Supporting End-User Development through a New Composition Model: An Empirical Study

David Lizcano
(School of Computer Science, Universidad Politécnica de Madrid, Spain
dlizcano@fi.upm.es)

Fernando Alonso
(School of Computer Science, Universidad Politécnica de Madrid, Spain
falonso@fi.upm.es)

Javier Soriano
(School of Computer Science, Universidad Politécnica de Madrid, Spain
jsoriano@fi.upm.es)

Genoveva López
(School of Computer Science, Universidad Politécnica de Madrid, Spain
glopez@fi.upm.es)

Abstract: End-user development (EUD) is much hyped, and its impact has outstripped even the most optimistic forecasts. Even so, the vision of end users programming their own solutions has not yet materialized. This will continue to be so unless we in both industry and the research community set ourselves the ambitious challenge of devising end-to-end an end-user application development model for developing a new age of EUD tools. We have embarked on this venture, and this paper presents the main insights and outcomes of our research and development efforts as part of a number of successful EU research projects. Our proposal not only aims to reshape software engineering to meet the needs of EUD but also to refashion its components as solution building blocks instead of programs and software developments. This way, end users will really be empowered to build solutions based on artifacts akin to their expertise and understanding of ideal solutions.

Keywords: end-user development; end-user software engineering; domain experts; domain-specific software development; ecologies of participation
Categories: C.2.4, D.1.7, D.2.2, D.3, H.4.m, H.5.2

1 Introduction

Over recent years, the prosumer concept, introduced by Web 2.0, has interestingly moved into the software development arena. Consequently, the notion of end-user programmer is gaining momentum. End-user programmers are knowledge workers versed in their job, who are neither acquainted with nor interested in software engineering. They develop far more software than professional programmers. In fact, Scaffidi et al. [Scaffidi, 05] estimated that there were over 80 million end-user programmers in American workplaces compared with 2.7 million professional programmers. Forecasts for 2012 suggest an even bigger gap: the number of end users is estimated to grow to 90 million against a much more moderate increase in the figure for professional programmers to just three million [Scaffidi, 05]. End users are

building an assortment of different software, including spreadsheets, multimedia simulations, e-mail filtering rules and more recently dynamic web pages and even applications.

Unfortunately, claims that end users wanting to develop their own software solutions to the problems that they encounter as part of their jobs do not have access to adequate support or a development model are founded [Jones, 03]. There are studies establishing that from 40 to 50 per cent of the software created using end-user development (EUD) techniques and tools does not satisfactorily remedy the problems that it was designed to solve [Lieberman, 06]. This leads, on the one hand, to major financial losses for small- and medium-sized enterprises and large corporations all over the world [Hilzenrath, 03], [Panko, 95], [Robertson, 03] and, on the other, to dissatisfaction, wasted time and unproductive effort on the part of knowledge workers [Davenport, 05], [Cook, 97].

Current EUD research sets out to get the end user more involved in the traditional software engineering process very early on in the software development cycles. These approaches try to elicit the features of the problems to be solved more effectively [Fischer, 09] or offer guidelines and heuristics to instruct users how to design and develop their EUD solutions [Erwig, 09], giving guidance for the testing and debugging process [Fisher, 06]. Launched within the field of end-user software engineering, these initiatives still fail to achieve part of their aims and purposes. Although they have managed to reduce the number and severity of development problems [Ruthruff, 06], they still produce software that is far removed from what would be ideal solutions for end users [Brandt, 09].

The most convincing reason for this failure is that end users are obliged to use and resort, for support, to components, artefacts, processes and algorithms that were originally conceived by and for programmers and that are far removed from the cognitive models of people that know little or nothing about programming [Blackwell, 99]. Remember that end users are acquainted with the problems that they come up against. Their systematic problem-solving process is based on creating data chains among problem-solving components that make sense in the real world from the knowledge worker's viewpoint rather than software elements (functions, objects, data structures, sentences, etc.) not directly related to the real problem [Davenport, 05].

Additionally, existing EUD approaches are often confined to mere spreadsheets and do not offer any support for creating other types of more powerful, richer and/or more versatile EUD solutions [Jones, 03].

The software engineering community cannot ignore the myriad end users that want and need to develop reliable, effective and secure solutions despite being programming illiterate. We must, then, address the needs of the EUD community and try to account for their particularities and characteristics.

To do this, the following three challenges have to be addressed [Curtis, 88]:

1. The tools that end users use and the developments that they carry out suffer from a thin spread of application domain knowledge.
2. There is a need for open, evolvable systems that can adjust to fluctuating, conflicting requirements. Conflicts arise between the evolving world and the software system modelling that world.
3. There is a need to support communication and coordination in a richer ecology of participants with different interests, skills, and background

knowledge. The hardest part of software development is often how to forge a mutual understanding and common ground among all participating stakeholders rather than the technical complexity of the problem.

