming APIs for programming-illiterate users (in repositories like ProgrammableWeb [78], which publishes resources for partially programming literate users).

8.1 RQ1: How well does EzWeb/FAST adapt to the needs and capabilities of programming-illiterate knowledge workers?

The first result of our research is that 95 users, irrespective of their programming literacy, managed to put together a valid solution that worked satisfactorily and solved the set problem. Only five end users did not manage to build a solution using EzWeb/FAST. All five ran into trouble during the process of interconnecting heterogeneous components.

Then, a theoretical/practical presentation of other EUD web tools, particularly iGoogle, Kapow Katalyst, Yahoo! Dapper and Netvibes, was held for the 100 users in the sample. Using these tools, a total of only seven programming-illiterate users (14% of end users) and 35 programmers (70% of the technical users) were able to develop RIAs that met most of the requirements through combination. These results, very poor compared with EzWeb/FAST outcomes, indicate that EzWeb/FAST acceptance and performance among real users is better. EzWeb/FAST actually scores higher on key aspects targeted by the compositional model presented here and elicited from the critical success factors in EUD, such as framework ease of use, user perception of real-world usefulness and user perception of RIA performance.

Members of the research group were present and witnessed the RIA development process throughout the experiment. These members were involved in measuring the development time and evaluating whether the RIAs met the specifications. They then evaluated each application developed by the 95 users in terms of efficiency, completeness, robustness and quality.

Each user rated the effectiveness, usability and simplicity of the end-user compositional model underlying EzWeb/FAST in a survey. The main survey questions are shown in Table 2. The survey was built according to the principles expounded by Lehmann and Romano [79] on a five-point Likert scale. Table 2 also shows the analysed results on a 1-to-5 rating scale (where 1 is the worst and 5 is the best result), listing user ratings in terms of programming expertise and total scores. They turned out to very positive. Note that, as the majority of the sample (86%) was unable to solve the set problem using one or more other EUD tools (iGoogle, AMICO, Kapow, Yahoo!, etc.), users were only questioned about EzWeb/FAST omitting the other tools used in the experiment. Table 3 shows the descriptive statistics of the evaluation conducted by users which is equivalent to a normal distribution with a mean of 4.09 points (on a scale of 1 to 5) and a standard deviation (σ) of 0.38. Mean is the mean user rating variable. Additionally, we show the descriptive statistics of the distribution of ratings by end users and technical users, and the normalized distribution of these last two samples (end user and technical user ratings). The histograms represent the density distribution of the end-user rating and technical-user rating variables. The x-axis represents the final scores given by the survey users (from 1 to 5), whereas the y-axis shows the density of the sample giving a particular score. The density of each interval is calculated as the percentage of the sample that gave a particular range of scores divided by the interval length. To assure histogram representativeness, the selected interval lengths were 0.3 points on a 1-to-5 scale. Accordingly, a value equal to y on the y-axis for a score x indicates that a percentage equal to (0.3 * y) of the sample gave a score in the range of x-0.3 and x+0.3 points, where the value y=1.5 indicates that 45% of the sample gave scores inside that interval.

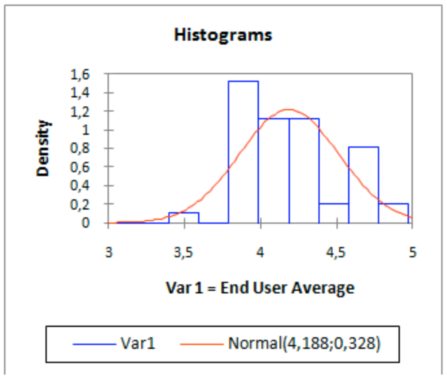
Table 2 Survey Questions

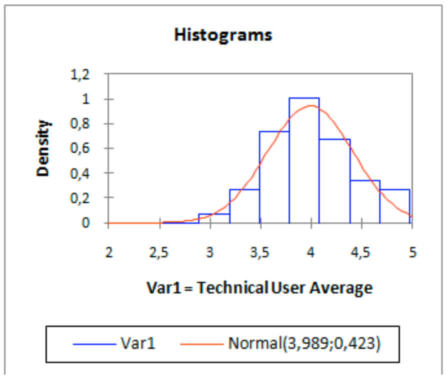
N.	Item	End Users Score	Technical Users Score	Total Score
Q7	EzWeb/FAST is a satisfactory means for creating solutions to meet personal needs when it is not feasible to develop a traditional solution due to time and/or budget constraints.	4.24	4.06	4.15
Q8	It is rewarding to use tools like EzWeb/FAST and be able to rapidly and simply create mashups.	4.28	4.12	4.20
Q9	Domain experts, web programmers and service providers should consider the end-user composition model as a design vision to be taken into account.	4.08	4.00	4.04
Q10	The more people that adopt the end-user composition model the easier it will be to find useful design components and create end-user solutions.	4.40	4.38	4.39
Q11	The end-user composition model enormously simplifies the stages of implementation, testing, debugging and any modifications to account for changing requirements of the EUD solution development process.	4.26	4.06	4.16
Q12	It was complicated to create a solution to the stated problem using EzWeb/FAST.	4.18	4.04	4.11
Q13	The design components available in the end-user composition model do not meet the needs of real-world problems: parts are either overly general or overly specific.	3.52	3.26	3.39
Q14	The communication mechanism between the design elements is not suitable for solving the problems that end users are likely to have.	3.92	3.74	3.83
Q15	The solution created using EzWeb/FAST can be straightforwardly evaluated in a stepwise manner to check that it is error free and be able to create increasingly complex solutions.	4.48	4.22	4.35
Q16	Using EzWeb/FAST, a change in the end-user requirements leads to major rework to tailor the solution to the new problem.	4.16	3.84	4.00
Q17	The EzWeb/FAST EUD platform is easy to use even first time round.	4.28	4.00	4.14
Q18	Most people could learn to use EzWeb/FAST to develop end-user solutions.	4.20	3.90	4.05
Q19	I get the feeling that it is not easy to create real-world solutions using EzWeb/FAST.	4.18	3.98	4.08
Q20	The development model interface and support built into EzWeb/FAST are too complex for end users to be able to create solutions.	4.52	4.32	4.42
Q21	Users need a lot of additional training before they will be able to use EzWeb/FAST effectively to develop their own solutions.	4.48	4.38	4.43
Q22	It is easy to link several components in the EzWeb/FAST using pre- and postcondition mechanisms.	4.12	4.02	4.07
Q23	Useful design components are easy to locate thanks to EzWeb/FAST catalogues.	3.98	4.04	4.01
Q24	It is hard to publish new design components as gadgets for use in RIAs.	4.02	3.56	3.79
Q25	The composite system built did not respond as expected.	4.20	3.98	4.09

