Web-centred end-user component modelling

David Lizcano [a,*], Fernando Alonso [b], Javier Soriano [b], Genoveva López [b]

[a] *Universidad de Distancia de Madrid (UDIMA), Spain*
[b] *Universidad Politécnica de Madrid, Spain*

A B S T R A C T

This paper formally defines a web component model enabling end-user programmers to build component-based rich internet applications (RIAs) that are tailored to meet their particular needs. It is the product of a series of previously published papers. The formal definition in description logic verifies that the model is consistent and subsumes currently existing models. We demonstrate experimentally that it is more effective than the others.

Current tools propose very disparate web component models, which are based on the appropriate invocation of service backends, overlooking user needs in order to exploit these services and resources in a friendly manner. We have proposed a web model based on a detailed study of existing tools, their pros and cons, limitations and key success factors that have enabled other web end-user development (WEUD) solutions to help end-user programmers to build software to support their needs. In this paper we have verified that the proposed model subsumes and is instantiated by the models of the other existing tools that we analysed, coming a step closer to the standardization of end-user centred RIAs and development environments. We have implemented a development tool, called EzWeb, to produce RIAs that implement the proposed model. This tool enables users to develop their application following the model's component structure based on end-user programming success factors. We report a statistical experiment in which users develop increasingly complex web software using the EzWeb tool generating RIAs that conform to the proposed component model, and other WEUD tools generating RIAs that conform to other models. This experiment confirms the applicability of the proposed model and demonstrates that more end-user programmers (EUPs) (users concerned with programming primarily for personal rather public use) successfully develop web solutions for complex problems using the EzWeb tool that implements the model, which is more efficient than existing tools that implement other models.

1. Introduction

Interest and investment in web end-user development (WEUD) are mounting all the time, and its impact [1] has even outstripped forecasts made by Christopher Scaffidi, Brad Myers and Mary Shaw back in 2005 [2]. There are many web-based mashup development environments that enable millions of users to personally develop software solutions to solve their own problems.

Many software suppliers including Microsoft, Apple, IBM, Yahoo!, Oracle, etc., have developed tools providing support for end-user programmers (EUPs) (programmers who wish to achieve the result of a program primarily for personal rather public use) [1] to develop web applications, particularly rich internet applications (RIAs), offering do-it-yourself (DIY) [3] guidance on how to

* Corresponding author.
E-mail address: david.lizcano@udima.es (D. Lizcano).

evolve end-user developments to meet end-user demands and requirements. Such applications include Chrome Web Store (Chrome WS) and its Developer Tools [4], Yahoo! Pipes and Dapper [5,6], Microsoft Popfly [7] (currently closed and offered as part of Microsoft WebMatrix), Kapow Platform [8], JackBe Presto [9], AMICO Sketchify [10], Marmite [11] or EzWeb [12].

These solutions enable EUPs to develop their own software solutions. These solutions help EUPs create a graphical user interface (GUI) by visually connecting components with different levels of abstraction in order to access and exploit different types of services and resources and solve their particular problem. Each solution has pros and cons [3,1] and offers distinctive WEUD functionalities for creating end-user solutions. The major weakness is that each tool defines a web application development model for building solutions to problems of a particular type and complexity. These models are of no use for EUPs to develop more complex general-purpose RIAs [13,14]. For example, Yahoo! Pipes is confined to building mashups of data from RSS or HTML sources, whereas Kapow Platform specializes in building web portals using screen scraping techniques, and so on. The important thing, though, is that these WEUD solutions are promoting a new web component model [15] that has not yet, however, been either fully structured or formalized. The component models used in these tools, their strengths and generated products have not yet been studied in detail in order to define a comprehensive component model for the web. The race to compete in an increasingly globalized WEUD solutions ecosystem has forced developers (Google, Yahoo!, Microsoft, Amazon, Apple, Sun, IBM, etc.) to develop and optimize their own tools in their application environments without formalizing a common underlying component model. Therefore, a common component model needs to be built in order to promote interoperability between building blocks supplied by different manufacturers [16] and raise acceptance among EUPs by guaranteeing that users can successfully build more complex general-purpose RIAs than they can now [17].

The challenge, then, is to come up with an emerging web end-user component model [18] that covers the functionalities of a well-known set of existing tools, exploits their strengths and, whenever possible, reduces their weaknesses, encouraging EUPs to create and/or customize their own software [19]. This paper studies a representative set of existing tools, which were selected as being the most commonly used and successful tools in recent years, analyses the component models underlying the RIAs created using each tool and defines and formalizes in description logic a component model that subsumes the RIA models by merging their functionalities and strengths and incorporating EUD (end-user development) success factors. A WEUD tool that instantiates this model has been tested on real EUPs and found to more effectively scale up to increasingly complex problems than today's EUD tools. We designed this tool, called EzWeb [20], along with other partners under the auspices of a Networked European Software and Service Initiative (NESSI) strategic research project. EzWeb is now being used in two European Union 7th Framework Programme projects in which we are participating: 4CaaSt [21] (building the future Platform as a Service) as part of its mashup-as-a-service solution and FI-WARE [22] (building the Future Internet core platform) as part of its applications and services ecosystem and delivery framework's generic enablers for EUPs to build application mashups.

The remainder of the paper is structured as follows. Section 2 presents related work and analyses the principal WEUD tools and the component models governing the end-user solutions that they can each build. Section 3 presents a set of target features for an end-user oriented component model and presents our WEUD component model that combines the strengths of the other models with EUD success factors that we have analysed during our research. This model has been mathematically formalized in Section 4 using formal logic to demonstrate that it is consistent and is instantiated by the models produced by the analysed WEUD tools. Section 5 describes the use of an automatic reasoning tool to check whether the component model generated by each tool described in Section 2 is a valid instance of the global model reported in Section 3. Section 6 presents the results of a study that we conducted to test whether EzWeb, which generates RIAs that conform to the proposed component model, achieves better results than other WEUD tools, which generate RIAs that conform to other models. Section 7 addresses the EUD dilemma of whether it is better to define generic or domain-specific EUD tools. Finally, Section 8 concludes this paper and presents a brief outline of future work.

2. Related work: existing solutions for end-user development

Software suppliers are in the process of converting their products into web services (an approach termed Software as a Service, SaaS), and all sorts of software solutions are readily available in the shape of services scattered over the Internet [23]. These approaches target end users that are generally unfamiliar with the details of the technology used to implement services. Users should now be just as able to use these services to their own advantage as they used to be able to use commercial software products in the past [24]. There are compilations of available services, together with examples, guidelines and success stories in service use, including the Programmable Web repository [25]. Programming knowledge, knowledge of SOAP, WSDL, BPEL, etc., is required to use these resources [26]. This breach between the high availability of web resources and the low prospects of their use by EUPs has led many large software enterprises to create mashup development environments targeting EUPs like Chrome WS and its Developer Tools, Yahoo! Pipes and Dapper, Microsoft Popfly, Kapow Platform, JackBe, AMICO, Marmite or EzWeb. They all share the goal of enabling EUPs to develop a composite web application that solves their particular problem.

The major problem with these tools is that EUPs are often unable to translate their particular requirements into a specific software product [1,17], because each tool focuses on achieving a particular solution type that does not necessarily meet user needs. For example, Yahoo! Pipes creates a correctly filtered data list feed, Kapow Platform creates an execution flow based on pre-existing interlinked web portals, and so on. Users who require a more complex RIA or need to solve a problem type other than for which the tool was designed will be disappointed.

Our working hypothesis is that the component models controlling the different WEUD tool solutions are not general enough to be able to create more complex general-purpose RIAs. Additionally, the tools do not match the way in which EUPs conceive their solution; nor do they offer a natural development process for end-user characteristics and needs. This hypothesis is based on the study of many related papers focusing on the EUD field and applicable to WEUD, which are described below.

End-user development or EUD is a term first proposed by European researchers ten years ago at an international symposium held in Bonn, Germany. It has attracted a lot of scientific interest since the first biannual International Symposium on End-User Development (IS-EUD) focusing on this domain was held in 2007. Four top-level meetings have been organized since then. The main topic of these conferences is how to empower EUPs to develop and adapt systems themselves.

These symposiums, together with other international congresses, have promoted several lines of EUD-related investigation akin to the research reported in this paper: (1) attempts at simple programming languages or environments focused on a particular domain, such as EnglishMash (an end user-oriented language

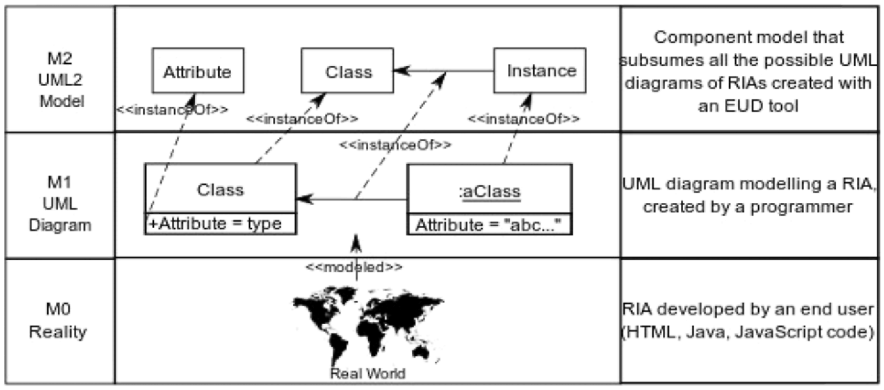

Fig. 1. OMG modelling levels.

with massive natural language use and proactive aids such as auto-completing code) designed by Aghaee and Pautasso [27], or ResEval (an EUD environment for mixing R&D domain data sources using a XML-based visual language, which requires knowledge of mark-up languages) created by Muhammad et al. [28]; (2) demonstration of the potential of wizard-supported services integration in the EUD field [29,30]; (3) research into human–computer interaction and its particularities in the EUD field, such as [31] stressing how important it is to have user-oriented building blocks in order to focus on solution design rather than the interaction with programming components, or Grudin [32] reporting a study suggesting that HCI consistency is not relevant in the EUD field and can be counterproductive because it hems in user creativity.

All these papers point to the potential of a visual environment based on correctly catalogued building blocks described simply in natural language, which are easy to use, interoperable and useful for building component-based end-user software applications. Such an environment would be able to take advantage of the major benefits of component-based software, although the problems troubling this type of approaches also need to be addressed. These problems are described by Garlan et al. [33], Stiemerling [34] and Wulf et al. [35], highlighting the need for component types with a good trade-off between ease of use and usage, and between generality and specificity. These researchers draw attention to the conflict within end-user off-the-shelf software: if the products and components are generic enough to be used by most users, then they will never be perfectly tailorable to their particular needs and vice versa. Our research addresses this problem directly within the EUD field.

Several researchers have already addressed EUD conceived on the basis of off-the-shelf parts, including Mørch [36] who came up with the idea of using basic architectural patterns in order to implement end-user software using Lego brick-like user-oriented building blocks. Although Mørch's paper did not focus on the web components world, it was the first to deal with building blocks as they are used in the most successful WEUD tools today. Mørch et al. [37] later identified the need to create RIAs to support routine work by EUPs, although they included other roles in the development process, such as, for example, advanced programmers to assemble and configure the parts used. This is a job that EUPs could do themselves if they had access to the right mechanisms and simple, general-purpose but at the same time fully functional, component models, as illustrated by the success of some of today's WEUD tools [17,38].

These tools can be used to build mashup web applications, which have proven to be very useful in the business field as a services integration alternative to rigid SOA architectures, as illustrated by papers like [39,29,30]. There is to date, however, no comparative study analysing the component models implemented by each WEUD tool or proposing a general component model subsuming their functionalities. Such a model would enable EUPs to address problems of increasing complexity that can only be solved with a wider variety of component types and relationships to interrelate components. This is a well-known weakness in this web field: all WEUD component models are missing a gentle slope of complexity [40], that is, when the complexity of the RIA to be built increases, the performance of the development tool plummets. WEUD will not thrive unless end users are provided with models for complex component-based development.

Our working hypothesis is founded on considering that a broad-spectrum component model designed for non-programmers and implemented by a web tool will enable EUPs to build complex and general-purpose RIAs more successfully than the models implemented by today's EUD tools that do not enable users to develop really complex RIAs. To do this, we have studied the component models driving EUD in the RIAs generated by these tools in order to analyse their features, strengths and weaknesses. Alongside other EUD success factors that we have analysed during our research, they have been used to define a general model that adopts all of these strengths.

According to Object Management Group principles, there are three modelling levels in the software engineering world. They are illustrated in Fig. 1.

WEUD tools are useful for building a RIA application that uses source code in more than one programming language to perform a function. This final application is equivalent to the M0 modelling level: reality. That M0 is the RIA created by a EUP for her own needs, with specific information inputs and outputs.

The M1 level includes the UML (Unified Modeling Language) diagrams that represent the component model to which a RIA built by the end user conforms. M1 is the diagram modelling a RIA, that is, a specific WEUD platform considering all different widgets, operators, etc., that this platform provides. In the example of Yahoo! Pipes, M1 will include all the sources, like "Fetch CSV", and user inputs, like "URL input", etc. The M2 level defines the UML2 diagrams that represent the general component model implemented in any product generated by a particular WEUD tool. Section 2 presents these M2 diagrams in order to explore what type of solutions each tool is capable of building in theory.

We propose a bigger model subsuming the other M2 models of other WEUD platforms. It uses UML2 and MOF (Meta-Object Facility) to propose component models that subsume the M2 models. Our M2 model is the underlying component model for different WEUD platforms, defining the associations between all different objects including the "workspace", "widget", "input", "output", etc. So, instances of "object" like "workspace" and instances of "association" including all those UML associations in the component models can be defined at the M2 level. This M2 model is presented in Section 3 as a general EUD model subsuming the existing models. The EzWeb tool presented in Section 6 is now

being used to develop composite applications that conform to this model. This confirms the strengths of a tool that outputs RIAs that are instances of this model.

We now describe the existing WEUD tools and rate the pros and cons, uncovered by an in-depth analysis at all levels: M2 component model, development tool UI, repository, services, composition techniques, etc.

2.1. From iGoogle to Chrome Web Store: independent resource mashup platforms

One of the first mashup development and execution platforms targeting EUPs was iGoogle. iGoogle enabled users to create a personalized Internet home page composed of a mashup of off-the-shelf items (originally called widgets and later apps) organized by tabs or workspaces. This project was retired, but its premises and principles were kept alive by the Chrome Platform and by the iGoogle Portal[1] site.

The Chrome Platform is a browser-based environment which provides, via the Chrome Web Store [4], applications, extensions and web components for browser-based execution. Users have access to a comprehensive catalogue of visual elements, including applications, data sources or extensions. These components can be placed on a blank page where these apps are launched using representative icons. Working in Chrome Browser Developer Mode, users can use Chrome DevTools to inspect and adapt components, and even modify their operation and appearance. Developer mode requires knowledge of HTML and JavaScript. Code Labs is designed to simplify these steps, offering tutorial-like textual guidance. Even so users have need of basic web programming knowledge and have to use a programmer-targeted IDE called Dev Editor. On this ground, the iGoogle Portal was launched to cater for EUPs. The iGoogle Portal maintains the original iGoogle framework. The iGoogle Portal is independent of Google Chrome and the Google project.

As iGoogle sparked an explosion of WEUD solutions, such as Yahoo! Pipes and Dapper, Kapow Platform, AMICO, etc., that set out to emulate iGoogle's simplicity and success, we present the approach here. Widgets are independent interface elements with atomic and defined functionality. Multiple suppliers have used traditional web programming (based primarily on JavaScript) to create a great many widgets implementing all sorts of functionality: e-mail boxes, to-do lists, calendars, clocks, weather forecasts, multi-language translation, games, notebooks, multimedia players, etc. All these widgets generally implement their functionalities by invoking remote web services, although EUPs are completely unaware of this, as the widget is like a black box for users who are unfamiliar with its internal implementation. Fig. 2 illustrates the iGoogle component model [3], which is maintained by iGoogle Portal.