From our research on this issue [Lizcano, 11], [Lizcano, 08], [Lizcano, 09b], we have gathered that end users cannot be expected to have to cope with development processes, heuristics and steps that they do not know how to use to represent their expertise. The only way of tackling the above challenges is through new software design elements devised for end users that form the groundwork for a software development model. These are the two basic ingredients of any composition model: components translate the problem into a solution from a systemic viewpoint, and a development model specifies the phases and steps to be followed to complete the development based on the above components. The components of a composition model are the conceptual elements that define the composition model and specify how a real-world problem will be understood, modelled and conceived using that composition model [Floyd, 79].

This paper proposes set of components that end users require to be able to understand and compose the software that they develop based on the realistic view they have formed of the problem to be solved. We also present a development model guiding end users through the process of developing solutions based on the above components. But, the main contribution of this paper, however, is a statistical study that, for the first time, empirically demonstrates that the emerging EUD model meets the needs of end users and, thanks to the developed components and model presented here, empowers programming illiterate users to create their own ad hoc software solutions and is also useful for programmers that want to create solutions to support their own work, saving time and effort compared with traditional (object-oriented, imperative, etc.) programming paradigms. This study corroborates the growing body of evidence that end users can create real, reliable and satisfactory solutions provided that they have access to the right building blocks, cooperative and structured repositories that provide such building blocks and finally frameworks that instantiate development models based on catalogued elements.

The remainder of this paper is organized as follows. Section II discusses related work. Section III presents end-user composition model, discussing both the success factor-based component meta-model, the development model of the new composition model and the EzWeb/FAST framework implementing the composition model. Section IV describes the empirical study investigating the adequacy of the components and development model for achieving EUD aims. Finally, Section V discusses the conclusions.

2 Related Work

There are numerous studies [Lieberman, 06] focusing on research into the feasibility and potential of software development by end users. Those studies aim to achieve, extend and assure the success that popular EUD tools, such as spreadsheets, information filtering tools, etc., have already achieved [Jones, 03].

Most have focused mainly on the production of heuristics enabling end users to apply traditional software engineering and development processes [Fischer, 09],

[Erwig, 09], [Fisher, 06]. Actually, they aim to get end users to participate along with software engineers in the early design and development phases. They lack, however, elements, components, processes and artefacts that are familiar to users and their understanding of the problem from a non-programming viewpoint. End users will not be able to properly manage programming resources, because they are not at all like their cognitive model [Jones, 03].

In 2001, publications and research reports began to emerge considering spreadsheets as a new programming paradigm capable of bringing software development to the masses [Burnett, 01]. This work outlines the instructions for the successful use of these tools, and provides insight into the development process for this type of solutions. However, they do not offer guidelines for supporting other more general EUD solutions. Partial research on the EUD field, like [Myers, 06], [Chin, 06], [Riecken, 94], [Chengchun, 05], proliferated. But all these researchers addressed the composition, development and debugging process of particular types of EUD solutions (pervasive computing applications, agent-oriented applications and Web design visual languages) and failed to consider the general-purpose EUD solution as a regular composition model [Riecken, 94], conceived, like any composition model, on the basis of components, but centred on end users instead of programmers.

Large companies like Amazon, Google, Yahoo!, IBM, HP, Sun Microsystems, SAP, Apple and so on have realized that their future on the Internet hinges on adopting a series of basic business principles [Anderson, 06], such as offering software as services (SaaS), ensuring that these services run efficiently in the cloud and can also be used straightforwardly, naturally and simply by the long tail [Burnett, 01].

Consequently, these companies have researched the EUD field and started to publish components that partially conform to the premises for end-user components described in this paper. These components are today empowering millions of non-professional programmers to use repositories of wrapped user-centred back-end services, like [ProgrammableWeb, 11], as a sandbox for finding, remixing, hacking and even exploiting services, resources and wrapped data feeds to thus compose solutions and end-user developments. These design elements are a de facto unstructured implementation of the ideas formalized in the end-user composition model and give an idea of the interest in further expanding the target audience capable of exploiting the ecosystem of user-centred services that many companies are producing (like the Google Chrome Web Store, see [Chrome Web Store, 11], [Myers, 06] and [Chin, 06]).

The need to formallize the end-user solution as a normal composition model is what motivated our research work.

3 End-User Composition Model

As noted above, EUD has been considered as an emerging paradigm [Lieberman, 06], [Jones, 03], but no attempt has yet been made to formalize this discipline as a composition model. Our aim was to formalize this paradigm reshaping software engineering to meet the needs of EUD and refashioning its components as solution building blocks (instead of programs and software developments) [Schroth, 07].

In this section we first present our approach to the new end-user component model, giving an example of a current web service (with SOAP or REST, POX-RPC or similar invocation) wrapped as an end-user component. Then we describe a general end-user development model that will be used to guide end users through the process of developing their own solution based on the composition model components, listing an algorithm that states the development steps paralleling the end user thought model. Finally, we introduce the EzWeb/FAST framework that implements the complete end-user composition model and is used to conduct an empirical study of its feasibility and potential in the software development world.