Q26	It is hard to create a composite solution to a specific problem using EzWeb/FAST (considering that the catalogue is well enough populated with design components).	4.48	4.18	4.33
Q27	Which of the following do you think is the most realistic development time ratio considering two development options for a real problem: a) implement a solution from scratch and b) use the end-user composition model? 1. The end-user composition model can reduce development time/workload enormously 2. The end-user composition model can reduce development time/workload appreciably 3. The workload for the end-user composition model and for programming a solution from scratch is similar. 4. The end-user composition model takes more development time than traditional programming. 5. The end-user composition model does not always manage to produce a valid solution to a set problem, even if the catalogue contains the necessary components.	4.36	4.30	4.33
Q28	Using the end-user composition model and tools like EzWeb/FAST, any user (no matter how much programming knowledge they have) can create their own solution to a particular problem.	4.02	3.56	3.79
Q29	Users need to know how to program to create functional and stable solutions using EzWeb/FAST.	3.98	3.92	3.95
Q30	Developing and tailoring new design components for EUD platforms like EzWeb/FAST will be key occupation of information technology enterprises in the future.	4.16	3.88	4.02
	TOTAL	4.19	3.99	4.09

Table 3 Descriptive statistics of the evaluation

						95% Confidence Interval for Mean			
	N	Mean	Std. Dev.	Variance	Std. Error	Lower Bound	Upper Bound	Minimum	Maximum
End-user composition evaluation (1-5)	95	4.09000	0.389307	0.151560	0.038931	3.326972038	4.853027962	2.960	4.880
End-user rating	45	4.19	0.327787	0.107444	0.046356	3.547550177	4.832449823	3.580	4.880
Technical user rating	50	3.99	0.422729	0.178700	0.059783	3.161466661	4.818533339	2.960	4.880





The data in Table 3 are evidence in response to RQ1, that the sample of end users rated the EzWeb/FAST tool very positively (4.09 out of 5) and that 45 out of 50 users (90% of end users) managed to compose a RIA that solved the set problem. These results are very representative, as the sample had the opportunity, before rating EzWeb/FAST, of testing similar solutions with other products on the market that did not meet all the premises of the described compositional model and that achieve empirically poorer results [38].

The data also suggest that end users rated the tool better than programmers. This may a priori appear to be an obvious point: end users, who have no other way of developing a RIA, will be very satisfied at having been able to do this by themselves, whereas a more technical user is likely to be able to program a RIA without the help of EUD solutions, and hence will be less impressed by the framework. However, further statistical analysis is required to reveal which and what type of quantitative and qualitative variables characterizing the sample are implicated in the tool rating.

We conducted an ANCOVA analysis (Table 5) to empirically check whether age, educational attainment, employment or previous EUD and IT expertise cause the rating to vary. This analysis will be able, on the one hand, to check that the selected sample is not biased and, therefore, does not contaminate the conducted survey and, on the other, to identify what variables appear to have a direct effect on user satisfaction with EzWeb/FAST.

Analysing the study, we find that the coefficient of determination R^2 in Table 5 is very low (0.395). This indicates that there is a high percentage of variability in the modelled variable so that gender, age, educational attainment, employment and previous experience appear to explain only 30% of the rating data. The other values are due to other unknown variables. This value of R^2 and adjusted R^2 suggests that the rating of EzWeb/FAST is largely (70%) independent of the characteristics of the users rating the paradigm (Figure 6). First, this result validates the sample, indicating that there is no bias related to the qualitative and quantitative variables characteristic of the users and to their recruitment for the study. The regression model shows a horizontal and vertical dispersion of predictions, with error ranges from 2.5 to 5 points (out of 1 to 5 points), meaning that the mean rating variable is completely independent.