The model underlying the Chrome Web Store and apps deployment and adaptation targeting end users with HTML and JavaScript knowledge also conforms to the above model, as it includes the components of the general model specified in this paper. The approach is designed to build a web execution platform composed of user-selected apps. The platform may have one or more workspaces, each of which is composed of several user-selected apps (which can be tailored, although this requires knowledge of HTML and JavaScript). These apps are the main element underlying the Chrome approach, and they are composed of an interface (with a Manifesto or XML meta-information and a visual interface) and backend resources (including a background

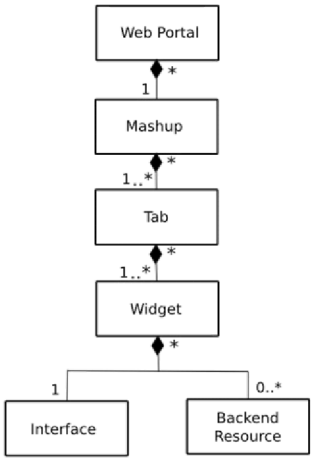

Fig. 2. iGoogle (and iGoogle Portal) component model.

script) that users indirectly manage through the widget (for example, to invoke weather forecast web services, email server access, map visualization, etc.).

iGoogle (iGoogle Portal) and Chrome Web Store have serious weaknesses with respect to WEUD, as their approach is rather simplistic and limited:

- The final solution should be a blank page in the browser listing apps and its development is subject to numerous constraints.
- The widgets published in the catalogues are limited, and EUPs cannot develop new visual elements because this calls for some programming knowledge. Therefore, developers have to stick to available off-the-shelf apps adapted to their needs.
- Apps are independent functional elements and are not at all interoperable or intercommunicable. EUPs may want to develop apps that cooperate with each other (for example, to import and directly translate a news source using the translator widget or use a map widget to visualize a location received in a message read by an e-mail widget, etc.). The WEUD solutions that can be created using iGoogle Portal or Chrome Web Store are limited in this respect. EUPs that want their solutions to have such functionalities will have to use another type of WEUD environment.

The major strength of the approach based on iGoogle is the idea of using apps: visual parameterizable elements designed to perform a function in the global RIA. This is an easy idea for EUPs to understand and which they can use to do things as far apart as invoking web services to displaying heterogeneous data homogeneously. According to previous studies, this idea enjoys widespread acceptance among EUPs [17], and is, therefore, adopted as a key element of our model for use by EUPs to generate their RIA. But being isolated elements, widgets do not serve our purpose, as this limits the complexity and generality of the solutions built enormously.

2.2. Yahoo! Pipes and Dapper: heterogeneous data source mashup platforms

Many users' Internet experience is based on gathering and reading portal contents, information, news and opinion. In this respect, many web sites publish all sorts of contents, news and information as RSS feeds (enriched web site summary) that users can receive easily. This technology was so successful that EUPs often needed tailored solutions to filter, sort or prioritize the syndicated information that they received from the web. Yahoo! Pipes [6] and Yahoo! Dapper [5] emerged in response to the end-user need for this type of solution. These tools enable EUPs to filter,

[1] iGoogle Portal, http://www.igoogleportal.com.

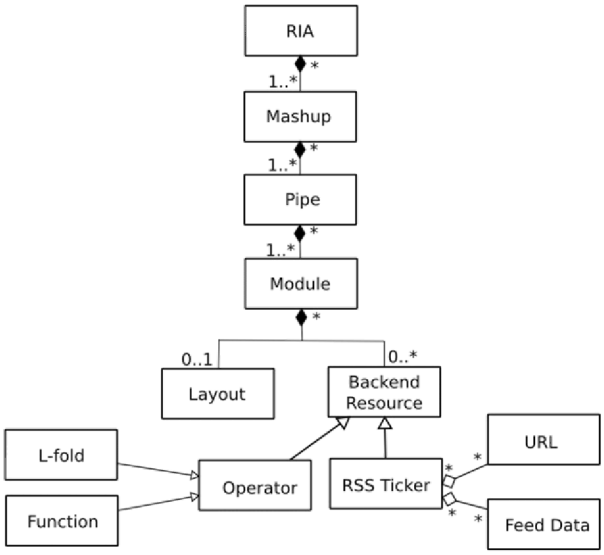

Fig. 3. Yahoo! Pipes and/or Yahoo! Dapper component model.

mix, organize, label and, in short, manage web data sources. Fig. 3 shows the Yahoo! Pipes and Dapper component model.

EUPs use the Yahoo! platform to develop a RIA composed of one or more pipe mashups. Each mashup will be composed of the concatenation of one or more pipes that perform specific functions (filter contents based on defined criteria, mix data sources, sort results according to a specific criterion, etc.) and implement a dataflow. Each pipe is composed of one or more modules. A module is an atomic component performing a specific function like a black box. The tool contains different types of catalogued modules (data sources, data operators, data filters, etc.) which users can link to generate a new pipe. The modules are fully generic at design time, and users convert them into special-purpose tools through instantiation. Each module has a graphical interface called layout (optional) and several backend resources which are accessed unbeknown to users (resource invocation is not always necessary). It is these backend resources that implement the RIA business logic, transparently created for EUPs. The business logic can be implemented by data operators (functional or list operations) or by invoking remote data sources. Feed invocation is composed of the source URL and the data transiting from the remote web source to the importing module.

Yahoo! WEUD tools enable EUPs to manage diverse Internet data sources, but do not solve the problems of how to enable EUPs to generate final solutions with complex functionalities. Using these tools users can manage and adapt data sources to their needs, but they cannot easily manage heterogeneous services scattered across the Internet in a user-centred manner, as the tools are unable to invoke general web services. This is their major shortcoming.

However, the idea of creating a dataflow among operators and components linking typed inputs and outputs proved to be very popular among EUPs in a previous study that we conducted [3]. Input and output types are useful for providing users with recommendations and feedback about what connections can be created among the components that they are using. Our model applies this idea of dataflow among components and operators.

2.3. Kapow Platform: mashing up visual interfaces and service frontends

There is a whole series of WEUD solutions focusing on the integration of visual web elements with very similar purposes and

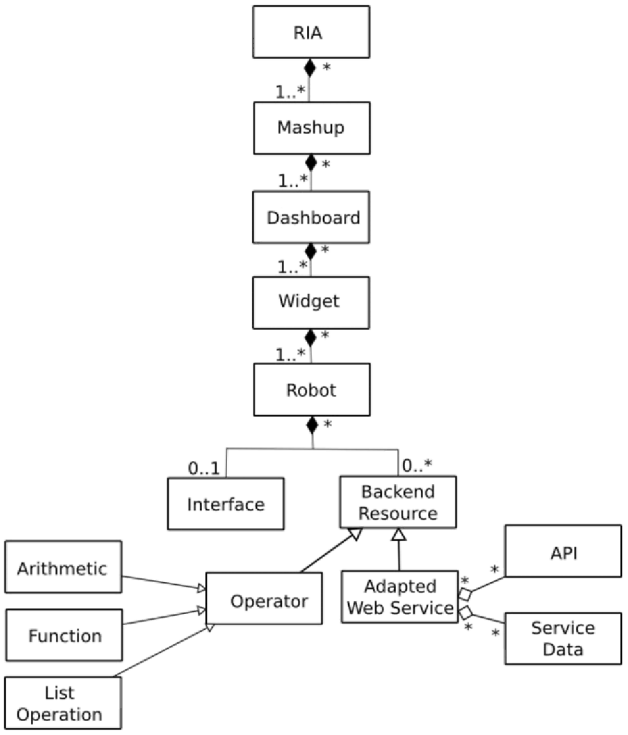

Fig. 4. Kapow Platform software component model.

characteristics: Kapow Platform, JackBe Presto, AMICO Sketchify, Marmite, etc. They are all designed to enable EUPs to develop their own RIA based on one or more mashups of visual elements organized in tabs or workspaces: a very similar approach to the iGoogle Portal homepage. In this case, however, they provide for cooperation and communication and data import/export between visual elements.

In the following, we discuss the architectural schema underlying the RIAs created with Kapow Platform only, as component models generated by EUPs using the above solutions are all equivalent. Fig. 4 illustrates the component model of the Kapow Platform tool.

RIAs built with Kapow Platform have one or more mashups of virtual elements of different types. For a better understanding of this tool, Fig. 5 shows the framework architecture.

Each mashup is composed of one or more dashboards that play the role of offering different runtime workspaces, called Kapow KappZones, for users by way of sets of separate widgets, created with a tool called Design Studio. There are several off-the-shelf dashboards published in a catalogue called Kapplets Library. Each dashboard is composed of one or more interlinkable widgets, although the widgets of different dashboards are not interconnectable. Widgets are the basic visual element that will make up the final solutions created using Kapow Platform, and no knowledge of web technologies is required for their use because they use APIs set up to act as middleware with business services. Each widget is implemented by one or more robots (also known as roboservers) cooperating with each other, and the management, use and configuration of these robots requires some knowledge of web programming using the management console. A robot is a functional part that performs a specific computational task. It is composed of an interface (hidden and not displayed at run time) and backend resources that execute workflows of the final solution without EUPs being able to perceive all its implementation details. These resources can run operations on data (arithmetic, functional or list operations) or invoke remote web services through a dedicated API using the Kapow integration engine. For a robot to

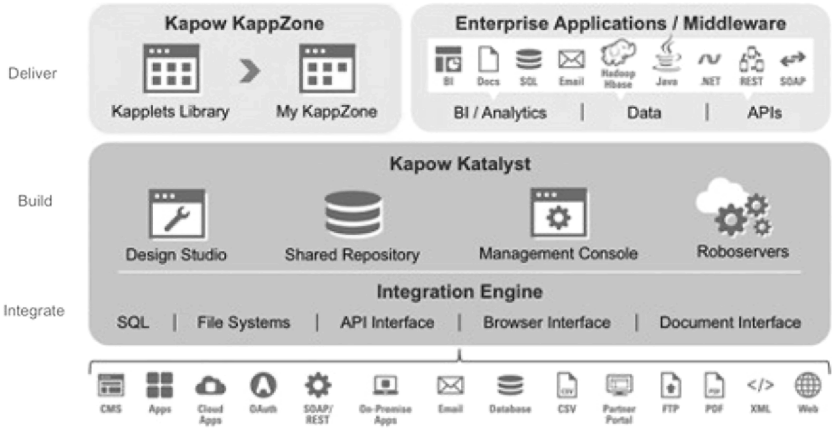

Fig. 5. Kapow Platform layers and tools [8].

invoke a service, the off-the-shelf service must be built in such a manner that details how to manage the service API (syntax and technology), and sent and received data must be specified in its implementation. Therefore EUPs can never modify the robot to manage any other web services.

These solutions contribute three concepts that have been adopted in our model:

- Relationships of composition among different elements: a RIA is composed of one or more mashups, made up in turn of more detailed components and so on. This component hierarchy organized by level of abstraction enables users to create (or adapt) a component not included in the tool from its constituent parts.
- Spatial and temporal grouping components: these tools include elements for spatially combining diverse interrelated components, leading to the concept of workspace or dashboard, which is a great facility for creating complex applications. They also include elements for creating widgets (used atomically by other tools), such as the sequential execution of parts called robots as a temporal flow.
- Use of UI design elements in order to compose each widget. This way, users can form new widgets by combining interface elements (forms, buttons, etc.), data operators and backend services that ultimately wrap the invocation of a remote service or resource so that users do not have to bother with a request–response protocol.

As regards the weaknesses of these tools, several studies, including Rode et al. [13], Rosson et al. [14], Ko et al. [1], or Lizcano et al. [3,17], suggest that a much earlier version of Kapow Platform (with similar applicability but worse usability) and a long list of similar existing solutions signified a major step forward towards the goals set out by the WEUD community but do not have all the features required in order to enable EUPs to develop their own general-purpose web applications at leisure:

- EUPs still have limited use of web services available on the Internet. They will only be able to use services for which there are predesigned robots (or equivalent design elements) that offer a suitable frontend for EUPs to manage this service in a shared repository. EUPs will not have access to the other (most) services.
- When there are no off-the-shelf components that meet user needs, the design detail level has to be increased to the point where users are required to manage robots, something that, as already mentioned, calls for web programming knowledge. This means that EUPs will not find a guided design and implementation process partitioned into stages adapted to end-user experience or that has a level of abstraction tailored to end-user knowledge and their way of understanding software development [41].
- They appear to account for widget interconnection, but this requires some knowledge of data structures and typing, and again EUPs can find it hard to use this development solution.

These weaknesses can be mitigated by offering users visual techniques and components that they can use to create new robots (or robot-level components) that they can link using the dataflow and component interconnection ideas proper to other tools like Yahoo!

2.4. Other non-web EUD approaches

Studies conducted by Rode et al. [13] and Ko et al. [1] reveal that no web EUD tool has yet managed to repeat the success stories reported in other EUD fields like spreadsheets, despite the enormous functional potential of the Internet as an ecosystem of web services that can be exploited, orchestrated and syndicated to generate really powerful composite web applications. They report that most EUPs are unable to use existing WEUD tools to create a RIA that satisfies a straightforward requirements specification stated by such users. The validation reported in this paper also shows up these deficiencies.

As regards the explanation of this failure, studies carried out by Davis et al. [42], Wu et al. [41], and other authors [38] disclose that the problems with EUD support web tools can be traced back to the component models, environments and techniques of the tools used to build solutions on the grounds explained above. The above component models control the development process, its stages and activities [16]. Therefore, these models should be tailored to end-user needs and take into account that EUPs may want to build more general-purpose RIAs than they can do using each one separately. Component models should be consistent, prevent errors and assure quality. And no EUD tool supplier has yet made such an effort to define and formalize a general-purpose model valid for creating rather diverse or complex RIAs.

In [38] we studied spreadsheet-based EUD tool strengths. Based on these strengths and the weaknesses of existing EUD and WEUD tools, together with other investigated EUD success factors, we propose a new generic component model for RIAs created by EUPs. This model takes into account correct component abstraction, parameterization, component interaction, etc. This model should:

- Be a consistent component model, like the models of the analysed WEUD tools.
- Subsume existing models so that any existing WEUD tool can be tailored to the proposed model and tool components can be exploited generically by a tool that conforms to the proposed model.
- Perform better in experiments conducted with EUPs than existing tools.

3. EUD component model for the web

There are several studies [41,42] on the factors determining whether or not a particular EUD will be successful with EUPs. However, almost all focus on spreadsheet-based EUD tools. Very few studies to date have focused on web-centred EUD tools for building RIAs, the most prominent being [15,16,13,14]. They propose improvements to existing EUD component models, some focusing on HCI improvements, such as tool user perception, error proneness, viscosity to change or correct component abstraction, and yet others on component improvement to cater for the trade-offs between component specificity and generality for use in different domains. Based on the presented research works, we describe a set of premises (success factors) that we consider a good WEUD model should have, which were partially reported in [38, 43]:

1. Any component of an end-user solution should be a black box that performs a specific and precise function (that is, call a service, invoke a resource, etc.) that makes problem-solving sense to the user. At the same time, a rich and expressive visual interface should make such components manageable, simple and understandable and be clearly described in natural language. In fact, users should be able to understand the components that they use and grasp what they do without having to bother about how they do it.
2. The runtime component will usually process some input data to produce outputs. Users should be able to communicate the dataflow between the components underlying the task to be performed. Non-programmers need to have access to abstractions tailored to their mental pattern in order to model this dataflow. Simple data together with a visual representation of the semantic compatibility among these data constitute the right level of abstraction. These data can be considered as pre- and postconditions that drive the execution of a state machine [3]. This saves users from having to deal with the syntax of the backend resources. Users should also have the option of specifying the meaning of such data. This would be helpful for people using the elements in the future.
3. Users should have access to mechanisms for both spatially and temporally managing the dataflow. Users should be able to formulate changes to the interfaces/visualizations depending on particular data, management processes, etc.
4. Finally, a very important EUD success factor (and one of the secrets behind the spreadsheet sensation) is the abstraction gradient. Not all users have the same knowledge of compositional aspects, technical expertise or experience in EUD fields. Instead of programming a RIA or component, which they are not qualified to do, users should parameterize off-the-shelf components to meet their needs, or put together finer-grained parts to visually compose more abstract, original and useful components. A catalogue of off-the-shelf components for composing new components should offer a full-blown hierarchy of components, ranging from comprehensive, complex and problem domain-specific RIAs to simple services, data and/or resources wrapped by software providers for use by less expert users. We propose a component hierarchy formed by final solutions (full-blown RIAs), mashups, workspaces, widgets, visual items, data operators and finally backend resource wrappings. Throughout our research, this hierarchy has proved to be good enough to meet the abstraction needs of most EUPs.

Apart from premises concerning the component model, our research work revealed other assumptions that would be advantageous for EUD that we intend to address as future lines of research, such as:

1. The development tool should include some sort of visual aid, such as a user interface with help based on colour coding or highlighted items to suggest data flows among elements depending on the data types that they manage for users at design time. This would help EUPs to solve their problem and reduce process viscosity.

   The feedback that we get after reviewing studies on this topic [41,42,15] is that EUD success factors are related to HCI factors and the specialization–functionality relationship.
2. EUD components should be published in a collaborative and federated solution component marketplace. The development environment should have access to this marketplace to download and publish components. Software providers, which opted for SaaS years ago, can use this marketplace to publish business resources duly packaged according to end-user requirements. This principle would encourage new users to publish their solutions and reuse earlier components and elements built by other users. It would also reduce the difficulty curve for new developments [44] and produce an exponential benefit, known in economics as network externality.