3.1 A New User-Centred Component Model

User-centred component model should parallel the cognitive model of end users, and their view of the problem and pragmatic problem-solving methods. To define a valid component model, we need to be sure about what end users think, how they want to interact and what they expect of the software solutions. As this would be an empirically prohibitive undertaking, we inspected the most successful EUD solutions to find success factors. The initial component model was the result of abstracting the component models shared by existing EUD approaches. However, EUD solutions are very wide ranging (the component model underlying a spreadsheet has little or nothing to do with an e-mail or RSS filtering solution). For this reason, it is impossible to subsume all the component models of the EUD solutions by directly eliciting the common factor. Rather than aligning the components of all EUD solutions (which would mean mixing, for example, cells and filtering rules, that is, mashing up oil and water), we elicited the success and acceptance factors for all types of EUD tools. We then built a meta-model exploiting all these success factors. This meta-model is the focus of this section.

This research unveiled that the success of EUD solutions is dependent on three interrelated categories of factors, which have until now been addressed separately. These are [Lizcano, 11]:

1. Human factors: any EUD approach should be used and accepted by programming illiterate people. To do this, users must perceive the solutions and the components that they manage at design time as easy-to-use, useful elements, supported and verified by business entities and also having social backing from communities of users tackling similar developments and sharing part of the efforts to achieve collective success [Curtis, 88].
2. User-solution interaction factors: the user-centred components should have cognitive dimensions that fit the thought model of end users, such as the abstraction gradient, consistency, error-proneness, hidden dependencies, premature commitment, viscosity, and so on. Accordingly, the development and runtime components in the EUD domain should conform to a series of principles and heuristics [Chengchun, 05].
3. Successful specialization/functionality trade-off factor: a good tradeoff between the specialization and the functionality of the created solutions is essential in the EUD domain [Jones, 03]. Often EUD solutions are able to create very specialized solutions that are less functional and generally applicable for diverse problems and domains (e.g., spreadsheets), whereas

other solutions offer very diverse functionalities but do not manage to solve entire real-world problems (e.g., Web mashups). EUD solutions should strike a balance between these two factors.

In [Lizcano, 11], we identified a set of success factors for each category. We then mapped these success factors to target features for user-centred component model, which are the groundwork of the proposed composition model. These target features are [Lizcano, 11]:

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 [Schroth, 07b]. 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. Folksonomies, and even in tools like Excel, have used natural language to describe complex functions so that the lay public can comprehend their purpose. 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 [Lizcano, 11].
2. The executing component will usually process some input data to produce outputs. Users should be able to convey the data flow between the components underlying the task to be performed [Lizcano, 08]. As users are programming illiterate, they need to have access to abstractions that fit their mental pattern to model this data flow. As today's EUD tools have shown, 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 the pre- and postconditions that drive the execution of a state machine, which stops users from having to deal with the syntax of the back-end 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. Looking at real examples of these factors, Excel cells, for example, offer users an interface for invoking functions with pre-/postconditions and developing solutions based on the creation of data flows among cells. Other approaches like Web mashups offer widgets that encapsulate service invocations, enabling the user to set up data flows among front-end elements.
3. Users should have access to mechanisms for both spatially and temporally managing the data flow. 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 solution or component, which they are not qualified to do, users should parameterize prefabricated components to meet their needs, or put together finer-grained parts to visually compose more abstract, original and useful components. The catalogue of

prefabricated components and EUD tools for composing new components should offer a full-blown hierarchy of components, ranging from comprehensive, complex and problem domain-specific final solutions 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, mashups, workspaces, gadgets, visual items, data operators and finally back-end resource wrappings.

a) User-centred components will be published in a collaborative and federated solution component marketplace. Software providers, which opted for SaaS (Software as a Service) years ago, can use this catalogue to publish business resources duly packaged according to end-user requirements. This principle would encourage new users to publish their solutions and reuse earlier EUD efforts, reducing the difficulty curve for new creations and producing an exponential benefit, known in economics as network externality [Wu, 04].
b) The development environment suggests components and compositions to users at design time based on their current data flow and light-weight semantic annotations by other users. This information is, in fact, the basis for recommending new elements for users to use to build their solutions and check for errors. This boosts consistency and reduces process viscosity [Jones, 03].

All these components are part of what would be a new end-user component model, with an extensive component hierarchy [Lizcano, 11]. This conceptual model will relate components to each other, composing components from the bottom level of the hierarchy (see Figure 1). End users know how to solve familiar problems systematically using distributed information sources, data flows among these sources, accessing remote resources, etc. They may be able to find an exact (or a similar) solution in a components catalogue published by a software provider or an end user that has already wrestled with a similar problem. In this case, the user will simply have to instantiate and parameterize this solution. More often than not, though, users will have to create their own solutions by mashing up several components, including spreadsheets, Web mashups, etc. (Figure 1-a). Each mashup is composed of multiple workspaces. Workspaces are visual spaces in which a user will set up tangible data flows. Again, users have the option of looking up previously published workspaces in a catalogue or composing them visually from gadgets. Gadgets are the basic and atomic user-centred component (e.g., a spreadsheet cell or a Web mashup widget); they are the minimum component that makes sense to a programming illiterate user and fulfills the premise of offering users a visual interface for managing a wrapped resource (function, service, data access). Thanks to the support of software providers, these are the most populous elements in today's end-user catalogues (Figure 1-b). Through visual and semantically-driven wiring, end users will be able to build these elements into their workspaces and create data flows between them (Figure 1-c). These flows will help to convey the knowledge workers' systematic knowledge, explicitly exploiting their problem expertise. If the catalogue does not contain the gadget that the end users need, they will have to use components, finer-grained end-user components (Figure 1-d), to create this component: visual items, data operators