Table 5 ANCOVA analysis of the sample

Goodness of fit statistics

Observations	Sum of weights	df	R²	Adjusted R²	MSE	RMSE	MAPE	DW	Cp	AIC	SBC	PC
95.000	95.000	64.000	0.395	0.065	0.142	0.377	5.462	1.157	36.000	-167.99	-74.21	1.285

Analysis of variance:

Source	df	Sum of squares	Mean squares	F	Pr > F
Model	33	5.932	0.169	1.196	0.264
Error	61	9.072	0.142		
Corrected Total	94	15.004			

Computed against model Y=Mean(Y)

Type I sum of squares analysis:

Source	DF	Sum of squares	Mean squares	F	Pr > F
3.- Age	1	0.134	0.134	0.943	0.335
4.1- Education	3	0.752	0.251	0.968	0.362
4.2-Employment	2	0.163	0.081	0.575	0.566
5.- Programming expertise	22	4.387	0.199	1.407	0.146
6.- EUD experience	6	0.456	0.076	0.536	0.779
2.- Gender	1	0.042	0.042	0.294	0.589

Type III sum of squares analysis:

Source	DF	Sum of squares	Mean squares	F	Pr > F
3.- Age	1	0.084	0.084	0.595	0.443
4.1- Education	3	0.524	0.175	1.232	0.305
4.2- Employment	2	0.212	0.106	0.749	0.477
5.- Programming expertise	22	4.041	0.184	1.296	0.209
6.- EUD experience	6	0.445	0.074	0.523	0.789
2.- Gender	1	0.042	0.042	0.294	0.589

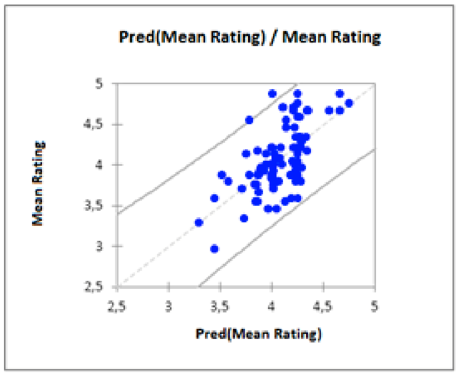


Figure 6 EzWeb/FAST rating fit based on the regression model

Having validated the surveyed sample, it is worth mentioning that the ANCOVA analysis (Table 5) indicates that the selected explanatory variables cannot be considered to be the source of a significant amount of model information ($Pr > F = 0.264 >> 0.01$). The model is not significant because the rating of EzWeb/FAST is independent of the characterization of the sample, and this means that we can assume that the results are not biased by the selected sample.

Analysing the results of the sum of squares analysis I and III in Table 5, we find the variable that has most impact on the rating. Of the studied variables (age, gender, educational attainment, employment, previous EUD experience and programming expertise), the variable with the greatest Fisher F-distribution is previous programming expertise (F=1.407). Pr > F is equal to 0.146 (the closest to 0.01) for that variable. Therefore, we can infer that the aspect of sample characterization that is most statistically significant for the rating is whether or not the user has programming expertise. The other variables have a weaker Fisher F-distribution (and, therefore, less impact on the rating). The variable with the least impact on the model is gender, followed by previous EUD experience and then employment and educational attainment. Judging by the probabilistic values Pr > F, everything appears to indicate that previous EUD experience does not affect the rating of EzWeb/FAST at all. This way, respondent age, employment, educational attainment, etc., will not alter the user rating of EzWeb/FAST.

Finally, the extent to which each variable has an impact on the EzWeb/FAST rating can be quantified using a regression model and its standardized coefficients. Figure 7 lists and plots the model coefficients. The full survey supporting these data is available in [80].

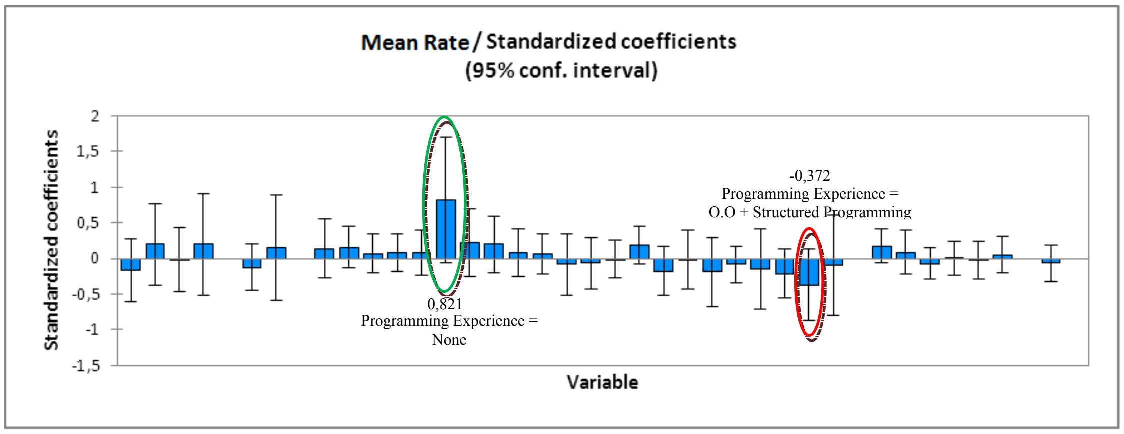


Figure 7 Impact of each sample characterization variable on rating

The factor that most negatively affects the rating is that users are experienced in object-oriented and structured programming of RIAs. These users do not appear to assimilate as well as other users the bottom-up development process based on off-the-shelf components and the top-down searches of composable parts in catalogues. However, these concepts, together with the idea of wiring closed components and the composition of new components based on building blocks, are present in most RIA composition technologies and/or tools, such as Symphony, Django, Dojo, Prototype, etc. [5], which they had used to some extent in practice. This finding suggests that the end user-centred RIA compositional model is not based on the same premises as the compositional or visual programming paradigm [30] with which the above technical users are acquainted. This, linked with several publications discussing an emerging user-centred programming paradigm [8] on which our research is based, possibly categorizes the compositional model presented here within a new self-contained programming paradigm. This would explain why the technical users conceive the described EUD components as being different from the low-level editable components managed by familiar development frameworks, just as imperative paradigm programmers found it harder to adapt to the object world than first-time developers using the object-oriented languages [81].