None of the premises are recognized requirements; they are a set of good practices and desirable elements uncovered by our research in this field over the last ten years. The only way to empirically check the adequacy of each of these premises is to test whether a tool that generates RIAs implementing a model that accounts for these premises is successful or fails.

With these premises, we created a web component model that was more successful among EUPs because it adopted the above factors [45] and a tool that enabled users to create RIAs according to this model [46].

The main contribution of this paper is to *model* the proposed EUD web component model [45] in description logic. Description logic was used to check model consistency and confirm that the analysed WEUD models are an instance of the proposed model that subsumes the models generated by existing tools that we have studied and which is more efficient than the other models at addressing complex problems in the WEUD field. We state two aims to be achieved separately. The first aim is to adapt elements of the other tools to the more general model that we propose, thereby increasing their usefulness and existing EUD software reuse. The second aim is to demonstrate that a EUD tool producing RIAs that conform to the proposed model behaves more efficiently in response to problems of increasing complexity than the analysed existing tools which produce RIAs conforming to other models. The model presented below was designed to comply with the above premises and is capable of providing support for the development environments that set out to use joint catalogues and end-user recommendation techniques. Fig. 6 illustrates the proposed M2 component model using UML2 [43,45], which needs to be correctly specified and validated. For this purpose, it has been described in formal logic. This formal logic description, which is the groundwork of the contribution of this paper, demonstrates the consistency and validity of the component model and that it subsumes and instantiates the other analysed models.

We employ a UML2 class diagram that conforms to the UML2 superstructure specification defined in ISO/IEC DIS 19505-2. We use Meta Object Facility (MOF) Core Specification to create our M2 diagram [47]. MOF is a facility defined and used in ISO/IEC 19502:2005. The international standard describes its importance and applicability in model-driven engineering, enabling the creation of a bigger M2-level schema and offering the possibility of running or checking schema instances or subsumptions in UML notation [48].

As the model shows, the design element is the basic component of the component model. This element is composed of a user-centred visual interface for accessing a wrapped resource. Any

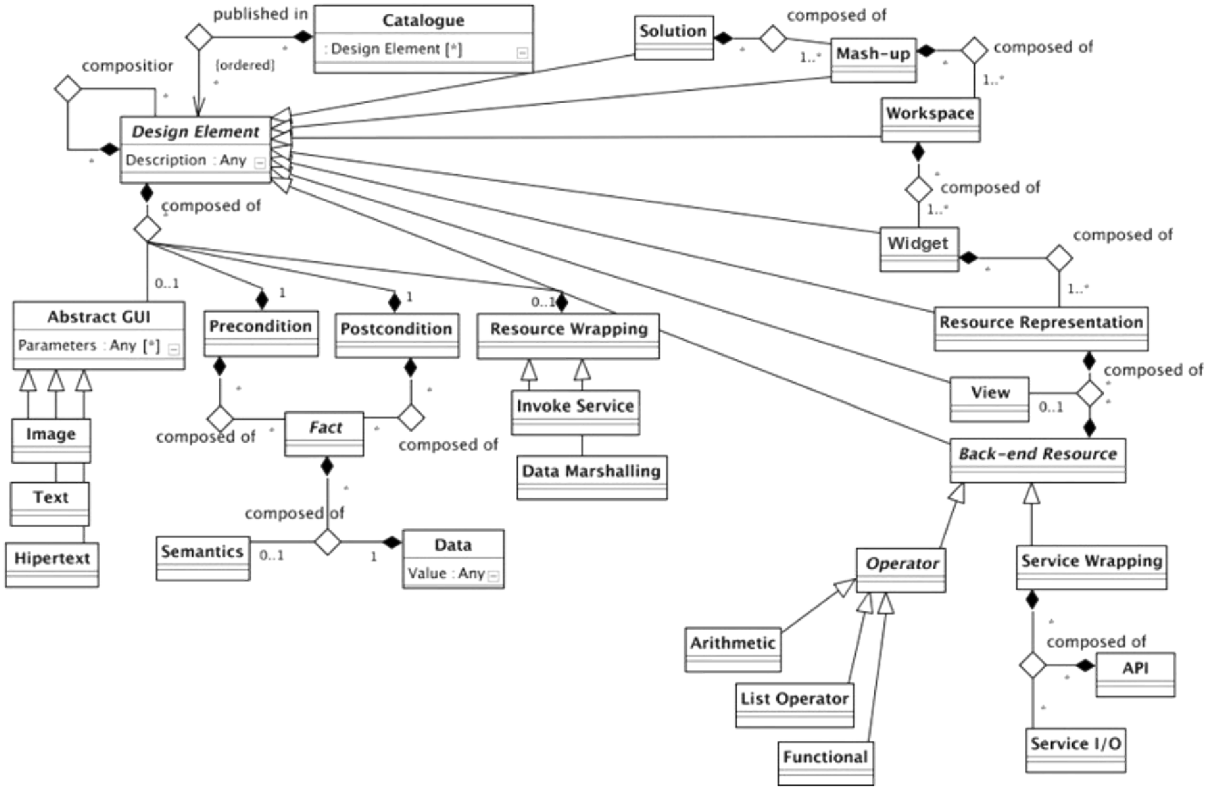

Fig. 6. Proposed WEUD component model.

component will be linked with other components in the final solution through pre- and postconditions based on facts that guide the dataflow, where a fact is an information item composed of a datum and its associated lightweight semantics. As already mentioned, visual aids (such as coloured or highlighted elements) suggest valid data flows to users at design time thanks to a reasoning model based on component ontologies, lightweight labels created by other users and/or suppliers and the data types and elements already used previously in the actual design.

Ideally in the future all the end-user components should be published in a business marketplace-style collaborative and federated catalogue. This obviously requires massive user participation where a critical mass of users download, reuse and update components. This is not immediately applicable in the course of research into WEUD models. In any case, any user will be able to search the catalogue for new components and compose solutions sourced from other user recommendations about the data managed by the partially designed solution, etc.

Finally, the components should cater for the needs of as many EUPs as possible and be adapted to the levels of abstraction that we can expect to find depending on user problem knowledge and previous experience at developing RIAs. For this reason, there is a full-scale hierarchy of design elements devised to fit the level of abstraction required by users for different development process workflows. These levels of abstraction include anything from full solutions to backend resources (simple data operators, like filters, concatenators, etc., or wrapped services). Each element in this hierarchy is adapted to a different level of abstraction. This way, users can find and focus on the detail level that they expect to find and with which they are confident: the full solution (or RIA) is equivalent to the system that the user envisages to address the problem; this solution is composed of a mashup of several design elements, and has several workspaces. Workspaces are visual spaces all displayed at the same time by a composite interface that aims to tackle part of the problem. These workspaces include several interconnected widgets, where a widget is a visual element that manages user interaction with a particular remote resource. This widget may present a single view or a screen flow (such as a survey composed of several forms) for the user to interact with the remote resource or resources associated with the widget. Each of these visual interaction items is termed resource representation. A resource representation is composed of the view and the backend resource. The backend resource is composed of operators and service wrappings.

To illustrate the component model components, suppose that a user wants to run an Amazon product search and list the results on screen (see Fig. 7). To do this, the search flow or screen flow will be composed of several successive widgets leading the user to access the Amazon backend. The user will enter the product that he or she wants to search in the first widget. The second widget will list all the products found. If the user selects a product, a third widget will display the product details. The first widget will be a form where the user enters the search data. These data will flow to the second widget, where they will access the Amazon product search backend resource. Operators are available to order, filter, etc., the results. Finally, the facts suggest and implement data flows among components, enabling an item list form to receive all the products offered by Amazon for the established search criterion. Fig. 7 shows for example a solution created with EzWeb for the described problem.

This component model is useful for establishing a dataflow among visual elements where a new data item in one component leads all the collaborative interfaces to take a computational step. This approach bears some resemblance to data flows among cells in spreadsheets, save that each element displays a richer visual interface and invokes particular remote services, resources or distributed data as wrapped services.

Service wrappings are the atomic design elements of our component model; they are the smallest pieces that EUPs can handle and understand. These elements, composed of an API

Fig. 7. Example of visual components with linked data.

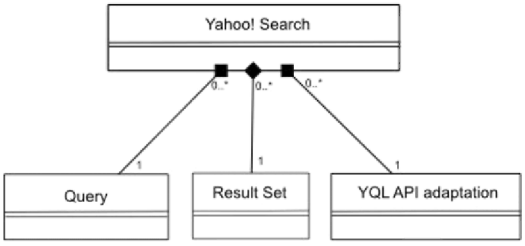

Fig. 8. Yahoo! search service wrapping in UML which instantiates part of the proposed model.

and some inputs and outputs, are especially abundant on the Internet thanks to web service ecosystems, as such web services are really easy to transform into wrapped service components. There follows an example of a web service using the Yahoo! search engine, transformed into a user-centred component. This example illustrates how such a web service can be wrapped. Fig. 8 is a simplification of the wrapped service described in UML as a specific instance of our component modelling language.

The wrapped Yahoo! Search service includes the Yahoo! Query Language (YQL) API adaptation, a data input, which, in this case, is a web search query, and the output, which is a result set. This resource will be associated with a visual interface, which will depend on the WEUD tool on which it is run, and a natural language description. Fig. 9 illustrates the XML wrapping necessary to define the web service input and output as the pre- and postconditions of the end-user component.

Additionally, when the component precondition is satisfied, the web service has to be sure to be invoked and the results returned by the service should satisfy postconditions that are meaningful in the WEUD field. This means developing a small service adapter according to a traditional development process using a language such as JavaScript, for example (see Fig. 10).

Using the component model RIAs can be created by composing visual building blocks. This abstracts users from the invocation of resources and services that constitute the RIA backend, an approach already explored by Obrenovic and Gasevic [16].

The proposed model has been formalized to check that it achieves the research aims, that is, the high-level components have a consistent architecture composed of bottom-level component aggregations, and the created components instantiate the component models of the above EUD web tools. Additionally, we have conducted a statistical study to check whether the model is adequate and demonstrate that, implemented as a WEUD tool that produces RIAs that are instances of the model, it is more successful than the component models implemented by the above tools used by EUPs. The following sections address these topics.

4. Mathematical formalization of the WEUD metamodel by means of description logic

Description logic-based inference tools, like RacerPro [49], are a good option for checking model consistency and confirming that the analysed WEUD models (located at level M2) are an instance of the proposed model. This is not a straightforward comparison even though the proposed model takes into account the strengths and weaknesses of the analysed tools. To do this, the proposed component model is formalized by means of ad hoc description logic used to reason about whether the model is consistent and whether other consistent models analysed and translated to the same description logic are an instance of the proposed model.

RACER software, used in conjunction with eMOFLON [50], is able to validate UML2 models (M2), inspect their consistency based on reference models, and check whether a UML model (M1 or M2) is an instance of another more general model [51]. To compare UML models, these models have to be formalized by means of a description logic-based mathematical description so that instance verification operations can be performed on the model [52]. By mapping UML2 to description logic (similarly to the work reported by Wang and Jin [53]), it is possible to use the above automatic reasoning systems to check that the proposed WEUD model achieves the research aims, that is, it is consistent and the existing WEUD tools models are an instance of the proposed model. Table 1 sets out the concept and role constructors used to develop the proposed mapping in order to set up a sufficiently expressive type of description logic. Additionally, it indicates the computational complexity associated with the two fundamental operations targeted by the mathematical reasoning: subsumption ($\models C \subseteq D$) and instance checking ($\models C(i)$).

Table 1 shows that, according to the naming scheme defined by Baader et al. [52], we use $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-}{}^{+\mathrm{o}}$ description logic [54,55]. The naming scheme consists of assigning a letter or symbol to each expressive extension of elementary attributive language (AL) logic. These elements can appear in different forms:

- A letter after AL to add a concept constructor
- A superindex to add a role constructor
- A subindex to add a constraint on role interpretation.

The following explanations are useful for understanding the description logic analysis. Four basic concept constructors are added to traditional description logic: qualified existential (ε), negation (C), cardinality (N) and enumeration (O). The qualified existential constructor is necessary to infer whether a more abstract component contains a finite set of less abstract elements that meet a particular condition. This is useful for modelling components built from other components. The negation constructor is useful for negating particular model assertions, placing constraints on component models (for example, that a low-level component cannot

Table 1
Constructors of concepts and roles used in the proposed mapping.

Constructor	Syntax	Semantics	Logic type	Compl. $\models C \sqsubseteq D$	Compl. $\models C(i)$
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$\mathcal{FL}_0$	P	P
Domain	T	$\Delta^{\mathcal{I}}$			
Empty	$\bot$	$\emptyset$			
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$			
Universal	$\forall R.C$	$\{x \mid \forall y : R^{\mathcal{I}}(x,y) \rightarrow C^{\mathcal{I}}(y)\}$			
Existential	$\exists R.T$	$\{x \mid \exists y : R^{\mathcal{I}}(x,y)\}$	$\mathcal{FL}^-$	P	P
Atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$	$\mathcal{AL}$	P	P
Qualified existential	$\exists R.C$	$\{x \mid \exists y : R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}(y)\}$	$\mathcal{E}$	NP	PSPACE
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	$\mathcal{C}$	PSPACE	PSPACE
Enumeration	$a_1 \ldots a_n$	$a_1^{\mathcal{I}} \ldots a_n^{\mathcal{I}}$	$\mathcal{O}$	PSPACE	PSPACE
Disjunction	$C \cup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$\mathcal{U}$		
Cardinality	$\geq nR$	$\{x \mid \#\{y \mid R^{\mathcal{I}}(x,y)\} \geq n\}$	$\mathcal{N}$		
	$\leq nR$	$\{x \mid \#\{y \mid R^{\mathcal{I}}(x,y)\} \leq n\}$			
	$= nR$	$\{x \mid \#\{y \mid R^{\mathcal{I}}(x,y)\} = n\}$			
Qualified cardinality	$\geq nR.C$	$\{x \mid \#\{y \mid R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}(y)\} \geq n\}$	$\mathcal{Q}$	EXP	EXP
	$\leq nR.C$	$\{x \mid \#\{y \mid R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}(y)\} \leq n\}$			
	$= nR.C$	$\{x \mid \#\{y \mid R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}(y)\} = n\}$			
Selection	$f : C$	$\{x \in Dom(f^{\mathcal{I}}) \mid C^{\mathcal{I}}(f^{\mathcal{I}}(x))\}$	$R_{\mathcal{F}}$		
Transit. roles	R^+	$\bigcup_{n \geq 1} (R^{\mathcal{I}})^n$	$()^+$		
Inverse roles	R^-	$\{(y,x) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(a,b)\}$	$()_{R^-}$		
Role composition	$R \circ S$	$R^{\mathcal{I}} \circ S^{\mathcal{I}} = \{(x,z) \mid \exists y \in \Delta^{\mathcal{I}} \cdot R^{\mathcal{I}}(x,y) \wedge S^{\mathcal{I}}(y,z)\}$	$()_{R^\circ}$		

```
<?xml version="1.0" encoding="UTF-8"?>
<resource-adapter endpoint-url="http://search.yahooapis.com" endpoint-service
name="/WebSearchService/V1/">
...
<method name=" webSearch" precondition name="keyword" type="text" label="ServiceHired"
friendcode="service">
<parameter name="query" type="xsd:string" type-qualifier="xsi:type">
&lt;%=query_to_search%&gt;
</parameter>
<result update- postcondition ="search-suggestion" type="text" label="deviceId" friendcode="deviceId"/>
</method>
...
</resource-adapter>
```

Fig. 9. XML resource adapter that defines the mapping of WEUD pre-/postconditions to the Yahoo web search service parameters.

```
function setKeyword(string){
...
}
var keyword_to_search = EzWebAPI.createPreconditionFact("text", keyword);
...
document.getElementById('keyword').data=keyword_to_search.get();
var suggestion = EzWebAPI.PostFact ("keyword");
...
suggestion.set("example text");
...
var currentSuggest = suggestion.get();
```

Fig. 10. JavaScript service adapter. The variables declared in the adaptation have to be previously declared in code, casting types and programming the remote invocation.

be composed of high-level elements). The cardinality constructor is useful for creating conceptual relationships with cardinality, providing an option for an element to participate in an association with one or more other elements. This is essential for modelling a high-level component that can be composed of $0 \ldots n$ elements of the next level of abstraction. The enumeration constructor is useful for enumerating valid elements for a particular component, such as, for example, facets of a basic data type or visual taxonomies of valid elements for a component.