or back-end resource wrapping. The most important of these elements are the remote resources. Remote resources enter new data in the flow devised by the user and require the backing of software providers capable of offering the resources to users. The new data entered in the solution will be managed by piping operators, like filters, selectors, mixers, etc. Finally, the visual elements will display information to the user and capture their actions on the gadget. This is an ordinary model-view-controller, designed, in this case, to be handled and used by programming illiterate end users. By piping all these building blocks, end users will be able to design and add their own gadget and build their ideal solution.

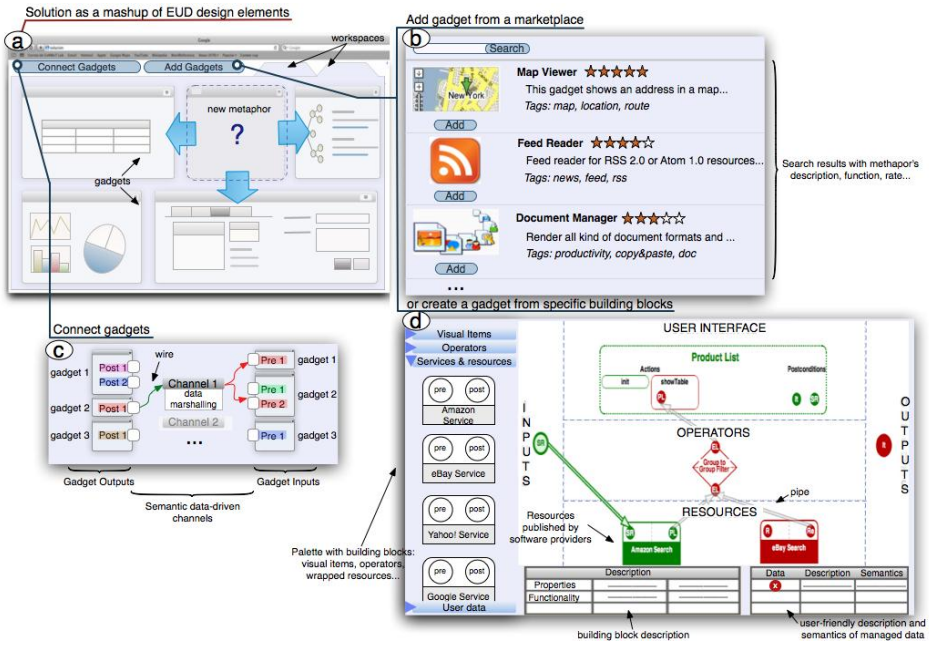

Fig 1. Development of rich end-user solutions through the end-user composition model

This conceptual model is subsumed or instantiated by all the known EUD applications [Lizcano, 09b], and, if exploited to the full, will be able to solve the challenges listed by Bill Curtis, Herb Krasner and Neil Iscoe in 1988 [Curtis, 88]. User-centred components and their relationships should take EUD beyond solutions that are based exclusively on designing a set of spreadsheets to process data, creating macros or chaining data filters.

Having defined the target features of the components of the new user-centred component model, we can formalize it that is useful for describing the architecture of an end-user solution (Fig. 2). To formalize this model, we employed the MOF (meta-object facility) notation. MOF is an international standard (ISO/IEC 19502:2005) [OMG, 06] enabling the creation of a strict level-3 meta-modelling schema [Sobek,

05], which offers the possibility of running or checking schema instances or subsumptions in UML notation (descending to modelling level 2). This way, it can output or validate component diagrams of different end-user development tools.

The model includes the *design element* as a basic component of the user-centred component model. This element is composed of a user-centred *visual interface* for accessing a *recovered resource*. Any component will be linked, in the final solution, with other components 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*. The development environment suggests components and compositions to users at design time based on their current dataflow and lightweight semantic annotations by other users. This information is, in fact, the basis for recommending new elements for users to use to build their solutions and check for errors, boosting consistency. It also reduces process viscosity.