8.2 RQ2: Is the time it takes to develop RIAs applying the end-user composition model viable and practicable?

Table 6 shows how long it took end users to develop solutions for the set problem, as well as five additional problems proper to the RIAs domain that are described in detail in [71] (see problems 1 to 6). Each set problem was more complex, had more components, and therefore required more development time than the last.

Table 6 Descriptive statistics for development time

Development Time (minutes)	N	Mean	Std. Dev.	Minimum	Maximum
Main problem	95	8.390	3.296	5.000	21.000
Additional problems					
1	95	4.308	1.054	2.970	8.205
2	95	4.089	0.965	2.750	7.541
3	95	4.910	0.818	3.865	6.518
4	95	4.932	0.782	3.764	6.405
5	95	7.744	2.179	4.640	16.262
6	95	9.711	1.640	7.000	14.282

To compare the development time required using the user-centred compositional development model against other programming models and/or techniques, 50 programming-literate technical users were asked to solve the main problem using any non-user-centred programming IDE or framework of their choice. They were only asked to solve the main problem because it would have taken weeks or even months of work to solve the other six. All 50 users built valid RIAs, but the gap in development times was enormous, as shown in Table 7.

Table 7 Descriptive statistics for development time without EUD approach

Development Time (minutes)	N	Mean	Std. Dev.	Minimum	Maximum
Main problem without Ezweb/FAST	50	1219.98	87.010	1049	1390
Main problem with EzWeb/FAST	95	8.390	3.296	5.000	21.000

This study shows that it took the technical users with programming expertise nearly twenty-and-a-half hours to do this. In other words, development time was over 140 times greater than was required to apply the EUD paradigm. It is, of course, only logical that software development will be much faster using a CASE tool like EzWeb/FAST than starting from scratch. Nonetheless, the study demonstrated that the development times required by EzWeb/FAST are independent of developer/user knowledge of IT, programming languages, web services management, etc. This is by no means a trivial outcome and has not been demonstrated to date by other EUD tools in use. To empirically demonstrate these findings, we ran a one-way ANOVA of the development time variable against the qualitative variable of whether or not they are end users. As the sample sizes of both types of user are similar (45-50), it is not necessary to conduct preliminary studies (such as, a Levene study or Fisher's F-test) to check whether or not ANOVA is applicable. Table 8 shows the results of the ANOVA.

Table 8 One-Way ANOVA Test for development time

One-Way ANOVA				α = 0.05			
SUMMARY STATISTICS							
Groups	N	Sum		Mean	Variance		
End Users	45	416		8.32	10.14040816		
Technical Users	50	423		8.46	11.8044898		
Source of variations		Sum of Squares (SS)	df	Mean Square	F	P-value	F critical
Between group		0.49	1	0.49	0.044657305	0.83307426	3.938110878
Within group		1075.3	93	10.97244898			
Total		1075.79	94				

Looking at Table 7, we find that the F value 0.044 is much less than its critical value of 3.9381, meaning that, with a significance level of 83%, the variation of the mean development times can be said to be due to specific users influencing the above means, and this variation is not statistically significant. That is, traditional programming expertise does not have an impact on the EUD RIA development time. The ANOVA study, conducted for the main problem, was also prepared for the other six standard problems, and the result was the same: there is no statistically significant difference among the development times required by a programming-illiterate user and a programmer for composing a RIA that solves the problems. On average, all users, irrespective of their programming expertise, will take just as long to develop an EUD solution using the end-user compositional model. Without the presented compositional model, this would be out of the question for end users. For these premises to be true, however, software providers have to populate the collaborative catalogues underpinning the end-user compositional model.

For the full statistical study of the EUD process, the set problems and the solutions built, see [80].

8.3 RQ3: Do the RIAs created by surveyed users meet the set requirements and how do they fare in terms of quality, robustness, reliability and completeness?

Finally, it remains to answer RQ3. After the development process, the experiment reviewers found that the created RIAs worked efficiently and were useful. To do this, they inspected the EUML languages and low-level code developed for each application, which they reviewed using white-box tests [11]. The applications met the set requirements and had similar computational efficiency, response times and robustness to the RIAs built by the reviewers for the set problem (see [80]). In some cases, the RIAs had dataflows that the users had added (under framework guidance in all cases) to solve aspects that were not specified in the statement or were optional. Such flows led to longer response times, albeit not to problems of robustness. These positive results are partly attributable to the building blocks used in the end-user compositional model developed using web development, testing and maintenance technologies [11]. Taking these operational components, the compositional model is responsible, through the described composition techniques, for maintaining an adequate consistency of the dataflow created by the user with guidance. Hoyer et al. [82] conducted an independent study and found that the 95 RIAs built for each of the 7 problems set in the study (the main problem and the other 6 problems in the test bench) met the specified requirements correctly, robustly and reliably.