On top of these basic elements, there is a complex constructor, qualified cardinality ($\mathcal{Q}$), with the restriction of having role hierarchies read from left to right ($^-{}_H$), the capability for reflexive relationships ($_R{}^+$) and annotated by means of nominals ($^\circ$). This element is necessary because of the type of inference to be made using this description logic. The inference engine will create hierarchical instance rules weighted by the number of matches between the two input models: a UML and a UML2 model. The roles will be dual – instance-of and subsumption-of – and may be reflexive. Nominals have to be used in order to use enumerated classes of object value restrictions, such as owl:oneOf, owl:hasValue, which are able to model the web ontologies used in WEUD tools.

The choice of this description logic is a trade-off between the expressiveness of the language used to construct the terminological information and the complexity associated with the reasoning processes on both terminological and assertive model information [51]. In this respect, the trade-off for using other types of logic, such as the family of description logics derived from DLR logic that remove the binary role constraint and introduce n-ary role constructors, which would have enabled a more straightforward mapping than proposed in this paper, would be a much greater computational cost. Note that tools like RACERPro have been unable to classify a UML2 model expressed in DLR_{reg}, that is, the DLR logic extension with union, composition and transitive closure of binary role constructors as a projection of n-ary roles on two of its components.

Apart from the traditional conceptual subsumption ($C \subseteq D$) and equivalence ($C \equiv D$) axioms, constrained in the sense that only D can be an expression of concept (and therefore C must be

an atomic concept), the subsumption axiom has also been used in order to create role hierarchies in relations, hence subindex H. The aim of the construction- and axiom-level $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-\ +\mathrm{o}}$ mapping is to specify how this description logic captures the semantics of mapped design elements.

The formal semantics of the UML2 model mapped to $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-\ +\mathrm{o}}$ is based on a model theory where $I = \langle D, \cdot^I \rangle$ is an interpretation, and:

- $D \neq \emptyset$ represents a domain or universe of discourse such that $D = \Sigma \cup \Upsilon$ with $\Upsilon = \bigcup_{i=1}^{n} \Upsilon_{Di}$, $\Upsilon_{Di} \cap \Upsilon_{Dj} = \emptyset$ and $\Sigma \cap \Upsilon = \emptyset$, and Σ is the domain of managed components and Υ_{Di} is the set of values associated with each basic data type D_i supported by UML2 (highlighting the free type any, although other types, such as integer, string, etc. can be specified)
- $\cdot^I$ is the projected interpretation function:
  - ${D^I}_i = \Upsilon_{Di}$
  - $C_i^I \subseteq \Sigma$
  - ${A^I}_i \subseteq \Sigma \times \Upsilon$
  - $R_i^I \subseteq \Sigma \times \cdots \times \Sigma \equiv \Sigma^n$

To illustrate the proposed mapping, we give an incremental description of the UML2 component model translation to the proposed description logic.

4.1. Model class mapping

A class described in UML2 (such as, for example, the class that describes a type of WEUD component) denotes a set of objects with common characteristics in terms of properties and associations and is therefore represented as a concept C in $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-\ +\mathrm{o}}$ description logic.

4.2. Generalization hierarchy mapping

A generalization or inheritance relationship among two classes described in UML2 specifies that each instance of the *child* class is also an instance of the *parent* class. The instances of the child class inherit the parent class properties (and satisfy other additional properties) and can participate in their associations. The UML2 generalization relationship is expressed as an inheritance among $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-\ +\mathrm{o}}$ concepts taking advantage of the fact that the semantics of inclusion statements ($C_i \sqsubseteq C_j$) are based on set inclusion and respect the concept of replacement. The UML2 generalization relationship accounts for four types of situations:

- Partial and non-disjoint: The meaning of this constraint is $C_i^I \subseteq C^I,\ i = 1, \ldots, n$, which can be translated into first order logic as

  $$\forall x \cdot C_i(x) \rightarrow C(x), \quad i = 1, \ldots, n.$$

  This constraint can be expressed in $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-\ +\mathrm{o}}$ description logic as $C_i \sqsubseteq C,\ i = 1, \ldots, n$.
- Partial and disjoint: The meaning of this constraint is

  $$C_i^I \subseteq C^I, \quad i = 1, \ldots, n$$
  $$C_i^I \cap C_j^I = \emptyset, \quad i = 1, \ldots, n$$

  which can be translated into first order logic as

  $$\forall x \cdot C_i(x) \rightarrow C(x) - \bigwedge_{j=i+1}^{n} C_j(x), \quad i = 1, \ldots, n.$$

  This constraint can be expressed in $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-\ +\mathrm{o}}$ description logic as

  $$C_i \sqsubseteq C, \quad i = 1, \ldots, n$$
  $$C_i \sqsubseteq C_j, \quad \text{for all } i \neq j.$$
- Total and non-disjoint: The meaning of this constraint is

  $$C_i^I \subseteq C^I, \quad i = 1, \ldots, n$$
  $$C^I \subseteq \bigcup_{i=1}^{n} C_i^I,$$

  which can be translated to first order logic as

  $$\forall x \cdot C_i(x) \rightarrow C(x), \quad i = 1, \ldots, n$$
  $$\forall x \cdot C(x) \rightarrow \bigvee_{i=1}^{n} C_i(x).$$

  This constraint can be expressed in $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-\ +\mathrm{o}}$ description logic as

  $$C_i \sqsubseteq C, \quad i = 1, \ldots, n$$
  $$C \sqsubseteq \bigsqcup_{i=1}^{n} C_i.$$
- Total and disjoint: This is the most commonly used type, because the end-user component taxonomy is intrinsic and this type enriches the semantics of the developed WEUD model. The meaning of this constraint is:

  $$C_i^I \subseteq C^I, \quad i = 1, \ldots, n$$
  $$C_i^I \cap C_j^I = \emptyset, \quad \text{for all } i \neq j$$
  $$C^I \subseteq \bigcup_{i=1}^{n} C_i^I,$$

  which can be translated to first order logic as

  $$\forall x \cdot C_i(x) \rightarrow \bigvee_{i=1}^{n} C_i(x)$$
  $$\forall x \cdot C_i(x) \rightarrow C(x) - \bigwedge_{j=i+1}^{n} \neg C_j(x).$$

  This constraint can be expressed in $\mathrm{ALC}\varepsilon\mathrm{NOQ}_{\mathrm{HR}}^{-\ +\mathrm{o}}$ description logic as

  $$C_i \sqsubseteq C, \quad i = 1, \ldots, n$$
  $$C_i \sqsubseteq \neg C_j, \quad \text{for all } i \neq j$$
  $$C \sqsubseteq \bigsqcup_{i=1}^{n} C_i.$$

This formalization also captures the generalization relationship among UML2 associations (classes).

The following is an example of the formalization of a total and disjoint generalization relationship among the concepts of backend resource, operator and wrapped service. The UML2 model is able to formally express the described generalization relationship types by means of a very similar graphical notation to what is used in UML.

$$\text{BackendResource} \sqsubseteq \text{Class} \sqcap (\text{Operator} \sqcup \text{WrappedService})$$
$$\text{Operator} \sqsubseteq \text{Class} \sqcap \text{BackendResource} \sqcap \text{WrappedService}$$
$$\text{WrappedService} \sqsubseteq \text{Class} \sqcap \text{BackendResource} \sqcap \text{Operator}$$

4.3. Class attribute mapping

The constraint imposed by assigning an attribute A with data type D to a class C is

$$C^I \subseteq \{x \in \Sigma \mid \#(A^I \cap (\{x\} x \Upsilon_D)) \geq 1\},$$

which can be translated into first order logic as

$$\forall x \cdot C(x) \rightarrow \exists y \cdot A(x, y) - D(y).$$

This constraint can be expressed in $\mathcal{ALC}\varepsilon\mathcal{NOQ}_{\mathcal{HR}^+}^{-+o}$ description logic as

$C \sqsubseteq \langle = 1A\rangle \sqcap \exists A.D$ for single-valued attributes,

$C \sqsubseteq \langle \geq 1A\rangle \sqcap \forall A.D$ for type $A[]$ attributes, and

$C \sqsubseteq \langle \geq n_i A\rangle \sqcap \langle \leq n_j A\rangle \sqcap \forall A.D$

for attributes with cardinality $(n_i \ldots n_j)$,

where C is a concept, A is a binary role and D is a data type. The UML2 mapping to description logic developed in this paper is also able to express the semantics of simple key attributes, which is not strictly necessary at the descriptive level required by programming paradigm metaphors, but would use the following scheme:

$C \sqsubseteq \langle = 1A\rangle \sqcap \exists A.D \sqcap \langle \leq 1A^-\rangle$.

The semantics of a compound key would be defined by a relationship $R : C \rightarrow (A_1 \times A_2 \times \cdots \times A_n)$ such that R is injective and R^- is functional. On this ground, it would have to be possible to consider the relationship $(A_1 \times A_2 \times \cdots \times A_n)$ as a concept such that its instances could constitute the range of relationship R.

Note that hardly any of the components of the developed WEUD model have parameters because UML2 is such a high-level modelling language. The properties emerge in the UML model that describes the instances of components in each development environment. The following is the mapping of the *any* type design element class attribute (*any* is a non-constrained type commonly used in UML2. This mapping serves the purpose of indicating that a conceptual level parameter is required in the model's UML instantiations).

$\text{DesignElement} \sqsubseteq \text{Class} \sqcap \text{Description.any} \sqcap \langle \leq 1\text{Description}\rangle$.

4.4. Associations and class dependency mapping

The constraint imposed by the interrelation of n classes $C_1 \cdots C_n$ by means of an association R is

$R^I \subseteq C_1^I \times \cdots \times C_n^I$,

which can be translated into first order logic as

$\forall x_1, \ldots, x_n \quad \cdot R(x_1, \ldots, x_n) \rightarrow C_1(x) - \cdots - C_n(x)$.

This constraint can be expressed in description logic by means of an n-ary role or by transforming the association R into a concept A and n roles $r_1, \ldots, r_n$. This option is useful for representing properties and qualifiers associated with this association by means of new roles such as are described in Section 4.3. Additionally, this option conforms to UML2 semantics determining that associations should not be managed as inverse relations with references associated with each participant class but as a different object that is associated with references to participant classes:

$A \sqsubseteq \exists r_1 \cdot C_1 \sqcap \cdots \sqcap \exists r_n \cdot C_n \sqcap \langle \leq 1r_1\rangle \sqcap \cdots \sqcap \langle \leq 1r_n\rangle$.

As an UML2 association is (as demonstrated by the MOFLON tool) a specialization of a UML2 class, there may be, apart from properties whose domain is an association, generalization relations among associations. These are treated as inheritance relations among concepts. Unlike an association, a UML2 dependency can only be binary and has no roles. A dependency can be expressed by means of a binary role.

4.5. Associations and class dependency mapping

UML2 demands the association of cardinalities to association roles. This is another constraint per role that can be expressed as:

$C_i^I \subseteq \{x \in \Sigma \,|\, m_i \leq \#(R^I \cap (\Sigma x\{x\}x\Sigma)) \geq n_i\} \quad i = 1, \ldots, n, m$

being the lower bound on the cardinality of association R, and n the upper bound. This expression can be translated into first order logic as:

$\forall x_i \cdot C(x_i) \rightarrow \exists^{\geq p} x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \cdot R(x_1, \ldots, x_n)$
$\quad - \exists^{\leq p} x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \cdot R(x_1, \ldots, x_n)$

with

$\exists^{\leq n} x \cdot R(x, y)$
$\quad \equiv \forall x_1, \ldots, x_n, x_{n+1} \cdot R(x_1, y) - \cdots - R(x_n, y) - R(x_{n+1}, y)$
$\quad \rightarrow (x_1 = x_2) \vee \cdots \vee (x_1 = x_n) \vee (x_1 = x_{n+1})$
$\quad\quad \vee (x_2 = x_3) \vee \cdots \vee (x_2 = x_n) \vee (x_2 = x_{n+1})$
$\quad\quad \vee \cdots \vee (x_n = x_{n+1})$

and

$\exists^{\geq n} x \cdot R(x, y)$
$\quad \equiv \exists x_1, \ldots, x_n \cdot R(x_1, y) - \cdots - R(x_n, y)$
$\quad\quad - \neg(x_1 = x_2) - \cdots - \neg(x_1 = x_n)$
$\quad\quad \neg(x_2 = x_3) - \cdots - \neg(x_2 = x_n)$
$\quad\quad - \cdots - (x_{n-1} = x_n)$.

Therefore, an n-ary association with cardinalities $(k_i \ldots l_i)$ can be expressed in description logic as

$A \sqsubseteq \exists r_1 \cdot C_1 \sqcap \cdots \sqcap \exists r_n \cdot C_n \sqcap \langle \leq 1r_1\rangle \sqcap \cdots \sqcap \langle \leq 1r_n\rangle$

$C_i \sqsubseteq \forall r_i^- \cdot A \sqcap \langle \geq k_i r_i^-\rangle \sqcap \langle \leq l_i r_i^-\rangle, \quad i = 1, \ldots, n.$

Note that generally cardinalities can only be defined for non-transitive roles that do not have any transitive subroles. This is a constraint that is intrinsic not to the description logic used but to the associated deductive reasoning system.

4.6. Association navigability expression mapping

In the case of a binary association, a new role A_r is introduced to restrict the cardinality to the concept that acts as the domain and specify the navigability of the association:

$A_r \equiv r_1^- \circ r_2$

$C_1 \sqsubseteq \forall A_r \cdot C_2 \sqcap \langle \geq m_i A_r\rangle \sqcap \langle \leq m_j A_r\rangle$

$C_2 \sqsubseteq \forall A_r^- \cdot C_2 \sqcap \langle \geq m_i A_r^-\rangle \sqcap \langle \leq m_j A_r^-\rangle$

$A \sqsubseteq \exists r_1 \cdot C_1 \sqcap \exists r_2 \cdot C_2 \sqcap \langle \leq 1r_1\rangle \sqcap \langle \leq 1r_2\rangle$

$C_1 \sqsubseteq \forall r_1^- \cdot A \sqcap \langle \geq m_i r_1^-\rangle \sqcap \langle \leq m_j r_1^-\rangle$

$C_2 \sqsubseteq \forall r_2^- \cdot A \sqcap \langle \geq n_i r_2^-\rangle \sqcap \langle \leq n_j r_2^-\rangle$.

The role A_r is also able to specify the transitivity of an association (note that r_1 and r_2 are not transitive roles). The mapping of the *composition* aggregation is shown below. It illustrates the part of the description logic that translates this aggregation only. The example also shows the roleComposition role to specify the navigability of the *composition* association. UML2 and its support tools (like RACERPro) specify this same semantics formally by means of a naming rule for references of the respective association class. The model does not include a binary association for which reason no example is given.

$\text{DesignElement} \sqsubseteq \text{Class} \sqcap \forall\, \text{roleComposition.DesignElement} \sqcap$
$\quad \forall\, \text{group\_Composition}^-\text{.DesignElement} \sqcap$
$\quad \forall\, \text{part\_Composition}^-\text{.DesignElement} \sqcap$

$\text{Composition} \sqsubseteq \text{Aggregation} \sqcap$
$\quad \exists\, \text{composition\_DesignElement.Group} \sqcap$
$\quad \exists\, \text{composition\_DesignElementPart} \sqcap$
$\quad \langle \leq 1\text{composition\_Group}\rangle \sqcap \langle \leq 1\text{composition\_Part}\rangle$

$\text{roleComposition} \equiv \text{composition\_Group}^- \circ \text{composition\_Part}.$

Table 2
Summary of the UML2 to ALCεNOQ$_{HR}^{-}{}^{+o}$ mapping.