The end-user components will be published in a business marketplace-style collaborative and federated *catalogue*. Software providers, which opted for SaaS years ago, can use this catalogue to publish resources duly packaged according to end-user requirements. 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 be adapted to the end-user cognitive model and specific end-user knowledge, meaning that 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 *back-end resources* (simple data operators, like filters, concatenators, etc., or recovered services). Each element in this hierarchy is adapted to a different level of abstraction in the end-user cognitive model: the *full solution* fits the systemic view that the user envisages for tackling 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 *gadgets*, where a gadget is a visual element that manages user interaction with a particular remote resource. This gadget may present a single view (for example, an Excel cell or a single form) or a screenflow (such as a survey composed of several forms) for the user to interact with the remote resource or resources associated with the gadget. Each of these visual interaction items is termed *resource representation*. A resource representation is composed of the *view* and the *back-end resource*. The back-end resource is composed of *operators* and *service wrappings.* This component model is instantiated as the different EUD solutions existing today [Lizcano, 11]. It is not, however, easy to build a system that instantiates the entire model and supports such a level of scalability, globality and interoperability among users, save in the case of the Internet. It is in the Web environment where our conception of the end-user composition model makes most sense and is likely to reach its full potential in terms of functionality and success. The EUD phenomenon has already left an imprint on the Web through mashup-based compositional applications created by iGoogle, Yahoo!Pipes, OpenKapow, etc., over the last few years. These applications subsume

the presented model. All these EUD tools are based on visual elements (commonly known as widgets, a shortened form of web gadgets) that represent data or special-purpose data processes (displaying an address on a map or a short list of news). The best tools establish a dataflow among these visual elements where a new data item in one element leads all the collaborative interfaces to take a computational step. This is a spreadsheet-like approach, save that each element displays a richer visual interface and invokes particular remote services, resources or distributed data as recovered services.

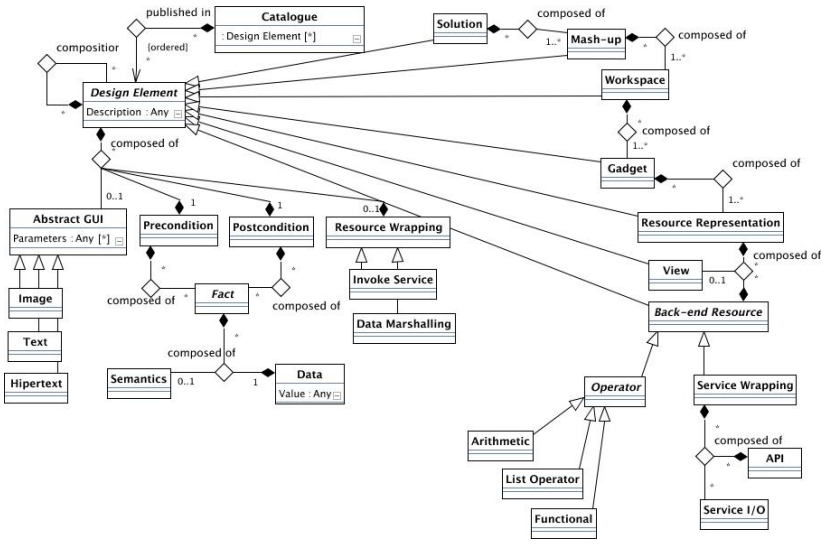

Fig. 2. MOF 2.0 end-user composition model component model

These service wrappings are the atomic design elements of the end-user component model; they are the smallest pieces that a programming illiterate user can handle and understand. These elements, composed of an API and some inputs and outputs, are especially abundant on the Internet thanks to Web services ecosystems, as these Web services are really easy to transform into *recovered services* components. The following is a specific example of a Yahoo! Web service using its search engine, transformed into a user-centred component. First, the Web service inputs and outputs have to be mapped to the pre- and postconditions of the end-user component (see Fig. 3).

```
<?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. 3. Mapping simple EUD data structures (pre-/postconditions) to the service parameters. An XML fragment of a resource adapter configuration file that defines the mapping of EUD facts to the Yahoo Search Web service

Additionally, it is necessary to assure that the Web service is invoked when the precondition of the component is satisfied and adapt the results returned by the service to the postconditions that are meaningful in the EUD field. This means developing a small adaptor for the service according to a traditional development process using JavaScript, for example (see Fig. 4).

```
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. 4. JavaScript service adaptor. The variables declared in the adaptation have to be previously declared in code, casting types and programming the remote invocation.

3.2 End-User Development Model

Having defined the component model, it is necessary to describe the development model whereby a programming illiterate user will be able to tackle a real problem and relate and use components together to build a software solution.

In this section we present the end-user development model as an algorithm. The algorithm establishes the steps to be taken by the end user and how model components are related, composed and interact with each other to build the final solution.