8.4 Discussion

The full study is documented at [83]. Additionally, the framework on which the study was conducted is available at [74] and [75]. This framework and the proposed end-user compositional model on which it is based are now being used by Spanish public administrations to promote new digital spaces for citizen interaction. Saragossa Town Council is using the end-user compositional model and its components to empower its citizens to compose their own software solutions to complete bureaucratic formalities, access citizens' services, report breakdowns or incidents on public thoroughfares, etc. [84]. Andalusia's Regional Government [85] is also exploiting the end-user compositional model to offer personalized public administration and citizens' services.

In view of the results of a brief additional survey (see [83] for additional material), EzWeb/FAST has received very positive ratings from a significant sample of end users with and without programming skills. The relevance of the research is not that it output a successful tool but that it elicited why this tool was more successful than others based on very similar components, such as iGoogle, Yahoo! Pipes, AMICO or Kapow. As interviewers of the surveyed sample, our perception is as follows: end users identified key tool weaknesses that map directly to the aspects of the compositional model that the above tools omit, primarily:

- 95% of the sample stated that they had no way of interconnecting iGoogle widgets with each other, whereas 76% missed the option of being able to compose widgets based on finer-grained components.
- 82% of the sample highlighted that, apart from feeds and screen scraping-based information sources, Yahoo! Pipes failed to provide useful wrapped services for end users.
- Over 60% of the sample found that component linking and tailoring mechanisms in Kapow solutions were not handy (required programming knowledge), whereas 85%

found the component search, location, parameterization and recommendation mechanisms (very similar to the ones proposed in the compositional model) to be very satisfactory

- 90% of the sample stated that the composition visual interface should not be confined to a mere spreadsheet available through a URI in AMICO.

All the above findings match the shortcomings of each of the specific tools examined from the viewpoint of the described compositional model [38]. On this ground, the key contribution of this paper is a general-purpose model for use in the implementation and improvement of a user-centred web software development tool (ranging from web pages and portals to complex composite applications) in the knowledge that the use of such guidelines and heuristics will lead to an improvement in end-user satisfaction and their perception of usefulness and ease of use.

9 Conclusions and future trends

Mashup creation and services composition is revolutionizing the Internet. Not having any programming skills, end users, on their own, are often unable to develop RIAs that solve their real problems. They are unable to use business services because they do not have a user-centred interface, and existing front-end-centred solutions are based on set and rigid component libraries that do not necessarily satisfy user requirements.

Current research in the field of EUD applied to RIAs has focused on encouraging software engineers to get end users involved in the design, implementation and testing processes from the very start of the web software lifecycle [86, 87]. But these approaches miscarry insofar as end users are obliged to grasp, understand, contend with and get the hang of components, implementation elements and development processes that were originally conceived by and for programmers. End users cannot be expected to manage programming resources (functions, objects, transactions, etc.) that have little or nothing to do with the cognitive perception of the problem and its potential solution as envisaged by the end user.

In this paper we have described an emerging end-user compositional model showing how visual and reusable building blocks can be used to easily create and efficiently build RIAs. Our research offers a new open end-user development model, where end users can exploit their unique expertise in an open innovative process of creation. Additionally, the compositional model described covers all the functionalities of EUD tools for building today’s composite web applications. It can also be used to formalize their design, compare their structure, improve their weaknesses and conduct research and development on new development frameworks in the future.

As illustrated in Section 8, our approach enables customers without programming skills to build complex RIAs from visual and tailorable components. A study was conducted to validate our research. This study confirmed that business people are able to develop RIAs that meet their expectations, provided, of course, that the adequate components are available. This, unfortunately, depends on such proposals being accepted by and successful among the long tail of users. It is easy for programmer users to wrap any SOA-, POX-RPC- or REST-based service for conversion into a EUD component, and, if the number of users interested in such tools increases, more and more programmers are likely to have a go at producing such wrappings. In any case, the study demonstrated that our EUD vision is successful: users with no programming skills are able to create RIAs. These RIAs are priceless since they exploit user domain expertise in a short time to market and with low development costs.

Additionally, we noticed that technical users experienced in different traditional programming paradigms find the compositional development process to be less intuitive than end users without any technical knowledge whatsoever. This also applies to imperative paradigm programmers wanting to switch to the object-oriented paradigm. They find it harder to adapt to than first-time object-oriented programmers [81]. This backs the hypothesis that EUD development models are possibly shaping a new programming paradigm, which, albeit spawned by earlier paradigms, such as the compositional, dataflow or visual programming paradigms, marks a turning point in terms of the conception of how to build software, where, for the first time ever, the developer will be programming illiterate [8].

Future work will concentrate on the development of a formalism that describes both the syntax and semantics of available end-user composition languages. This formalism could be used to verify and validate RIAs, automatically guaranteeing that they meet a set threshold of functionality, reliability, performance, security and clarity. Also, we are working on the definition of a taxonomy of visual elements. Our goal is to find a set of common visual patterns present in both the actual web application UIs and the service front-ends. This set would be an excellent seed for building a repository of visual UI components that could be exploited to tailor any service front-end to any requirements through reuse and connection [88].

Acknowledgements

We would like to thank the experiment participants, who, although they remain anonymous, played a fundamental role in this study. The authors thank the anonymous reviewers of this paper for comments that improved the content of the work.