UML2 element	New concepts and roles	New DL axioms
Class C	Concept C	
Attribute a of C with type T	Binary role a	$C \sqsubseteq \langle = 1a \rangle \sqcap \exists a.T$
Attribute a of C acting as keyword	Binary role a	$C \sqsubseteq \langle = 1A \rangle \sqcap \exists A.D \sqcap \langle \leq 1A^{-} \rangle$
Attribute a of C with type $T[]$	Binary role a	$C \sqsubseteq \langle \geq 1a \rangle \sqcap \forall a.T$
Attribute a with associated card. $(n_i \ldots n_j)$	Binary role a	$C \sqsubseteq \langle \geq n_i a \rangle \sqcap \langle \leq n_j a \rangle \sqcap \forall a.T$ $T \sqsubseteq \forall A.C_2 \sqcap \forall A^{-}.C_1$
Dependency A	Binary role A Roles R_1 and R_2	$C_1 \sqsubseteq \forall A.C_2 \sqcap [\geq n_i A] \sqcap [\leq n_j A]$ $C_2 \sqsubseteq \forall A^{-}.C_1 \sqcap [\geq m_i A^{-}] \sqcap [\leq m_j A^{-}]$ $A \sqsubseteq R_1, R_1 \sqsubseteq A, A^{-} \sqsubseteq R_2, R_2 \sqsubseteq A^{-}$
n-ary association with multiplicity	Concept A Roles $A_r, R_1 \ldots R_n$	$A \sqsubseteq \exists R_1.C_1 \sqcap \cdots \sqcap \exists R_n.C_n \sqcap$ $\langle \leq 1R_1 \rangle \sqcap \cdots \sqcap \langle \leq 1R_n \rangle$ $C_i \sqsubseteq \forall R_i^{-}.A \sqcap \langle \geq n_i R_i^{-} \rangle \sqcap \langle \leq n_j R_i^{-} \rangle$ $i = 1, \ldots, n$
Binary association with multiplicity	Concept A Roles $A_r, R_1 \ldots R_n$	$A \sqsubseteq \exists R_1.C_1 \sqcap \exists R_2.C_2$ $\sqcap \langle \leq 1R_1 \rangle \sqcap \langle \leq 1R_2 \rangle$ $C_1 \sqsubseteq \forall R_1^{-}.A \sqcap \langle \geq m_i R_1^{-} \rangle \sqcap \langle \leq m_j R_1^{-} \rangle$ $C_2 \sqsubseteq \forall R_2^{-}.A \sqcap \langle \geq n_i R_2^{-} \rangle \sqcap \langle \leq n_j R_2^{-} \rangle$ $A_r \equiv R_1^{-} \circ R_2$ $C_1 \sqsubseteq \forall A_r.C_2 \sqcap \langle \geq m_i A_r \rangle \sqcap \langle \leq m_j A_r \rangle$ $C_2 \sqsubseteq \forall A_r^{-}.C_2 \sqcap \langle \geq m_i A_r^{-} \rangle \sqcap \langle \leq m_j A_r \rangle$
Non-disjoint partial inheritance relation		$C_i \sqsubseteq C,\ i = 1, \ldots, n$
Disjoint partial inheritance relation		$C_i \sqsubseteq C,\ i = 1, \ldots, n$ $C_i \sqsubseteq \neg C$, for all $i \neq j$
Non-disjoint total inheritance relation		$C_i \sqsubseteq C,\ i = 1, \ldots, n$ $C \sqsubseteq \bigcup_{i=1}^{n} C_i$
Disjoint total inheritance relation		$C_i \sqsubseteq C,\ i = 1, \ldots, n$ $C_i \sqsubseteq \neg C$, for all $i \neq j$ $C \sqsubseteq \bigcup_{i=1}^{n} C_i$

4.7. UML2 keyword/qualifier mapping

UML2 modelling may include stereotypes or keywords between brackets before a class to characterize that class. The methodological criterion for mapping UML2 qualifiers is: a new concept is created if the differentiation between the two concepts has specific implications for their relations to other concepts or places constraints on other properties in other concepts. Otherwise, it is preferable to use a property to express this differentiation.

The proposed WEUD model does have qualifiers and they must be stated in description logic. The only qualifiers that require the creation of new concepts are *aggregation* and *aggregate*. The others can be expressed by means of properties associated with concepts covered by the qualifier.

Table 2 summarizes the entire UML2-ALCεNOQ$_{HR}^{-}{}^{+o}$ mapping. Appendix A shows the WEUD model originally created in UML2 and translated to ALCεNOQ$_{HR}^{-}{}^{+o}$ according to the reported translation process.

5. Validation of model completeness with respect to the studied solutions

The presented UML2-ALCεNOQ$_{HR}^{-}{}^{+o}$ mapping is a semantic formalization of the UML2 conceptualizations for the performance of automatic reasoning services, such as checking that a model is an instance of another model, if both have been previously expressed in description logic. The knowledge base semantics is equivalent to a set of first-order predicate logic axioms. Therefore, like any other set of axioms, it contains implicit knowledge that can be specified through logical inference. The fundamental inference service is the verification of consistency for assertive knowledge bases (ABox) in terms of which the other models can be expressed.

The proposed UML2 model mapping is the terminological knowledge base (TBox), that is, entails the construction of a terminology T. Both ABox and TBox are concepts explained at length in [52,51]. To validate the proposed UML2 model, we have to discover whether each class and/or relation makes sense or, otherwise, whether it contradicts the remainder of the model, in which case it will never be able to be instantiated. From the logical viewpoint, a new concept C makes sense if there is at least one interpretation I that satisfies the axioms of T and for which the concept denotes a non-empty set. This interpretation is called model and is written $T \models C$. This property C with respect to T is called satisfiability. We have used the reasoning services offered by the RACERPro as a DL subsystem to verify that all the classes, relations, cardinalities and characteristics of the proposed WEUD model satisfy the studied WEUD component models and that they satisfy the expected generalization/specialization relations. The first step was to use description logic to codify the proposed WEUD model in the ABox and TBox and then check whether the model is consistent and if all the WEUD models of the studied tools are a valid instance of our general model.

All the reasoning services are based on the following prototype services:

- Satisfiability: A concept C is satisfiable with respect to a terminology T if there exists a model I of T such that $C^I \neq \emptyset$. It is written $T \models C$. The satisfiability of T is expressed as $T \models$.
- Subsumption: A concept C is subsumed by a concept D with respect to T if $C^I \subseteq D^I$ for any model I of T. It is written $T \models C \sqsubseteq D$. Subsumption can be expressed in terms of satisfiability as $T \models C \sqsubseteq D \Leftrightarrow T \not\models C \sqcap \neg D$. Similarly, satisfiability can be expressed in terms of subsumption as $T \not\models C \Leftrightarrow T \models C \sqsubseteq \bot$.
- Equivalence: A concept C is equivalent to a concept D with respect to T if $C^I = D^I$ for any model I of T. It is written $T \models C$ *equiv* D. Equivalence can be expressed in terms of satisfiability

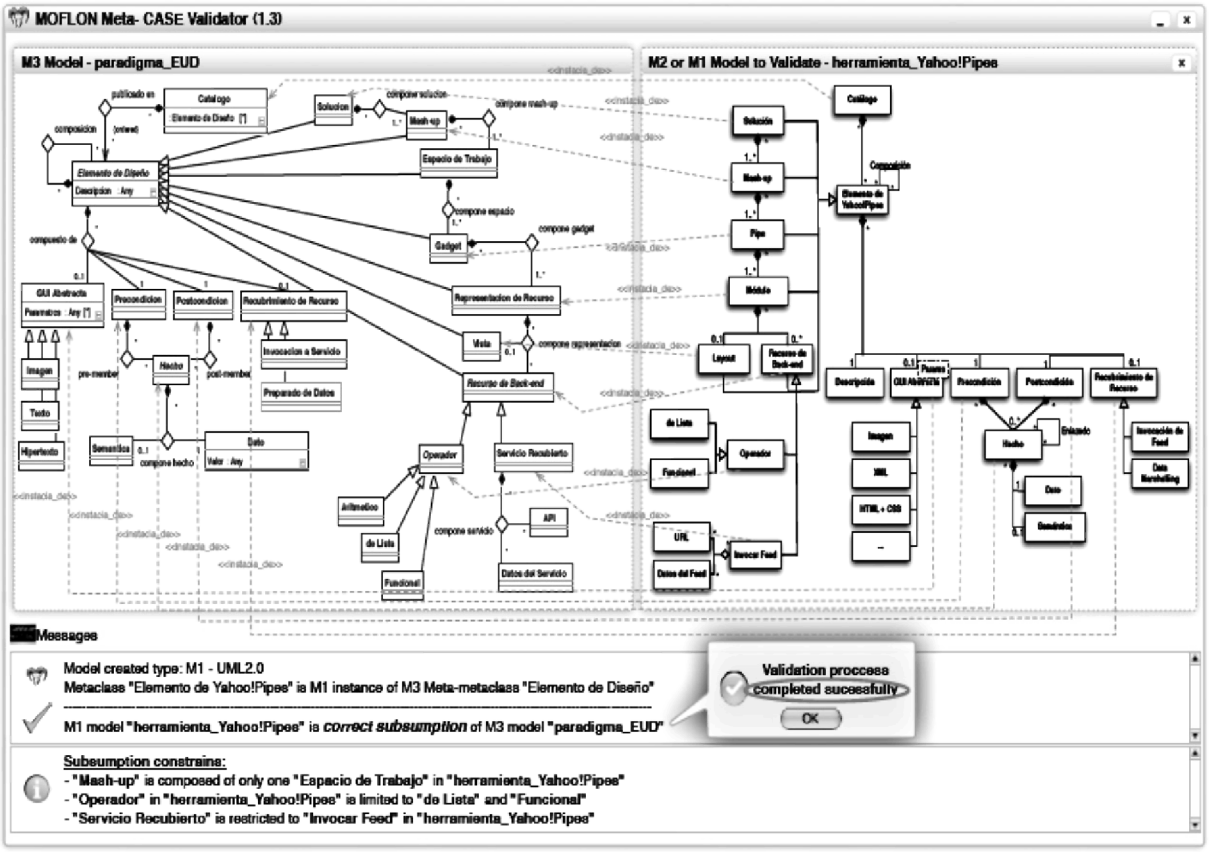

Fig. 11. Process and results of checking whether Yahoo!Pipes and Dapper is an instance of the proposed model using MOFLON.

as $T \models C \equiv D \Leftrightarrow T \not\models C \sqcap \neg D$ and $T \not\models \neg C \sqcap D$ and in terms of subsumption as $T \models C \equiv D \Leftrightarrow T \models C \sqsubseteq D$ and $T \models D \sqsubseteq C$.

- Disjunction: Two concepts C and D are disjoint with respect to T if $C^I \cap D^I = \emptyset$ for any model I of T. Disjunction can be expressed in terms of satisfiability as $T \not\models C \sqcap D$ and in terms of subsumption as $T \models C \sqcap D \sqsubseteq \bot$.

As a result, the RACERPro inference showed that the proposed model and the analysed models of WEUD tools are **consistent**, the proposed model is **unambiguous**, there are transformations that can be used to check whether other UML2 models are equivalent to our reference model, and these transformations are **computable in finite time**. Consequently, we will be able to computationally check whether the other analysed WEUD models are an instance of the general model described in this paper. The RACERPro tool [49] can be executed using the mapping described in Appendix A to confirm these claims.

To find out whether the different tool-specific WEUD M2 models are instances of our general M2 model, we use the MOFLON visual tool and the description logic mapping. First, we store the proposed model in the reference TBox and then we use the same description logic to describe the particular WEUD model to be checked. The result of the inference is an ABox establishing the necessary transformation rules to convert, whenever possible, the specific model into the reference model. The MOFLON tool visually illustrates equivalences and describes whether or not the particular model is an instance of the reference model and any identified discrepancies. Fig. 11 illustrates (using the Fujaba plugin rather than the more recent Eclipse, which is not altogether compatible with the RacerPro TBox) what checks were run to examine whether the Yahoo! Pipes and Dapper M2 component model, for example, is an instance of the proposed general WEUD model. The inputs for this MOFLON-based checking process are two models codified in UML using the eMOFLON tool. The process outputs a Boolean deciding whether or not one model is an instance or subsumption of the other. If it is, eMOFLON also plots a graph of the transformations needed to assure the subsumption.

As illustrated in Fig. 11, the process is completed successfully, stating that the Yahoo! Pipes and Dapper model is a particular instance of the proposed WEUD model and specifying a set of constraints:

- "Mash-up" is composed of only one "Workspace" in Yahoo!, so there is no possibility of the user creating more than one visually independent interrelated workspace.
- "Operation" is limited in Yahoo! to list and functional operations, and basic arithmetic operations are not permitted.
- "Wrapped Service" is restricted to "Invoke Feed" in Yahoo!, so that it is only possible to fetch RSS sources, and SOAP- or REST-based web services cannot be invoked in this environment.

The output constraints are also useful indicators of the shortcomings of each analysed WEUD tool with respect to what characteristics RIAs should have.

After checking all analysed WEUD models (Yahoo!, Kapow Platform, JackBe, PopFly, Netvibes, AMICO, Marmite, etc.), we found that all the models of these tools are subsumed by the WEUD model proposed in this paper, but they all have constraints with respect to the model. These shortcomings imply that specific models do not include all the functionalities of our model (in fact, their qualitative study suggests that they do not have the target features that we propose in this paper). The proposed model has been designed based on the premises described as key points to be taken into account, subsumes all the analysed models and is potentially more comprehensive than they are.

All this formalization work is very relevant for two reasons: (a) it is necessary to cover the main models and tools in use today in

Fig. 12. WEUD model compliant EzWeb platform.

order to assure backward compatibility, standing on the shoulders of giants such as Google, Microsoft or Yahoo!, and (b) it appears to be feasible to create a unique homogeneous tool that could be fed by the components and models of all existing tools, exploiting their ecosystems of services and resources. Having completed these proofs, it remains to empirically check that a tool based on this model achieves better results than the other analysed WEUD tools.

6. Controlled experimental study of the proposed model using the EzWeb tool

The WEUD component model is an abstract element that cannot be statistically studied without a RIA development tool that implements this model in the real world (level M0). The only other way of validating the model would be to have a panel of experts in the WEUD field analyse its features. This would unquestionably result in a rather artificial qualitative validation removed from end-user concerns. Therefore, we studied the model as follows: we built an end-user tool called EzWeb as part of a NESSI strategic R&D project [20] targeting the construction of composite web applications. Fig. 12 shows a screenshot of this tool. This tool was designed and implemented from scratch to assure that the products built by using this environment conform to the described model [20]. We described two experiments to demonstrate that EzWeb was suitable for users without programming skills and helped them to build web solutions to a particular problem that they were set elsewhere [43,45,46]. In this study, we report an experiment addressing a broader set of problems of increasing complexity in order to address the goodness of the model. We also analysed all the applications built using EzWeb to check that these results really are an instantiation of the proposed model. Although it would be necessary to test a larger number of end-user applications built using EzWeb to fully validate the tool, we believe that this experimental study is sufficient for the purpose of this paper. EzWeb is described in greater length in [3,17,38].

The procedure was as follows. We recruited a sample that was large enough to characterize the target users of WEUD tools. Targets are non-programmers who are very well acquainted with the problem domain and have all the resources that they need to solve the problem except tools to automate the problem-solving process. The sample of 210 EUPs was divided into seven groups. Each group used one WEUD tool (Yahoo! Pipes and Dapper, Kapow Platform (OpenKapow version 9.3.0), PopFly, JackBe, AMICO, Marmite or EzWeb) to solve a set of seven problems of increasing complexity, which were carefully selected to assure that they could be solved by all the tools using the component catalogues made available during the experiment. The results were analysed to find out which component models each tool is able to create and confirm that the RIAs output by each tool conform to their component model described above. This meant that we were able to validate the models described for each tool (including EzWeb) throughout this paper. We also studied three factors: number of EUPs that manage to successfully develop a RIA based on the component model used by each tool under study, the development time taken using each tool under study and the number of errors in solutions built using each component model under study. This paper sets out to study these factors in response to the following research questions:

- RQ1: Does the EzWeb tool based on the proposed WEUD model enable many more EUPs to successfully develop RIAs than other WEUD tools?
- RQ2: Do the RIA designs built conform to the component model described for each tool throughout this paper and are they instances of the WEUD metamodel?

The total sample of 210 EUPs was selected so as not to be biased by age, gender, education or employment. The sample was chosen in a selection process, capturing interested users through training programs promoted from several NESSI project web portals. Users were captured and invited to two macro events organized by NESSI and by its associated 7th programme projects at Madrid and Brussels, respectively. Sixty people (divided into two groups of 30 users) attended the Madrid event and 150 (another five groups) attended the Brussels meeting. The sample was specially selected to ensure that, like the tool target population, none of the users had programming knowledge or were familiar with the analysed WEUD tools. Also users forming an unbiased sample were invited (see Table 3).

The size of each group is statistically representative, and normality tests can be run. The first major step of this study is to statistically check that there is no bias. After group formation, we

Table 3
Sample characterization.

Characterization	Sample (210)	Group characterization (7 30-member groups/)
Gender		
Male	112	16
Female	98	14
Age		
<20 years	35	5
20–34 years	49	7
35–49 years	49	7
50–64 years	42	6
>65 years	35	5
Educational attainment		
Secondary school	49	7
Vocational training	56	8
Bachelor's degree	49	7
Master's degree	56	8
Employment		
Student	70	10
Researcher	70	10
Employee	70	10
Experience and previous knowledge		
Mashup platforms	7	1
Web services (SOAP, ESB, BPEL, etc.)	0	0
HTML, CSS	7	1
Java, J2EE	0	0
JavaScript, AJAX	0	0
Php, ASP	0	0
OO programming	0	0
C, C++, C#	0	0
Scripting, Perl	0	0
Haskell, Prolog	0	0

ran an analysis of covariance (ANCOVA), using the group to which each end user was allocated as the study variable and the user characteristics as the explanatory variables in order to validate the division of users into different groups. This study builds a regression model to explain the study variable with respect to the other qualitative and quantitative variables. If the ANCOVA study were to return a well-fitted regression model, then the division would not be valid, as the groups would be biased with respect to the most influential explanatory characteristics in the fitted model.