End-User_Development procedure enables an end user to solve a real problem by instantiating, interrelating and composing components of variable abstraction. This procedure relies on the *End-User_Analysis function*, which aims to decompose the problem into problem-solving components that make sense to the user. When an atomic component, containing interface and functionality (gadget), has not been fabricated by another user or a software provider, the *End-User_Development procedure* offers the heuristic for building this component through element visualization, services invocation and dataflow management. Finally, the

Test_Solution and *End-User_Deployment* procedures are responsible for helping the user to test and deploy the final solution.

```
1: procedure End-User_Development(realProblem)
2:         searchfinalSolution from catalogue equal to realProblem
3:         if finalSolution has not yet been created then
4:                 create solutionNarrativeDescription, and
5:                 search partialSolution from catalogue, and
6:                 End-User_Analysis (solutionNarrativeDescription,
                   partialSolutionby ref, 1)
7:         else
8:                 parameterize finalSolution
9:         end if
10: end procedure
11:
12: function End-User_Analysis(solutionNarrativeDescription,partialSolution,
      iteration)
13:   if partialSolution solves solutionNarrativeDescription then
14:         Test_Solution (solutionNarrativeDescription, partialSolution, error
          by ref)
15:         if error then
16:                 adderror to solutionNarrativeDescription
17:                 End-User_Analysis (solutionNarrativeDescription,
                    partialSolution by ref, iteration)
18:         else
19:                 End-User_Deployment (partialSolution)
20:         end if
21:   else
22:         case iteration = 1{EUD centred mashup-type abstraction}
23:                 search mashup from catalogue subsumption of
                    solutionNarrativeDescription
24:                 add mashup to partialSolution
25:                 interconnect mashup to partialSolution following mashup's semantics
26:                 if mashup has not yet been created then
27:                         End-User_Analysis (solutionNarrativeDescription,
                            partialSolution by ref, iteration+1)
28:                 else
29:                         End-User_Analysis (solutionNarrativeDescription,
                            partialSolution by ref, iteration)
30:                 end if
31:         case iteration = 2{EUD centred workspace-type abstraction}
32:                 search workspaces from catalogue subsumption of
                    solutionNarrativeDescription
33:                 add workspace to partialSolution
34:                 interconnect workspace to partialSolution following workspace's
                    semantics
35:                 if workspace has not yet been created then
36:                         End-User_Analysis (solutionNarrativeDescription,
                            partialSolution by ref, iteration+1)
37:                 else
38:                         End-User_Analysis (solutionNarrativeDescription,
                            partialSolution by ref, iteration-1)
39:                 end if
40:         case iteration = 3{EUD centred on gadget-type abstraction}
41:                 search gadget from catalogue subsumption of
                    solutionNarrativeDescription
42:                 add gadget to partialSolution
```

```
43:                 interconnect gadget to partialSolution following gadget's semantics
44:                 if gadget has not yet been created then
45:                         create emptynewGadget
46:                         Resource_Development
                            (solutionNarrativeDescription, newGadget by ref)
47:                         add newGadget to partialSolution
48:                         End-User_Analysis (solutionNarrativeDescription,
                            partialSolution by ref, iteration)
49:                 else
50:                         End-User_Analysis(solutionNarrativeDescription,
                            partialSolution by ref, iteration-1)
51:                 end if
52:         end case
53:   end if
54: end function
55:
56: procedure Resource_Development (solutionDescription, gadget)
57:         search view from catalogue subsumption of solutionDescription
58:         add view to gadget
59:         for all back-endSource in solutionDescription do
60:                 search back-endSource from catalogue
61:                 add back-endSource to gadget
62:         end for
63:         while solutionDescription's out ≠ gadget's out do
64:                 search operator compatible with solutionDescription's in and
                    gadget's operator's out from catalogue subsumption of
                    solutionDescription
65:                 add operator to gadget
66:         end while
67: end procedure
68:
69: procedure Test_Solution (solutionDescription,
      partialSolution, error)
70:         for all solutionDescription's testCase do
71:                 error = test partialSolution following testCase
72:                 if error then
73:                         error = write output error
74:                 end if
75:         end for
76: end procedure
77:
78: procedure End-User_Deployment (partialSolution)
79:         publish and describe partialSolution in catalogue
80:         parameterize partialSolution
81: end procedure
```

As the above algorithm shows, the end-user development model focuses on problem analysis and component creation. Problem analysis aims to decompose the problem into increasingly fine-grained end-user components, whereas component creation assembles components from their components if the elements are missing from the component catalogue.

It is precisely this catalogue that plays a major role and will be a key factor in the achievement of the end-user composition model objectives. This algorithm has been implemented through a real EUD framework, explained in next section.

3.3 FP7 FAST/EzWeb: Developing an EUD framework

By devising a new composition model for end-user developments, we can conduct a structured and objective analysis of EUD solutions and proposals to find out their strengths and weaknesses and establish guidelines for improvement, enable the interoperability of several heterogeneous EUD tools based on generally applicable common principles, and create the groundwork for the end-user composition model defined according to the elicited information about current tool success factors rather than from the software engineering angle [Soriano, 07].

The construction of a framework empowering end users to build their own software solutions was the focus of our research, which statistically evaluated the success of both the framework and the solutions created by the users. The aim was to boost and shed light on the EUD domain, which was forbidden territory to users unacquainted with programming issues, services orchestration, etc., who generated unreliable software, or a disappointment to users that saw how their valuable domain knowledge was misspent on mere spreadsheets, business process management applications, data tables or simplistic scripts [Lizcano, 08].