References

1. Gartner Inc (2006), *Hype cycle for software as a service in Gartner research*, in Gartner Inc Report.
2. OASIS (2003), *Web services composite application framework*, in ws-caf tc.
3. Service Front End Open Alliance (2009), *Building the front end of the future internet of services*, in EU Technical report.
4. T. Janner, R.Siebeck, C. Schroth and V. Hoyer (2009), *Patterns for Enterprise Mashups in B2B Collaborations to Foster Lightweight Composition and End User Development*, in Proceedings of the 2009 IEEE International Conference on Web Services (ICWS '09), IEEE Computer Society, Washington, DC, USA, pp. 976-983.
5. S. Meliá, J. Gómez, S. Pérez, O. Díaz, (2010), *Architectural and Technological Variability in Rich Internet Applications*, Internet Computing, IEEE, vol.14, no.3, pp.24-32.
6. A. P. McAfee (2006), *Enterprise 2.0: The dawn of emergent collaboration*, in MIT Sloan Management Review, 47, 3, pp. 21–28.
7. T. O'Reilly (2005), *What is web 2.0: Design patterns and business models for the next generation of software*, in O'Reilly Media Inc. tech report.
8. H. Lieberman, F. Paternò and V. Wulf (2006), *End User Development*, Ed. Springer.
9. I. Garrigós, J. Gomez and G.-J. Houben (2010), *Specification of personalization in web application design*, in Inf. Softw. Technol. 52, 9, pp. 991-1010.

10. D. Lizcano, M. Jiménez, J. Soriano, J. M. Cantera, M. Reyes, J.J. Hierro, F. Garijo and N. Tsouroulas (2008), *Leveraging the upcoming internet of services through an open user-service front-end framework*, in Towards a Service-Based Internet, Proceedings of the ServiceWave 2008 Conference; volume 5377 of Lecture Notes in Computer Science.
11. M. Gaedke (2008), *Web Engineering: Creating Solutions in the Age of Emotion, SOA, and Web 2.0*, Tutorial in the 17th International World Wide Web Conference, Beijing, China, April 20, 2008.
12. J.C. Preciado, M. Linaje, S. Comai, S., F. Sanchez-Figueroa (2007), *Designing Rich Internet Applications with Web Engineering Methodologies*, in Proceedings of the 9th IEEE International Workshop on Web Site Evolution (WSE 2007), pp.23-30, 5-6 Oct.
13. A. Blackwell and T. R. G. Green (1999), *Investment of Attention as an Analytic Approach to Cognitive Dimensions*, in Collected Papers of the 11th Annu. Workshop Psychology of Programming Interest Group (PPIG-11), T. R. G. Green, R. H. Abdullah & P. Brna, Eds. , 1999, Leeds, UK. pp. 24-35.
14. T. H. Davenport (2005), *Thinking for a living: How to get better performance and results from knowledge workers*, Ed Harvard Business Press, Boston, Massachusetts.
15. C. Anderson (2006), *The Long Tail: Why the Future of Business Is Selling Less of More*, in Hyperion, 2006.
16. D. Lizcano, J. Soriano, M. Reyes and J.J. Hierro (2009), *A user-centric approach for developing and deploying service front-ends in the future internet of services*, in International Journal of Web and Grid Services, Vol. 5, issue 2, pp.155-191.
17. P.E. Johnson (1983), *What Kind Of Expert Should A System Be?*, in Journal of Medicine and Philosophy, 8, 1, pp. 77-97.
18. J. Soriano, D. Lizcano, M.A. Canas, M. Reyes, and J.J. Hierro (2007), *Fostering Innovation in a Mashup-oriented Enterprise 2.0 Collaboration Environment*, in System and Information Science Notes, 1, pp. 62-69.
19. C. Scaffidi, M. Shaw and B.A. Myers (2005), *Estimating the numbers of end users and end user programmers*, in Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing; pp. 207-214; IEEE Computer Society, Washington, DC, USA.
20. R.T. Fielding (2000), *Architectural styles and the design of network-based software architectures*, in doctoral dissertation at University of California, Irvine.
21. IBM (2008), *Services sciences, management and engineering*, in Technical report, IBM, retrieved at http://www.research.ibm.com/ssme.
22. G. Toffetti, S. Comai, J.C. Preciado and M. Linaje (2011), *State-of-the Art and trends in the Systematic Development of Rich Internet Applications*, in Journal of Web Engineering, Rinton Press, 10, 1 March 15, 2011, pp. 070-086.
23. J. Wong and J. I. Hong (2007), *Making mashups with marmite: towards end-user programming for the web*, in Proceedings of the SIGCHI conference on Human factors in computing systems; 1435-1444; ACM, New York, NY, USA.
24. Z. Obrenovic and D. Gasevic (2009), *Mashing up oil and water: Combining heterogeneous services for diverse users*, in IEEE Internet Computing, 13, 6, pp. 56-64.
25. Open Mashup Alliance (2011), *Official web site*, retrieved at http://www.openmashup.org
26. J.H. Wu, Y.C. Chen and L.M. Lin (2007), *Empirical evaluation of the revised end user computing acceptance model*, in Computers in Human Behavior, 23, 1, pp. 162 – 174.
27. S. P. Jones, A. Blackwell and M. Burnett (2003), *A user-centred approach to functions in excel*, in Proceedings of the eighth ACM SIGPLAN international conference on Functional programming. Sweden, EU, ACM Press, pp. 165–176.
28. D. Lizcano, F. Alonso, J. Soriano and G. López (2011), *End-User Development Success Factors and their Application to Composite Web Development Environments*, in Proceedings of The Sixth

International Conference on Systems (ICONS 2011), St. Maarten, The Netherlands Antilles. Ed. IEEE Computer Society Press.