Table 4shows the results of the ANCOVA study, which suggest that the model fit is extremely poor. This validates the selected sample and its distribution.

Looking at Table 4, we find that the coefficient of determination R^2 is very low (0.014). This suggests that there is a high percentage of variability in the modelled mean variable so that gender, age, educational attainment, employment and previous experience (the quantitative and qualitative variables for each individual) appear to explain only 1.4% of the division of users into the seven groups. The other values are due to other unknown variables. The R^2 and adjusted R^2 values suggest that the group to which each end user was allocated is largely (98.6%) independent of user characteristics. The model error values, MSE (mean squared error) and MAPE (mean absolute percentage error), are very high (well above the ideal value 0), again suggesting that the model does not precisely explain the behaviour of the variable under study in the sample. Additionally, DW (Durbin–Watson statistic) values are not close to 0. This implies that there is no autocorrelation among the qualitative variables. If there were, the study would not be valid. Finally, Cp (Mallows' Cp statistic) suggests that the model is able to exactly explain the group to which only one (see df value in the model) of the 210 individuals was allocated. We have conducted a Type I and Type III sum of squares analysis. Type I (sequential) analysis provides an incremental improvement in the sum of squared errors as each effect is added to the model, and Type III (orthogonal) analysis is able to reduce the sum of squared errors by adding the term after all other terms have been added to the model. Their combined use means that we do not have to be concerned about the order in which the factors were added to the regression model. Taken together, the model results validate the sample, indicating that there is no bias related to the qualitative and quantitative variables characterizing the users and their recruitment for the study. Looking at the Pr > F values of the ANCOVA model, we find that the characteristic that is most related to the allocation of a user to one group or another is education (the greatest Pr > F in the study, equal to 0.250). We examined user education and found no statistical evidence of a direct correlation between education and division into groups.

The validated sample was analysed as follows. In response to RQ1 and RQ2, all seven groups were asked to solve the same set of seven problems, each using one of the following

Table 4
ANCOVA statistical analysis of identified errors.

Goodness of fit statistics								
Observations	Sum of weights	*df*	R^2	Adjusted R^2	MSE	MAPE	DW	Cp
210	210	56	0.014	0.023	5.024	4.451	1.143	2
Source		df	Sum of squares		Mean squares	F		Pr > F
Analysis of variance								
Model		1	5.803		0.180	1.180		0.250
Error		163	9.082		0.150			
Corrected total		164	15.184					
Computed against model = Mean(Y)								
Type I sum of squares analysis:								
2.- Gender		1	0.035		0.040	0.212		0.115
3.- Age		1	0.151		0.143	0.422		0.034
4.1- Education		3	0.702		0.235	0.469		0.128
4.2-Employment		2	0.138		0.021	0.770		0.029
5.- Experience and previous knowledge		28	3.977		0.177	1.864		0.015
Type III sum of squares analysis:								
2.- Gender		1	0.034		0.041	0.212		0.110
3.- Age		1	0.154		0.143	0.422		0.034
4.1- Education		3	0.704		0.233	0.468		0.130
4.2-Employment		2	0.139		0.025	0.775		0.031
5.- Experience and previous knowledge		28	3.965		0.177	1.864		0.015

Table 5 Summary of experiment statistical results.

	Mean number of solutions for all 7 problems ($N = 30$)	Mean time taken	Std. Dev. time taken	Q1 time taken	Q3 time taken	Mean bugs	Std. Dev. bugs	Q1 bugs	Q3 bugs
Yahoo!Pipes and Dapper	12.00	21.50	2.01	18.65	24.13	6.33	0.12	5	7
Kapow platform	12.14	25.80	1.89	23.44	28.37	2.35	0.09	2	3
Popfly	8.57	31.42	1.55	29.42	33.01	5.69	0.21	5	6
JackBe	11.43	20.33	1.42	18.63	22.19	2.36	0.18	2	3
AMICO	11.43	19.60	0.98	17.98	21.08	9.68	0.97	9	10
Marmite	7.71	25.73	0.99	24.50	27.45	5.41	0.45	5	6
EzWeb	23.00	15.33	0.79	14.75	17.21	0.37	0.02	0	1

tools: Yahoo! Pipes and Dapper, Kapow Platform, PopFly, JackBe, AMICO, Marmite and EzWeb. The complexity of the problems increases in terms of the number of components to be used (including screens, screen flows, forms, connectors, operators, back-end services, etc.), and the dataflow connections among components. The first problem (problem 1) can be solved using a few components (about six elements), problem 2 using eight, 3 using ten, 4 using from 12 to 14 components (depending on how each individual user proceeds), 5 using from 15 to 17, 6 requires from 18 to 20 components, and it takes around 22 to 24 components, and about 20 data flows between components to solve the last problem 7. Thus problem complexity (in terms of the number and variety of components and interconnections) increases linearly. The full problem statements (together with a description of their complexity) are described at http://apolo.ls.fi.upm.es/eud/incremental_problems_description.pdf.

All the problems were carefully defined to assure that all seven tools under evaluation have all the components, composition and dataflow creation techniques necessary to solve each one. Before we conducted the study, we personally solved the problems using each tool to check that the task was feasible. Additionally, we also set up a catalogue of components, resources and operators for each tool before running the experiment. These catalogues included all the components and elements necessary to solve all the problems, as well as general-purpose components that were of no use for the problem at hand. All seven catalogues contained around 400 components of different levels of abstraction. Therefore, this is not a straightforward development. For a full and detailed description of these seven catalogues of components and connectors and the development processes enacted by the sample of users, see http://apolo.ls.fi.upm.es/eud/solution_development_process.pdf.

After the tools and equivalent component catalogues for each group had been set up, the EUPs received basic training via video tutorials on the tool that they were to use. Each group was separately given the same number of training sessions. The training sessions focused on explaining how to use each tool to solve like problems, explaining the components to be used provided by each tool. Accordingly, the design of the sessions was identical for all tools:

- Fundamentals session (4 h): Introduction to and contact with development technology and intercommunication for composite applications, widgets and mashups.
- Practical session (4 h): Development of solutions using existing WEUD tools: Yahoo! Pipes and Dapper, Kapow Platform, PopFly, JackBe, AMICO, Marmite and EzWeb platforms.

Then each group tackled the seven stated problems in separate sessions. Members of group 1 were asked to develop the RIA using Yahoo! Pipes and Dapper, group 2 members used Kapow Platform, group 3 members, PopFly, group 4 members, JackBe, group 5 members, AMICO, group 6 members, Marmite and, finally, group 7 members, Ezweb. We supervised each group and counted

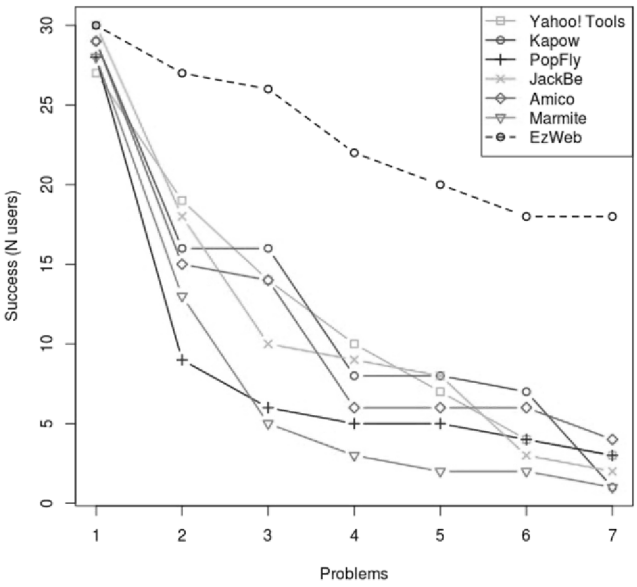

Fig. 13. WEUD success for each tool as problem complexity increases.

how many users were successful. We stored the resulting RIAs in order to later examine their component models (and validate whether they conform to the model reported for each tool in this paper), their qualities and functionalities. We also measured development time. Table 5 summarizes the aspects measured for each tool: mean number of users who managed to build a solution for the seven stated problems using the respective tool, mean time taken in minutes (provided a solution was built), standard deviation, first and third quartile of times taken, mean number of errors/bugs/inconsistencies detected in the RIAs, standard deviation of number of errors, first and third quartile of such errors. The illustrated data are mean data, calculated using the simple mean of the measurements taken for the seven assigned problems.

Looking at Table 5, we find that EzWeb has a much higher success rate after the experiments conducted on the seven assigned problems (a mean of 23 of the 30 users in the group managed to solve all of the problems). However, the results for the number of users that were successful at tackling each problem (of increasing complexity) using the different tools (see Fig. 13 and Table 6) are even more illustrative.

A total of 84 RIAs were generated using Yahoo! tools, 85 using Kapow Platform, 60 using PopFly, 80 using JackBe, 80 using Amico, 54 using Marmite and 161 RIAs using EzWeb for all seven stated problems, as shown in Table 6. Note that if the success rate for each problem had been 100%, each tool should have built 210 RIAs (30 for each problem).

Table 6
Summary of WEUD solutions reached in each group and problem.

	Yahoo! Pipes and Dapper	Kapow Platform	PopFly	JackBe	AMICO	Marmite	EzWeb
Problem 1	27	29	28	30	29	28	30
Problem 2	19	16	9	18	15	13	27
Problem 3	14	16	6	10	14	5	26
Problem 4	10	8	5	9	6	3	22
Problem 5	7	8	5	8	6	2	20
Problem 6	4	7	4	3	6	2	18
Problem 7	3	1	3	2	4	1	18
Total	84	85	60	80	80	54	161
Valid instances of respective WEUD tool model	100%	100%	100%	100%	100%	100%	100%
Instances subsumed by proposed WEUD metamodel	100%	100%	100%	100%	100%	100%	100%

Looking at Fig. 13 and Table 6, we find that almost all the tools achieve very positive results with simpler problems, but, as of problems whose solution requires eight to 10 components or more, there is a quadratic decrease in the number of successful users for all tools, except EzWeb, which has a consistently higher success rate. Understandably, the more complex the problem is, the fewer users solve it satisfactorily. However, the number of successful EzWeb users decreases linearly, and the minimum success rate never drops below 60% of the sample, unlike the number of successful users using the other tools, which plunges to approximately 14% of the sample at best for the most complex problem. Incidentally, this problem is a paradigmatic example of the most complex RIAs that this type of tools should help EUPs to develop.

6.1. Discussion of RQ1: does the EzWeb tool based on the proposed WEUD model enable many more EUPs to successfully develop RIAs than other WEUD tools?

The conducted study showed that all existing tools have shortcomings with respect to the support that they provide for EUPs. Of the 30 EUPs in each group, most managed to solve the simplest problem (27 Yahoo! tool users, 29 Kapow Platform users, 28 PopFly users, 30 JackBe users, 29 Amico users, 28 Marmite users and 30 EzWeb users). All the analysed tools achieve good results for this type of problems which can be solved using only five or six components, although it is true that the problems that EUPs have to deal with on a day-to-day basis are more complex than these. Looking at the last of the assigned problems (the most complex), only three Yahoo! Pipes and Dapper users managed to create a RIA that solved the problem, whereas 90% of the group was unable to solve the assigned problem using this tool. The results for the other tools are very similar, one Kapow Platform user, three PopFly users, two JackBe users, four Amico users and one Marmite user were successful. Kapow Platform and Marmite fared worst: users had to interconnect widgets using public APIs documented by these tools. The tool manufacturers' claim that these interfaces are easy to use for target EUPs, but the only users in the sample that were able to build an operational RIA using Kapow Platform or Marmite had some previous knowledge of web mashups. Moreover, users that successfully solved problem 7 using any of the tools had been working with their tool for several sessions and had dealt with six simpler problems beforehand. It is reasonable to assume then that the success rate would have been lower if the EUPs had to deal with this problem first (which is a possibility considering that end-user programming is opportunistic, and users want to solve problems without having to spend a lot of time learning the tool, techniques or components or heuristics required to do so).

EzWeb, however, enabled 18 users to successfully solve the two most complex problems. Besides, it is the only tool that did not suffer a sharp drop in the number of successful users as the complexity of the problem increased. This demonstrates that EzWeb successfully generates RIAs that conform to the proposed model, although the success of EzWeb cannot definitely be attributed to this fact. Worthy of note, however, is the fact that the proposed model is more complete than the models of the analysed existing tools (which it subsumes) and, although it is substantially more complex than the other component models, this did not, contrary to expectations, compromise the performance of the tool implementing the model in the experiments conducted with end users.

A statistical study of the subsamples that were successful using each tool individually did not return any significant data to suggest that this success was due to the descriptive variables of the sample, such as gender, age, education, background knowledge or profession. These findings reproduce the results of other statistical studies reported by Rode et al. [13], Ko et al. [1], and so on.

6.2. Discussion of RQ2: do the RIA designs built conform to the component model described for each tool throughout this paper and are they instances of our WEUD model?

We used UML to model the component diagram (models M1) of each of the generated RIAs (located at level M0). We then used the MOFLON automated reasoner to check that the M1 models of the RIAs created using each WEUD tool are valid instances of each M2 component model described in this paper as the general model implemented by each tool. We also checked that all the solutions are instances of the proposed M2 component model described for the WEUD tool. We used description logic to check that the 161 RIAs built using EzWeb conform to and are valid partial instances of the general component model proposed in this paper. As a result, we can state that the EzWeb tool is more efficient than the other analysed tools and implements the proposed general model. Furthermore, all the satisfactorily built RIAs (a total of 604) are valid instances of the general model proposed in this paper. To verify this, all 604 RIAs were input into a software modelling tool (the Altova UModel), which output their M1 models in UML format. These models were used to check whether they were instances of the proposed general model.

The component model implemented by EzWeb is the only one to instantiate the existing models for the WEUD field, because it has the right level of abstraction and a high number of component "types" for use. As we found by applying the MOFLON tool at the end of the experiment, the component models used by the other analysed WEUD tools are valid instances of the proposed metamodel. However, the 161 solutions built using EzWeb are not instances of all the component models implemented in the other WEUD tools, as these tools do not have enough available component types (at different levels of abstraction). As the component models are what distinguish the seven tools examined in the experiment (they all have the same compositional techniques, similar visual interfaces, equivalent visual languages, similar catalogues, similar visual aids, the same technology, services and resources), it is the component model implemented by EzWeb that is most likely to affect its success rate. Noteworthy

is the fact that the other WEUD tools have simpler component models with less diverse levels of abstraction that EUPs can use to solve their problems. Therefore, these tools work very well for simple problems, but the user success rate plummets as problems get more complicated. On the other hand, EzWeb implements a more comprehensive and varied component model, accounting for all the levels of abstraction stated in the proposed general model. This helps users to successfully tackle more complex problems, and the number of successful users does not fall off sharply.

In view of the widespread failure to develop complex RIAs that solved the more complex problems using different analysed tools, users were asked open-ended questions about the problems, obstacles and disadvantages that they had found during tool development (see the website [56]). The findings from the examination of the responses were:

- Regarding Yahoo! tools, 88% of the sample that used this tool stated that they had trouble interconnecting Yahoo! Dapper widgets with each other, whereas 80% missed the option for composing widgets based on finer-grained components. Also 82% percent of the sample highlighted that, apart from feeds and screen scraping-based information sources, Yahoo! Pipes failed to provide useful wrapped services for EUPs.
- Regarding Kapow Platform and Kapplets (an auxiliary tool supporting Kapow Platform), over 85% of the sample that used this tool found that Kapow component linking and tailoring mechanisms were not handy (required programming knowledge), whereas 78% found that the component search, location, parameterization and recommendation mechanisms were hard to use and understand.
- Regarding PopFly, over 90% of the sample that used this tool criticized the fact that they were unable to find the right elements in the catalogues and were not able to locate the right components for a particular problem.
- Regarding JackBe, 92% of the sample that used this tool stated that the composition visual interface should not be confined merely to linking visual elements, as it is not possible to parameterize or adapt the components to new situations or problems. Additionally, 75% of the sample found it impossible to establish correct data flows among the different components and widgets.
- Regarding Marmite, 85% of the sample that used this tool stated that the available components are too generic and their internal behaviour is not easily and visually modifiable and requires programming knowledge.
- Regarding Amico, 90% of the sample that used this tool was unable to use the adapters that enable specific SOAP service invocation from a spreadsheet. Note specifically that adapter use is not possible without knowledge of functional programming and data typing.