The result was the EzWeb/FAST framework (Fig. 5.) (see http://ezweb.morfeo-project.org/lng/en and http://fast-fp7project.morfeo-project.org/lng/en respectively). EzWeb/FAST was the open-source product of research by two international R&D project consortiums [Lizcano, 08]. Fast and Advanced Storyboard Tools (FAST) Project [FAST, 11] is a Small or Medium-scale Focused Research Collaborative Project (STREP) supported by the European Commission under its 7th Framework Programme (FP7). This framework instantiates the above end-user composition model and services as a test bench for checking if the created component model achieves its objective: end user access to the tools that they need to create software solutions to support or boost their knowledge work, irrespective of their programming knowledge [Lizcano, 09].

Fig.5. Example of an EUD solution (trip planner) built using EzWeb/FAST. An agenda gadget was built from visual resources, services and data operators

4 Empirical Study about the proposed End-user Composition Model

As mentioned above, the main contribution of this paper is a statistical study that aims to evaluate how effective the end-user composition model is at empowering end users to develop their own ad-hoc solutions to tackle their real problems. As far as we know, no other study empirically comparing EUD with traditional programming in terms of development time and effort has been reported. The results and findings of this study should be leveraged to improve current EUD approaches and tools, thus furthering success, acceptance and outcomes.

4.1 Design

When developing the empirical evaluation of the end-user composition model, we consider two major factors for quantification: how satisfied both end and technical users are with the model for developing solutions and how successful they are at building an operational solution from the description of a real problem.

To conduct the statistical survey of how successful the end-user composition model is, we asked users to rate the EzWeb /FAST tool implementing the composition model. To do this, we used a sample of 100 users. This sample is characterized as shown in Table I below.

Characterization	End users (50)	Technical or advanced users (50)	Total (100)
Gender			
Male	26	25	51
Female	24	25	49
Age			
< 20 years	9	10	19
20-34 years	12	11	23
35-49 years	11	12	23
50-64 years	10	10	20
> 65 years	8	7	15
Educational Attainment			
Secondary School	12	12	24
Vocational Training	13	13	26
Bachelor's Degree	12	13	25
Master's Degree	13	12	25
Employment			
Student	13	15	28
Researcher	14	18	32
Employee	23	17	40

Table I. Sample characterization

The sample should properly characterize all users that undertake EUD today. A priori, the sample does not appear to be biased as regards user gender, age and employment. We ran an ANCOVA study to demonstrate that there are no statistical data to indicate that the sample is biased. Accordingly, the study checked how correlated the result of the evaluation was as a variable dependent on gender, age,

educational attainment and employment. As shown later, this analysis statistically proves that the end-user composition model rating is completely independent of respondent age, gender, employment or education, and therefore there is no bias in the sample. Therefore, the choice of the 100 users is valid (from the statistical viewpoint) for running the survey of the end-user composition model.

The characterized sample was asked, during the evaluation, to solve a specific problem with whose domain they were unacquainted. Using the proposed framework and an abundant set of design elements conforming to the end-user composition model principles, (see [EzWeb Catalogue, 11]), users were asked to develop a compositional application to plan business trips. They had to create a Web application that searched for and booked means of transport and hotels, and consulted tourist information on destinations listed on a personal agenda. This solution also had to control the financial costs against a spreadsheet that included a budget. The problem is detailed in the attached appendix.

Subjects used the EzWeb/FAST tool that implements the end-user composition model to solve this problem. The complete development process is explained at http://apolo.ls.fi.upm.es/eud/solution_development_process.pdf. All users had to complete a period of learning, a requirements study and analysis, and the final development. The end-user composition model teaching/learning period was confined to a 20-minute oral presentation and a 10-minute viewing of multimedia material (see [FAST Manual1, 11] and [FAST Manual2, 11]).

The study focused on two research questions that were measured independently:

- RQ1. Is the end-user composition model adequate for end users? This research question was examined using a variable termed *mean rating e*xtracted from a survey of end users.
- RQ2. How long did it take the user to build a valid solution using the end-user composition model and using traditional techniques? This research question was measured using the variable termed *time* empirically observed during the experiment.

Whereas the measurement of a time interval requires no further explanation, the measurement of sample satisfaction with the end-user composition model does need to be described in more detail. To take this measurement objectively, we built a 24-question survey concerning different aspects of the end-user composition model. Users had to give each question a rating of between 1 and 5 (five-point Likert scale), where 1 means *I totally disagree* and 5 means *I totally agree*.

The survey contains questions concerning only 12 key issues about the end-user composition model (Table II, allocated in Apendix II). These questions were then grouped into five blocks or sections, and six preliminary questions were added about the respondents' personal particulars (name, ID card no., gender, age, educational attainment, etc.) in order to characterize the sample. The questions were designed according to the principles expounded by Lehmann et al. [EzWeb, 11] and Jessen [Jessen, 78]: back-up questions were used to check response and process consistency (several questions address the same general topic to check that users answer them consistently), and questions were phrased affirmatively (where the highest score is 5 points) and negatively (where the maximum score is 1) to prevent automatic or

unmeditated responses, where respondents tend to consistently score all items either high or low without thinking about the meaning of the response.