29. D. Lizcano, J. Soriano, R. Fernández, J. López and M. Reyes (2008), *Tackling interoperability in composite applications from an Enterprise Mash-up perspective*, in Proceedings of the 14th International Conference on Concurrent Enterprising (ICE 2008), Centre for Concurrent Enterprise, Nottingham University Business School, pp. 991-1000.
30. R. W. Floyd (1979), *The paradigms of programming*, in Commun ACM 22, 8, pp. 455-460.
31. Y. Deshpande, S. Murugesan, A. Ginige, S. Hansen, D. Schwabe, M. Gaedke and B. White (2002), *Web Engineering*, in the Journal of Web Engineering, Vol. 1, No. 1 (2002), Rinton Press.
32. A. I. Morch, G. Stevens, M. Won, M. Klann, Y. Dittrich and V. Wulf (2004), *Component-based technologies for end-user development*, in Commun. ACM 47, 9 (September 2004), pp. 59-62.
33. C. Pautasso and G. Alonso (2003), *Visual composition of web services*, in Proceedings of the Twelfth International World Wide Web Conference, pp. 92-99.
34. D. Lizcano, J. Soriano, M. Reyes and J.J. Hierro (2008), *EzWeb/FAST: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services*, in ACM Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2008, pp.15-24. ISBN 978-1-60558-349-5.
35. Networked European Software and Services Initiative (2011), *Official web site,* retrieved at http://www.nessi-europe.com
36. NESSI Open Framework – Reference ArchitecturePproject (2011), *Official web site*, retrieved at http://www.nexof-ra.eu
37. V. Wulf, V. Pipek and M. Won (2009), *Component-based tailorability: Enabling highly flexible software applications*, in International Journal of Human Computer Studies 66, 1, pp. 1-22.
38. D. Lizcano (2010), Formalisation of the End-User Software Development Paradigm, in doctoral dissertation at Universidad Politécnica de Madrid, Madrid, Spain.
39. U. Assmann (2003), *Invasive Software Composition*, in Springer-Verlag New York Inc.
40. J. Sametinger (1997), *Software Engineering with Reusable Components*, Ed. Springer.
41. X. Liang, I. Marmaridis, and A. Ginige (2007), *Facilitating Agile Model Driven Development and End-User Development for Evolving Web-based Workflow Applications*, in Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE '07). IEEE Computer Society, Washington, DC, USA, pp. 231-238
42. P.Freudenstein, M. Nussbaumer, F. Majer and M. Gaedke (2007), *A Workflow-Driven Approach for the Efficient Integration of Web Services in Portals*, in IEEE International Conference on Services Computing (SCC 2007) pp.410-417, 9-13 July 2007.
43. B. Christian and R. Christoph (2000), *A comparison of service discovery protocols and implementation of the service location protocol*, in Proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications, Technische Universität München (TUM).
44. L. Clement, A. Hately, C. von Riegen and T. Rogers (2004), *UDDI Version 3.0. 2*, in UDDI Spec Technical Committee Draft 10.
45. C. Peltz (2003), *Web services orchestration and choreography*, in Computer 36, 10, pp. 46-52.
46. T. Nestler, M. Feldmann, A. Preussner and A. Schill (2009), *Service Composition at the Presentation Layer using Web Service Annotations*, in First International Workshop on Lightweight Integration on the Web (ComposableWeb 09), pp. 63-68.
47. C. Pautasso (2004), *A Flexible System for Visual Service Composition*, in Ph.D. thesis, Swiss Federal Institute of Technology Zurich.
48. B. De Silva and A. Ginige (2007), *Meta-model to support end-user development of web based business information systems*, in Proceedings of the 7th international conference on Web