The users that failed to solve the problem with EzWeb all came up against the same obstacle: 95% of the users that failed to solve the problem did not realize that they had to compose components from finer-grained components located in the resources catalogue. The other 5% of users stated that the components that they required were not available and promptly gave up. What really happened was that they were unable to locate the components that were in fact there. This discovery is the inspiration for an important future line of research which is to provide a wizard to translate specific requirements to a list of problem-solving components and help users to search large catalogues for these components.

We then conducted a survey with unstructured and open-ended questions about the tool used by each particular user. According to this survey, 83% of the sample that used EzWeb stated that this tool was very well suited to problem solving by means of compositional development. Because it was designed to create increasingly specific components, it achieved much better results than any other tool (the success rate for the other tools was at most 14% for the more complex problems). Proof of this is that using EzWeb, EUPs were able to build bigger components based on more specific components and create data flows among such components. The survey, its results and a statistical study are published in [56]. These key aspects, which were presented in Section 2, are the groundwork of the proposed WEUD model that yielded very good results in the experiment.

With respect to the errors detected in the RIAs, all the tools generated consistent results with some integration errors but no component errors. Statistical studies of covariance did not show any signs of the applications built by any of the tools being more or less error prone. These studies were reported in [56].

7. The domain specificity dilemma

Note that there is a major multifaceted dilemma that only the development of and empirical results regarding WEUD will be able to solve: will a multi-purpose and multi-domain web end-user development environment build better software solutions than specialized domain tools? Two lines of thought have attracted similar interest, with research and development papers advocating different courses of action. As some generic EUD applications, like spreadsheets, have been very successful, many researchers advocate setting up a web portal that is general enough to be used to develop solutions for more than one problem domain [1,29,30]. EzWeb, and the proposed model falls into this category. However, other researchers, like [27], [28], etc., take the opposite tack, claiming that, as EUPs are indisputable experts in their domain, it is best practice to design domain-specific EUD tools that will improve the efficiency and ease with which EUPs can build ad hoc solutions to their problems. According to the results of our research, the ideal thing would be a trade-off between the two lines of thought: domain-specific EUD tools can be used to successfully solve simple problems involving components and elements specific to only one or a very well-defined problem domain, whereas a component model like we propose, which is generic and powerful enough to model components for different problem domains (for example, personal organization elements such as email and calendars, heterogeneous visual web resources, RSS data feeds, remote web service invocations, etc.), is useful for solving more complex and multidisciplinary problems.

In any case, unless EUD addresses general development methods, as well as languages and approaches that are not ad hoc for a particular domain, it will conceivably end up facing a crisis like the one that rocked the software industry in the late 1960s. It makes sense that EUD should empower users to build increasingly more general-purpose and complex software following heuristics and methodologies driven by a wizard or similar to enact efficient and valid life cycles to develop solutions to their problems.

8. Conclusions

Currently, web software development tools by EUPs work well for small problems, but do not produce valid solutions for more complex problems. The study that we report shows that the EzWeb tool, based on the proposed general component model, yields better results than other tools for problems including more building blocks and data flows between components. The biggest shortcoming of the existing tools that we studied is not the visual languages or actual development techniques, but the component models that EUPs use to build their RIAs. EUPs, who generally have no notions of software development, cannot undertake the entire development process as they are unable to find software abstractions tailored to the problem to be solved

and to use complex techniques that are designed for programmers. Additionally, each tool has its own particularities. This prevents a component or set of components built for one tool from being exploited in another tool, thereby limiting the number of available resources and components. This problem is what has been repeatedly reported in the WEUD environment as an open issue [17].

In any case, this paper addresses the concern that the type of components that current WEUD tools use and the proposed interactions between component domains tend to restrict their problem-solving scope, as does the fact that their components are usually designed for specific application domains. Therefore, the general user-centred component model exploits the factors that have led current WEUD tools and other EUD approaches, such as spreadsheets, to triumph, offering a more comprehensive and powerful component model than other tools that should enable EUPs to address more complex problems. The general web component model subsumes the models of the analysed WEUD tools, taking on board their advantages. It is also able to overcome the barriers that the other tools face regarding available components, user interface, component repositories, etc. We have formalized this model in UML2 and in description logic in order to verify that the proposed general model is valid, that it subsumes the other existing models that we studied, and that these models are valid instances of the proposed model. Based on this formalization, semi-automatic mechanisms could be designed to adapt the components of any tool to the proposed general-purpose model. Additionally, we have built a tool, called EzWeb, which produces RIAs instantiating the proposed formalized model. We prove statistically that EUPs using this tool to solve rather complex real use cases achieve better results than the users of the other analysed tools. We cannot conclusively confirm that these sound results are a direct consequence of the proposed model (the model would have to be built into different EU tools to build more applications to check out this point), but the proposed component model is found to address more general and complex EUP problems than the component models of the existing tools analysed in this study.

The component model implemented by this tool has been used successfully in several 7th framework research projects, and EzWeb has been used to develop RIAs targeting citizens of several Spanish public administration Web 2.0 portals [57].

An important future research line in this topic is to use the proposed reference model as a starting point for standardizing existing web components for adaptation to end user-centred development environments so that all the mashup tools and platforms can exploit, create and parameterize components from different sources irrespective of the target tool for which the component was built. This would provide a component model with standardized components which would improve the development of WEU software and further increase the number of available components for end-user development.

Moreover, WEUD is only part of all the support that EUPs should receive from when they are assigned the problem until they manage to use a software solution to the problem. We agree with other authors [1,58] that it is necessary to provide a special-purpose life cycle and life-cycle support tools to shepherd EUPs through the stages of requirements specification, analysis, design and implementation, testing and use. Indeed, we set out elsewhere a web-based approach to end-user software engineering (EUSE) that provides such support [45].

Acknowledgements

The authors would like to thank the 240 EUPs of the experiment for their interest, participation and time. This work has been partially supported by the EU co-funded IST projects FAST: Fast and Advanced Storyboard Tools (GA FP7-216048), FI-WARE: Future Internet Core Platform (GA FP7-285248), and 4CaaSt: Building the Platform as a Service of the Future (GA FP7-258862).

Appendix A. Proposed EUD model formalized in $ALC\epsilon NOQ_{HR}^{-}{}^{+o}$ description logic

See Table A.1.

Appendix B. Description of the seven problems used during the experiment

Problem 1: Term translation and definition

Number of components and elements: 6

A user wants to build an application that she can use to check the translation and definition of a written word in a search dialogue. The application will therefore have a space for the user to enter the word. The application will look up the word in:

1. Wikipedia
2. Collins dictionary
3. The user will be given the choice of German, French or Spanish translations.

The application interface should contain three separate areas to display the search results.

Problem 2: Open Office document broadcasting

Number of components and elements: 8

The aim is to build a simple application to open and visualize any Open Office document from a simple and straightforward interface. Additionally, the user should be able to send the above document to the printer according to the standard network printer configuration at the click of a button. Additionally, the document will be sent to a set of email recipients listed on a pull-down list.

Both the configuration and the emailing options should be resettable every time the application is run.

Problem 3: Photo Viewer

Number of components and elements: 10

A user wants to be able to view photographs of a particular tourist destination. She wants to build an application that takes the name of the city or place and:

1. Searches images of the city in the network resources to which she has access from where she is running the application.
2. Search photographs of the place in Flickr and Yahoo! Image Search.
3. The photo will be magnified to full screen size in a new workspace, when the user clicks on a photo.

The search results (points 1 and 2) will be displayed in a main workspace, specifying the source of the photograph. On the other hand, a new "preview screen" workspace should open when the user clicks on any visual resource.

Problem 4: Alerting

Number of components and elements: 12–14

Part of a user's routine work is to supervise changes and notifications published by two of his country's public administration webpages. This user wants to build an application that is capable of automatically reviewing these two webpages and emailing and SMS messaging these specific changes to him. Therefore he is looking for an application that:

1. Stores a baseline containing the original status of the two webpages.

Table A.1

DesignElement ⊑ Class ⊓∃ Any.Description ⊓⟨≤ 1Description⟩⊓
∀ RoleComposition.DesignElement ⊓
∀ group_Composition-.DesignElement ⊓
∀ part_Composition-.DesignElement ⊓
∀ publishedin_Part-.publishedin ⊓
∀ publishedinRole-.Catalogue ⊓
∀ composedof_Group-.composedof ⊓
∃ composedofRole.AbstractGUIDE ⊓
⟨≤ 1composedofRole.AbstractGUIDE⟩⊓
∃ composedofRole.ResourceWrapper ⊓
⟨≤ 1composedofRole.ResourceWrapper⟩⊓
∃ composedofRole.Precondition ⊓⟨= 1composedofRole.Precondition⟩⊓
∃ composedofRole.Postcondition ⊓⟨= 1composedofRole.Postcondition⟩

composition ⊑ Aggregation ⊓∃ group_Composition.DesignElement ⊓
∃ part_Composition.DesignElement ⊓
⟨≤ 1group_Composition⟩ ⊓ ⟨≤ 1part_Composition⟩

roleComposition ≡ group_Composition-∘ part_Composition

publishedin ⊑ Aggregation ⊓∃ publishedin_Group.Catalogue ⊓
⟨≤ 1publishedin_Group⟩⊓
⟨≤ 1publishedin_Part⟩⊓
∃ publishedin_Part.DesignElement

publishedinRole ≡ publishedin_Group-∘ publishedin_Part

composedof ⊑ Aggregation ⊓∃ composedof_Group.DesignElement ⊓
⟨≤ 1composedof_Group⟩ ⊓ ⟨≤ 1composedof_Part⟩⊓
∃ composedof_Part.Precondition ⊓⟨= 1composedof_Part.Precondition⟩⊓
∃ composedof_Part.Postcondition ⊓⟨= 1composedof_Part.Postcondition⟩⊓
∃ composedof_Part.AbstractGUIDE ⊓⟨≤ 1composedof_Part.AbstractGUIDE⟩⊓
∃ composedof_Part.ResourceWrapper ⊓
⟨≤ 1composedof_Part.ResourceWrapper⟩

composedofRole ≡ composedof_Group-∘ composedof_Part

Catalogue ⊑ Class ⊓∃⟨DesignElement1,DesignElementn⟩⊓
∀ publishedinRole.DesignElement ⊓
∀ publishedin_Group-.publishedin

AbstractGUIDE ⊑ Class ⊓∃ Parameters.any ⊓⟨≥ 0Parameters⟩⊓
∀ composedof_Part-.composedof ⊓
∀ composedofRole-.DesignElement

Image ⊑ Class ⊓ AbstractGUIDE

Text ⊑ Class ⊓ AbstractGUIDE

Hypertext ⊑ Class ⊓ AbstractGUIDE

Precondition ⊑ Class ⊓
∀ composedof_Part-.composedof ⊓
∀ composedofRole-.DesignElement ⊓
∀pre-memberRole.Fact ⊓
∀pre-member_Group-.pre-member

pre-member ⊑ Aggregation ⊓∃ pre-member_Group.Precondition ⊓
⟨≤ 1pre-member_Group⟩⊓
⟨≤ 1pre-member_Part⟩⊓
∃pre-member_Part.Fact

pre-memberRole ≡ pre-member_Group-∘ pre-member_Part

Postcondition ⊑ Class ⊓
∀ composedof_Part-.composedof ⊓
∀ composedofRole-.DesignElement ⊓
∀ post-memberRole.Fact ⊓
∀ post-member_Group-.post-member

post-member ⊑ Aggregation ⊓∃ post-member_Group.Postcondition ⊓
⟨≤ 1post-member_Group⟩⊓
⟨≤ 1post-member_Part⟩⊓
∃ post-member_Part.Fact

post-memberRole ≡ post-member_Group-∘ post-member_Part

Fact ⊑ Class ⊓∀ pre-member_Part-.pre-member ⊓
∀pre-memberRole-.Precondition ⊓
∀ post-member_Part-.post-member ⊓

(continued on next page)

Table A.1 (*continued*)

$\forall$ post-memberRole-.Postcondition $\sqcap$
$\forall$ ccomposesfact_Group-.composefact $\sqcap$
$\exists$ composesfactRole.Semantics $\sqcap$
$\langle \leq$ 1composesfactRole.Semantics$\rangle \sqcap$
$\exists$ composesfactRole.Data $\sqcap \langle =$ 1composesfactRole.Data$\rangle$

composesfact $\sqsubseteq$ Aggregation $\sqcap \exists$ composesfact_Group.Fact $\sqcap$
$\langle \leq$ 1composesfact_Group$\rangle \sqcap$
$\langle \leq$ 1composesfact_Part$\rangle \sqcap$
$\exists$ composesfact_Part.Data $\sqcap \langle =$ 1composesfact_Part.Data$\rangle \sqcap$
$\exists$ composesfact_Part.Semantics $\sqcap \langle \leq$ 1composesfact_Part.Semantics$\rangle$

composesfactRole $\equiv$ composesfact_Group-$\circ$ composesfact_Part

Semantics $\sqsubseteq$ Class $\sqcap$
$\forall$ composesfact_Part-.composesfact $\sqcap$
$\forall$ composesfactRole-.Fact

Data $\sqsubseteq$ Class $\sqcap \exists$ Value.any$\sqcap \langle =$ 1Value$\rangle \sqcap$
$\forall$ composesfact_Part-.composesfact $\sqcap$
$\forall$ composesfactRole-.Fact $\sqcap$

ResourceWrapper $\sqsubseteq$ Class $\sqcap$
$\forall$ composedof_Part-.composedof $\sqcap$
$\forall$ composedofRole-.DesignElement

InvocationofService $\sqsubseteq$ Class $\sqcap$ ResourceWrapper

DataPreparation $\sqsubseteq$ Class $\sqcap$ ResourceWrapper

Solution $\sqsubseteq$ Class $\sqcap$ DesignElement $\sqcap \forall$ composessolution_Group-.composessolution $\sqcap$
$\exists$ composessolutionRole.Mash-up $\sqcap \langle \geq$ 1composessolutionRole.Mash-up$\rangle$

Mash-up $\sqsubseteq$ Class $\sqcap$ DesignElement $\sqcap \forall$ composessolution_Part-.composessolution $\sqcap$
$\forall$ composessolutionRole-.Solution $\sqcap$
$\forall$ composesmash-up_Group-.composesmash-up$\sqcap$
$\exists$ composesmash-upRole.WorkSpace $\sqcap$
$\langle \geq$ 1composesmash-upRole. WorkSpace$\rangle$

composessolution $\sqsubseteq$ Aggregation $\sqcap \exists$ composessolution_Group.Solution $\sqcap$
$\langle \leq$ 1composessolution_Group$\rangle \sqcap$
$\langle \leq$ 1composessolution_Part$\rangle \sqcap$
$\exists$ composessolution_Part.Mash-up $\sqcap \langle \geq$ 1composessolution_Part.Mash-up$\rangle$

composessolutionRole $\equiv$ composessolution_Group-$\circ$ composessolution_Part

WorkSpace $\sqsubseteq$ Class $\sqcap$ DesignElement $\sqcap \forall$ composesmashup_Part-.composesmashup $\sqcap$
$\forall$ composesmashupRole-.Mash-up $\sqcap$
$\forall$ composesspace_Group-.composesspace $\sqcap$
$\exists$ composesspaceRole.Widget $\sqcap$
$\langle \geq$ 1composesspaceRole.Widget$\rangle$

WorkSpace $\sqsubseteq$ Aggregation $\sqcap \exists$ composesmashup_Group.Mash-up $\sqcap$
$\langle \leq$ 1composesmashup_Group$\rangle \sqcap$
$\langle \leq$ 1composesmashup_Part$\rangle \sqcap$
$\exists$ composesmashup_Part.WorkSpace $\sqcap \langle \geq$ 1composesmashup_Part.WorkSpace$\rangle$

composesmashupRole $\equiv$ composesmashup_Group-$\circ$ composesmashup_Part

Widget $\sqsubseteq$ Class $\sqcap$ DesignElement $\sqcap \forall$ composesspace_Part-.composesspace $\sqcap$
$\forall$ composesspaceRole-.WorkSpace $\sqcap$
$\forall$ composeswidget_Group-.composeswidget $\sqcap$
$\exists$ composeswidgetRole.ResourceRepresentation $\sqcap$
$\langle \geq$ 1composeswidgetRole.ResourceRepresentation$\rangle$

composesspace $\sqsubseteq$ Aggregation $\sqcap \exists$ composesspace_Group.WorkSpace $\sqcap$
$\langle \leq$ 1composesspace_Group$\rangle \sqcap$
$\langle \leq$ 1composesspace_Part$\rangle \sqcap$
$\exists$ composesspace_Part.Widget $\sqcap \langle \geq$ 1composesspace_Part.Widget$\rangle$

composesspaceRole $\equiv$ composesspace_Group-$\circ$ composesspace_Part

ResourceRepresentation $\sqsubseteq$ Class $\sqcap$ DesignElement $\sqcap \forall$ composeswidget_Part-.composeswidget $\sqcap$
$\forall$ composeswidgetRole-.Widget $\sqcap$
$\forall$ composesrepresentation_Group-.composesrepresentation $\sqcap$
$\forall$ composesrepresentationRole.BackendResource $\sqcap$
$\exists$ composesrepresentationRole.View $\sqcap$
$\langle \leq$ 1composesrepresentationRole.View$\rangle$