engineering (ICWE'07), Luciano Baresi, Piero Fraternali, and Geert-Jan Houben (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 248-253.
49. M. P. Papazoglou and D. Georgakopoulus (2003), *Service-oriented computing*, in Communications of the ACM; 46, 10, pp. 25-28.
50. Gartner Inc (2001), *Enterprise Applications - Adoption of E-Business and Document Technologies: 2000-2001 North America Executive Summary*, in Technical report on AIIM, BookStore and Gartner.
51. A. Heil, M. Gaedke and J. Meinecke (2009), *Modeling Resources in a Service-Oriented World*, in Proceedings of the 42nd Hawaii International Conference on System Sciences, pp. 1-10.
52. P. J. Molina, I. Torres, O. Pastor (2003), *User Interface Patterns for Object-Oriented Navigation*, in Upgrade, Human-Computer Interaction: Overcoming Barriers Vol. 4, issue 1, February 2003.
53. P. J. Morrison (2008), *Tagging and searching: Search retrieval effectiveness of folksonomies on the World Wide Web*, Information Processing & Management, 44, 4, pp. 1562-1579
54. D. McIlroy (1968), *Software Engineering*, report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968, retrieved at http://cm.bell-labs.com/cm/cs/who/doug/components.txt
55. J. Díaz, O. López and J. Fons (2001), *From User Requirements to User Interfaces: A Methodological Approach*, in Advanced Information Systems Engineering, Lecture Notes in Computer Science, 2068. Springer Berlin / Heidelberg. pp. 60-75.
56. S. Sendall and W. Kozaczynski (2003), *Model transformation: The heart and soul of model-driven software development*, in IEEE software 2003, pp. 42-45.
57. S. Ceri, F. Daniel, M. Matera and F. Facca (2007), *Model-driven development of context-aware web applications*, in ACM Transactions on Internet Technology (TOIT), 7, 1.
58. J. Rumbaugh, I. Jacobson and G. Booch (2004), *Unified Modeling Language Reference Manual*, in The 2nd Edition of Addison-Wesley Object Technology Series, Addison-Wesley Professional.
59. S. Ortiz-Chamorro, G. Rossi and D. Schwabe (2010), *Use of Hypermedia Tools for End-User Development*, in Current Trends in Web Engineering, Lecture Notes in Computer Science Vol. 6385, pp. 533-537.
60. N. Narayanan and R. Hbscher (1997), *Visual language theory: Towards a human-computer interaction perspective*, in K. Marriot & B. Meyer (Eds.) Visual Language Theory; pp. 85-127; Springer-Verlag.
61. B. Shneiderman (2003), *Promoting universal usability with multi-layer interface design*, in Proceedings of the 2003 conference on Universal usability, 1-8; ACM, New York, NY, USA.
62. M. Govindaraju (2003), *Merging the CCA component model with the OGSI framework*, in CCGrid03 Proceedings; volume 5(8); pp.182-189.
63. A. Fukunaga, W. Pree and T.D. Kimura (1993), *Functions as objects in a data flow based visual language*, in Proceedings of the 1993 ACM conference on Computer Science, pp. 215-220.
64. V. R. Aragao and A. A. Fernandes (2003), *Conflict resolution in web service federations*, in Proceedings of the International Conference on Web Services (ICWS-Europe 2003), volume 2853 of LNCS, pp. 109-122, Springer.
65. Open Mashup Alliance (2011), *Enterprise Mashup Markup Language (EMML)*, available at http://www.openmashup.org/omadocs/v1.0/index.html
66. E. Prud'Hommeaux and A. Seaborne (2006), *SPARQL query language for RDF*, in W3C working draft, 4.
67. D. Crockford (2006), *The application/json media type for JavaScript object notation (JSON)*, in RFC 4627, IETF.
68. C. Forgy (1982), *Rete: A fast algorithm for the many pattern/many object pattern matching problem*, in Artificial Intelligence, 19, 1, pp. 17-37.
69. EzWeb Project (2011*), Official web site*, available at http://ezweb.morfeo-project.org/lng/en

70. FAST Project (2011*), Official web site,* available at http://fast-fp7project.morfeo-project.org/lng/en
71. D. Lizcano (2011), *Description of the Problems Set to Evaluate the EUD Paradigm*, available at http://apolo.ls.fi.upm.es/eud/problems_description.pdf
72. EzWeb Catalogue (2011), *Catalogue of available end-user building blocks*, video available at http://ezweb.tid.es/ezweb/videos/catalogo/catalogo.htm
73. D. Lizcano (2011), *Description of the development process enacted by surveyed users*, available at http://apolo.ls.fi.upm.es/eud/solution_development_process.pdf
74. Demo EzWeb Project (2011*), Official demo web site*, available at http://demo.ezweb.morfeo-project.org
75. Demo FAST Project (2011*), Official demo web site*, available at http://demo.fast.morfeo-project.org
76. EzWeb/FAST Projects (2011), *10-minute user's manual of EzWeb/FAST*, see http://www.youtube.com/watch?v=qFt2LBlxkwU and http://www.youtube.com/watch?v=dpoRhnF8_1A
77. J. Soriano, D. Lizcano, M.Á. Cañas, M. Reyes and J.J. Hierro (2007), *Fostering innovation in a mashup-oriented enterprise 2.0 collaboration environment*, in System and Information Science Notes, 1, 1, pp. 62-69, SIWN Conf. Adaptive Business Systems, Chengdu, China.
78. Programmable Web (2011), *Official web site*, available at http://www.programmableweb.com
79. E.L. Lehmann and J. P. Romano (2006), *Testing Statistical Hypotheses*, Ed. Springer, New York.
80. D. Lizcano (2011), *Statistical Survey of the EUD Paradigm*, in online tech report available at http://apolo.ls.fi.upm.es/eud/eud_paradigm_evaluation.pdf
81. J. Bosch (1997), *Object-Oriented Frameworks – Problems and Experiences*, in Research Report Series, Blekinge Institute of Technology, issue 9.
82. V. Hoyer, A. Fuchsloch, S. Kramer, K Moller and J. López (2010), *Evaluation of the implementation*, in Technical Report D6.4.1, FAST Consortium February 2010. URL https://files.morfeo-project.org/fast/public/M24/D6.4.1_ScenarioEvaluation_M24_Final.pdf.
83. D. Lizcano (2011), *Statistical Survey of the End-User Development Paradigm*, available at http://apolo.ls.fi.upm.es/eud
84. C. Tejo-Alonso, S. Fernandez, D. Berrueta, L. Polo, M.J. Fernandez and V. Morlan (2010), *eZaragoza, a tourist promotional mashup*, retrieved at http://idi.fundacionctic.org/eZaragoza/ezaragoza.pdf
85. Andalusia's Regional Government (2011), *Official web site*, retrieved at http://www.juntadeandalucia.es/index.html
86. G. Fischer, K. Nakakoji and Y. Ye (2009), *Metadesign: Guidelines for Supporting Domain Experts in Software Development*, in IEEE Software, vol. 26, no. 5, pp. 37-44.
87. M. Erwig (2009), *Software Engineering for Spreadsheets*, in IEEE Software, vol.26, no.5, pp.25-30.
88. A. P. McAfee (2006), *Enterprise 2.0: The dawn of emergent collaboration*, in MIT Sloan Management Review, 47, 3, pp. 21–28.