(continued on next page)

Table A.1 (*continued*)

composeswidget ⊑ Aggregation ⊓∃ composeswidget_Group.Widget ⊓
⟨≤ 1composeswidget_Group⟩⊓
⟨≤ 1composeswidget_Part⟩⊓
∃ composeswidget_Part.ResourceRepresentation ⊓⟨≥ 1composeswidget_Part.ResourceRepresentation⟩

composesspaceRole ≡ composesspace_Group-∘ composesspace_Part

composesrepresentation ⊑ Aggregation ⊓∃ composesrepresentation_Group.ResourceRepresentation ⊓
⟨≤ 1composesrepresentation_Group⟩⊓
⟨≤ 1composesrepresentation_Part⟩⊓
∃ composesrepresentation_Part.BackendResource ⊓
∃ composesrepresentation_Part.View ⊓⟨≤ 1composesrepresentation_Part.View⟩

composesrepresentationRole ≡ composesrepresentation_Group-∘ composesrepresentation_Part

View ⊑ Class ⊓ DesignElement ⊓
∀ composesrepresentation_Part-.composesrepresentation ⊓
∀ composesrepresentationRole-.ResourceRepresentation

BackendResource ⊑ Class ⊓ DesignElement ⊓
∀ composesrepresentation_Part-.composesrepresentation ⊓
∀ composesrepresentationRole-.ResourceRepresentation ⊓

Operator ⊑ Class ⊓ BackendResource

Arithmetic ⊑ Class ⊓ Operator

List ⊑ Class ⊓ Operator

Functional ⊑ Class ⊓ Operator

WrappedService ⊑ Class ⊓ BackendResource ⊓
∀ composesservice_Group-.composesservice ⊓
∀ composesserviceRole.API⊓
∀ composesserviceRole.ServiceData

composesservice ⊑ Aggregation ⊓∃ composesservice_Group.WrappedService ⊓
⟨≤ 1composesservice_Group⟩⊓
⟨≤ 1composesservice_Part⟩⊓
∃ composesservice_Part.API⊓
∃ composesservice_Part.ServiceData

composesserviceRole ≡ composesservice_Group-∘ composesservice_Part

API ⊑ Class ⊓∀ composesservice_Part-.composesservice ⊓
∀ composesserviceRole-.WrappedService

ServiceData ⊑ Class ⊓∀ composesservice_Part-.composesservice ⊓
∀ composesserviceRole-.WrappedService

2. Examines these web pages at a user-configurable time interval and checks whether any changes have been made compared against the baseline. The webpages do not publish RSS or notify content changes in any other way.
3. Is able to SMS message or email an outline of any changes made to the user's mobile phone and email address. Both data should obviously be configurable at runtime.

Problem 5: Planning a route between two points

Number of components and elements: 15–17

An application is to be implemented that plans the best route between two points taking into account images captured by cameras monitored by the administration that manages the traffic in the user's city. In Spain, this traffic information is managed by the Directorate General of Traffic. The application should:

1. Use Google Maps to search the optimal route between two points or locations,
2. Access the system clock to establish the exact execution time.
3. Query the traffic cameras of the main highways in the route.
4. Process each image using some graphical recognition mechanisms, such as the neural networks service. If there are many processing reference points, the image will be assumed to contain a lot of vehicles, and Google will be told to try to avoid that route.
5. Display the final route to the user, together with the images of the roads associated with the route. The other processed routes do not have to be displayed.

Problem 6: Customer service application management

Number of components and elements: 18–20

A domain expert routinely performs customer service tasks. Her job is to receive telephone calls and optimally process the reported fault or problem. The application to be implemented should:

1. Provide the user with a means to enter an address in the system (location of the fault) and a description.
2. Display the location on a map (Google Maps, Bing Maps, etc.).
3. Email the fault description to an operator who should travel to the fault location.
4. Display the position of the company's mobile operators on a map and recommend the nearest operator to be sent to repair the fault. The operator positions will be processed through a GPS built into their mobile phones and will be stored in a database.
5. If the fault is urgent, offer the user the option of calling the nearest operator (for example, using Skype). If the operator in question has a videoconferencing option, a visual connection will be established via the user's webcam.
6. When an operator has been allocated, save all the fault information in a database and print as a customer service

invoice. The database will be very basic (Access, MySQL, etc.). The virtual invoice will be printed out in PDF format.

Problem 7: Business trip booking and personal agenda management

Number of components and elements: 22–24

As part of a R&D project in which he is participating, a higher education worker has to make numerous national and international trips. The project has several partners of different types and origins.

The R&D project has a Web-based general agenda shared by all the project partners. All face-to-face meetings are posted in this agenda, specifying the meeting date and time, venue and agenda. The higher education institution employing the user actively cooperates with two travel agencies, one specialized in high-speed trains and the other in long-distance flights, and both manage all the travel and accommodation options at the full range of hotels.

1. The user consults the shared R&D project agenda every day to check whether there is a new meeting that he should attend.
2. If there is to be meeting, he has to check his personal agenda to find out whether he can attend the meeting and fill in the details of the new meeting, the meeting agenda, etc.
3. The user looks up the meeting venue, and searches for it on a map. Then, he accesses the travel agency services and checks what travel options they offer, as well as price. Normally he compares the two options and chooses one agency or the other depending on the travel options, length of stay and price.
4. If the trip is to last longer than a day, the user searches hotels near to the meeting venue and checks the prices per room and night offered by the travel agencies.
5. The department employing the user has a spreadsheet-based software program that manages the department-run R&D project budget. It contains spreadsheets that can be used to check the travel budget currently available for each project and manage new expenses. It is the user's job to calculate how much the travel and chosen accommodation will cost, add this up and check that there is enough money available for the trip and deduct it from the project budget.
6. Then the user makes the bookings one by one.
7. Finally, the user checks the Internet information about his destination, demographic characteristics, weather prediction, etc.

The user has many software solutions to tackle this repetitive task (project agenda, personal agenda, travel agency services, department cash flow program, etc.) but has to access distributed information, heterogeneous services, etc., separately. The user is a non-programmer, meaning that he has never thought of the possibility of building a solution that meets his needs and improves task performance.

References

[1] A.J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M.B. Rosson, G. Rothermel, M. Shaw, S. Wiedenbeck, The state of the art in end-user software engineering, J. ACM Comput. Surv. 43 (3) (2011) Article 21.

[2] C. Scaffidi, M. Shaw, B. Myers, The "55M End User Programmers" Estimate Revisited. Technical Report CMU-ISRI-05-100, Carnegie Mellon University, 2005.

[3] D. Lizcano, J. Soriano, M. Reyes, J.J. Hierro, A user-centric approach for developing and deploying service front-ends in the future Internet of services, Int. J. Web Grid Serv. 5 (2) (2009) 155–191.

[4] Google, Chrome Web Store for Chrome platform, 2014. https://chrome.google.com/webstore.

[5] Yahoo! Yahoo! Dapper web platform, 2012. http://open.dapper.net.

[6] Yahoo! Yahoo! Pipes web platform, 2012. http://pipes.yahoo.com/.

[7] Microsoft. Microsoft Popfly web platform, 2012. http://www.popfly.com.

[8] Kapow Software Kapow Platform, Kapplets and Kappzone, 2015. http://kapowsoftware.com/products/kapow-katalyst/index.php.

[9] JackBe, JackBe Presto Cloud web platform, 2012. http://prestocloud.jackbe.com/.

[10] AMICO, AMICO web platform, 2012. http://amico.sourceforge.net.

[11] Marmite. Marmite web platform, 2012. http://www.cs.cmu.edu/~jasonh/projects/marmite/.

[12] EzWeb, EzWeb web platform, 2012. http://ezweb.morfeo-project.org/.

[13] J. Rode, Y. Bhardwaj, M.A. Perez-Quinones, M.B. Rosson, J. Howarth, As easy as click: End-user web engineering, in: Proceedings of the 2005 International Conference on Web Engineering, 2005, pp. 478–488.

[14] M.B. Rosson, J. Ballin, J. Rode, Who, what, and how: A survey of informal and professional web developers. in: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, Washington DC, 2005, pp. 199–206.

[15] H. Lieberman, F. Paterno, V. Wulf, End-User Development, Kluwer/Springer Academic Publishers, Dordrecht, The Netherlands, 2006.

[16] Z. Obrenovic, D. Gasevic, Mashing up oil and water: Combining heterogeneous services for diverse users, IEEE Internet Comput. 13 (6) (2009) 56–64.

[17] D. Lizcano, F. Alonso, J. Soriano, G. Lopez, A new end-user composition model to empower knowledge workers to develop rich Internet applications, J. Web Eng. 3 (10) (2011) 197–233.

[18] A.P. McAfee, Enterprise 2.0: The dawn of emergent collaboration, MIT Sloan Manag. Rev. 47 (3) (2006) 21–28.

[19] J.A. Macías, F. Paternò, Customization of web applications through an intelligent environment exploiting logical interface descriptions, J. Interact. Comput. 20 (1) (2008) 29–47.

[20] EzWeb, Project, Official demo web site, 2011. http://demo.ezweb.morfeo-project.org.

[21] 4CaaSt Project, Official web site, 2012. http://4caast.morfeo-project.org.

[22] FI-WARE Project, Official web site, 2012. http://www.fi-ware.eu.

[23] L.M. Clark, R. Desisto, J. Holincheck, A. White, A. Kyte, A. Sarner, Hype cycle for software as a service. Gartner Research, August, Gartner Inc., USA, 2006.

[24] C. Schroth, O. Christ, Brave new web: Emerging design principles and technologies as enablers of a global SOA, in: Proceedings of the IEEE International Conference on Services Computing, IEEE Computer Society Press, Los Alamitos, CA, USA, 2007, pp. 597–604.

[25] Programmable Web. Programmable Web social portal, 2012. http://www.programmableweb.com.

[26] G. Alonso, F. Casati, H. Cuno, V. Machiraju, Web Services Concepts, Architectures and Applications, Springer, Germany, 2004.

[27] S. Aghaee, C. Pautasso, EnglishMash: usability design for a natural mashup composition environment, in: Proc. of Composable Web 2012, 2012.

[28] I. Muhammad, D. Florian, C. Fabio, M. Maurizio, ResEval Mash: a mashup tool that speaks the language of the user, in: CHI'12 NY, USA, 2012, ACM, New York, 2012, pp. 1949–1954.

[29] N. Mehandjiev, F. Lecue, U. Wajid, A. Namoun, Assisted service composition for EUPs, in: Proc. of ECOWS 2010, 2010.

[30] N. Mehandjiev, A. Namoune, U. Wajid, L. Macaulay, A. Sutcliffe, End user service composition: Perceptions and requirements, in: Paper presented at the Proceedings of the 2010 Eighth IEEE European Conference on Web Services, 2010.

[31] G. Fischer, A.C. Lemke, Construction kits and design environments: Steps toward human problem-domain communication, Hum.–Comput. Interact. 3 (3) (1988) 179–222.

[32] J. Grudin, The case against user interface consistency, Commun. ACM 32 (10) (1989) 1164–1173.

[33] D. Garlan, R. Allen, J. Ockerbloom, Architectural mismatch or why it's hard to build systems out of existing parts, in: Proceedings ICSE 1991, IEEE Computer Society Press, 1995, pp. 179–185.

[34] O. Stiemerling, Component-based tailorability (Ph.D.), University of Bonn, Bonn, 2000.

[35] V. Wulf, V. Pipek, M. Won, Component-based tailorability: Enabling highly flexible software applications, Int. J. Hum.-Comput. Stud. 66 (1) (2008) 1–22. http://dx.doi.org/10.1016/j.ijhcs.2007.08.007.

[36] A. Mørch, Application units: Basic building blocks of tailorable applications, in: Proceedings of the 5th Int'l East-West Conf. on HCI, Moscow, 1995, pp. 68–87.

[37] A. Mørch, G. Stevens, M. Won, M. Klann, Y. Dittrich, V. Wulf, Component-based technologies for end-user development, Commun. ACM 47 (2004) 59–62.

[38] D. Lizcano, F. Alonso, J. Soriano, G. Lopez, End-user development success factors and their application to composite web development environments, in: Proceedings of the Sixth International Conference on Systems, ICONS 11, 2011, pp. 99–108.

[39] D. Benslimane, S. Dustdar, A. Sheth, Services mashups: The new generation of web applications, IEEE Internet Comput. 12 (2008) 13–15.

[40] A. Maclean, K. Carter, L. Lövstrand, T. Moran, User-tailorable systems: pressing the issues with buttons, in: CHI90 Proceedings, 1990, pp. 175–182. ISBN: 0-201-50932-6.

[41] J.H. Wu, Y.C. Chen, L.M. Lin, Empirical evaluation of the revised end user computing acceptance model, Comput. Hum. Behav. 23 (1) (2007) 162–174.

[42] F.D. Davis, R.P. Bagozzi, P.R. Warshaw, Extrinsic and intrinsic motivation to use computers in the workplaces, J. Appl. Soc. Psychol. 22 (14) (1992) 1111–1132.

[43] D. Lizcano, F. Alonso, J. Soriano, G. López, Supporting end-user development through a new composition model: An empirical study, J. UCS 18 (2) (2012) 143–176.

[44] H. Kahler, More than words—collaborative tailoring of a word processor, J. UCS 7 (9) (2001) 826–847.

[45] D. Lizcano, F. Alonso, J. Soriano, G. López, A web-centred approach to end-user software engineering, ACM Trans. Softw. Eng. Methodol. 22 (4) (2013) 36.

[46] D. Lizcano, F. Alonso, J. Soriano, G. López, A component- and connector-based approach for end-user composite web applications development, J. Syst. Softw. 94 (2014) 108–128.

[47] OMG. Meta-Object Facility (MOF) core specification, Version 2.4.1. Object Management Group, 2006. http://www.omg.org/spec/MOF/2.4.1/PDF/.

[48] R. Sobek, Official MOF Specification from OMG. Technical Report, Object Management Group, Inc., USA, 2005.

[49] Racer Project. Racer Pro 2.0 Web Portal, 2013. http://racer-systems.com/.
[50] MOFLON. A standard-compliant metamodeling framework with graph transformations, in: A. Rensink, J. Warmer (Eds.), Model Driven Architecture—Foundations and Applications: Second European Conference, Vol. 4066, 2006, pp. 361–375 by C. Amelunxen, A. Königs, T. Rötschke, A. Schürr.
[51] V. Haarslev, R. Möller, Racer System Description, Springer-Verlag, Germany, 2001, pp. 701–705.
[52] F. Baader, D. Calvanese, D. Mcguinness, D. Nardi, P. Patel-Schneider, The Description Logic Handbook: Theory, Implementation and Applications, University Press, Cambridge, UK, 2003.
[53] S. Wang, L.J.C. Jin, Represent software process engineering metamodel in description logic, in: Proceedings of World Academy of Science, Engineering and Technology, 2006.
[54] K. Schild, A correspondence theory for terminological logics: preliminary report, in: John Mylopoulos, Ray Reiter (Eds.), Proceedings of the 12th International Joint Conference on Artificial Intelligence—Volume 1, (IJCAI'91), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991, pp. 466–471.
[55] H.J. Levesque, R.J. Brachman, Expressiveness and tractability in knowledge representation and reasoning, Comput. Intelligence 3 (1987) 78–93. http://dx.doi.org/10.1111/j.1467-8640.1987.tb00176.x.
[56] D. Lizcano, Statistical survey of the end-user development paradigm, 2013. Available at: http://apolo.ls.fi.upm.es/eud/.
[57] C. Tejo-Alonso, S. Fernández, D. Berrueta, L. Polo, M.J. Fernández, V. Morlán, eZaragoza, a tourist promotional mashup, 2011. Available at: http://idi.fundacionctic.org/eZaragoza/ezaragoza.pdf (Last access: 7.06.12).
[58] B.A. Myers, M. Burnett, M.B. Rosson, A.J. Ko, A. Blackwell, End user software engineering, in: Proceedings of chi'2008 Special Interest Group Meeting and in Extended Abstracts on Human Factors in Computing Systems, ACM, New York, NY, USA, 2008, pp. 2371–2374.