A major concern throughout the study was to prevent external factors from affecting the study or leading to the misinterpretation of the available objective data. This called for a number of checks and verifications. Specifically, we used statistical techniques to prevent the following threats:

- Threats to external validity, which limit the extent to which results can be generalized. The results will not be generalizable if the problems that were set for the sample to solve do not represent real scenarios routinely faced by users. To reduce this threat, we gathered real problem statements from a web survey of end users at the http://sites.google.com/site/fastonlinecontest web site. Hundreds of users described their routine EUD problems on this page, and an experiment was designed that combined most of the characteristics and aspects identified from the results.
- Threats to internal validity, which can lead to biased outcomes or incorrect interpretations. The types of components and components specifically evaluated in the study could affect the final results. For this reason, the sample had access to all the real design elements that major software developers, like Google, Amazon, Microsoft, Apple or Sun, propose as composable services and resources for technical users, which have been mapped to user-centred components in the EzWeb/FAST framework.
- Threats to construct validity, which affect the actual measurement of the response variables, preventing a proper evaluation of the fact or hypothesis to be tested. This is the biggest threat to this study. To assure that the metrics used properly captured the feedback from end users and technical users, objective and consolidated measures were used to evaluate each research question. On the one hand, the real development time, which we measured live during the experiment is a totally objective and reliable measure. As regards the adequacy of the user model and user satisfaction, measured by means of a survey, a pilot process was enacted to select the items that the survey was to contain. An initial sample of 50 users was surveyed about a set of 100 items or questions. The scores of each individual were evaluated, and each item was correlated with the sum total. Later 25% of the highest-scoring individuals and 25% of the lowest-scoring individuals were selected, and the mean between-group difference was calculated for each item. The final survey was built using the 25% of questions that had a high r (a correlation of the item to the final result greater than 0.5) and a high $[\overline{X}_{max} - \overline{X}_{min}]$. This, together with a mixture of questions phrased affirmatively and negatively to prevent acquiescence and the use of repeated questions to check respondent consistency (question pairs had to have a correlation greater than 0.5 points), assures a high study validity.

4.2 Results

All 100 individuals completed the EUD application that conformed to the set requirements. There follows a description of the results output in terms of user

satisfaction with the end-user composition model (RQ1) and development time required to apply the model (RQ2).

4.2.1 RQ1. Is the end-user composition model adequate for end users?

To answer this research question, we have to analyse the survey results. Table III below shows the user ratings (mean score) in response to each survey question (Q7 to Q30) for the whole sample.

Question No.	End User	Technical User	Total Score (all users)
Q7	4.24	4.06	4.15
Q8	4.28	4.12	4.20
Q9	4.08	4.00	4.04
Q10	4.40	4.38	4.39
Q11	4.26	4.06	4.16
Q12	4.18	4.04	4.11
Q13	3.52	3.26	3.39
Q14	3.92	3.74	3.83
Q15	4.48	4.22	4.35
Q16	4.16	3.84	4.00
Q17	4.28	4.00	4.14
Q18	4.20	3.90	4.05
Q19	4.18	3.98	4.08
Q20	4.52	4.32	4.42
Q21	4.48	4.38	4.43
Q22	4.12	4.02	4.07
Q23	3.98	4.04	4.01
Q24	4.02	3.56	3.79
Q25	4.20	3.98	4.09
Q26	4.48	4.18	4.33
Q27	4.36	4.30	4.33
Q28	4.02	3.56	3.79
Q29	3.98	3.92	3.95
Q30	4.16	3.88	4.02
TOTAL	4.19	3.99	4.09

Table III. Five-point Likert score for the whole sample

The scores have all been normalized on a scale of 1 to 5, where 1 is the lowest score and 5 is the highest score. To assure response consistency, numerous questions (Q12, Q13, Q14, Q16, Q19, Q20, Q21, Q24, Q25, Q26, Q27 and Q29) were stated inversely, that is, 1 is the highest and 5 is the lowest score. For all these questions, the score was inverted applying the formula: normalized score = score * (-1) + 6. This way, all the scores have the same scale and meaning, and can all be operated on equally. In anticipation of the rating results being different for the surveyed end users and technical users, we split the scores depending on the type of users doing the evaluation.

Table IV (row 1) shows the descriptive statistics for the rating given by users and the distribution of the sample fitted to the normal distribution with a mean of 4.09 points (on a scale of 1 to 5) and a standard deviation (σ) of 0.38.

						95% Confidence Interval for Mean			
	N	Mean	Std. Dev (σ)	Variance	Std. Error (SE)	Lower Bound	Upper Bound	Minimum	Maximum
EUD model rating (1-5)	100	4.09000	0.389307	0.151560	0.038931	3.326972038	4.853027962	2.960	4.880
End-user rating	50	4.19	0.327787	0.107444	0.046356	3.547550177	4.832449823	3.580	4.880
Technical user rating	50	3.99	0.422729	0.178700	0.059783	3.161466661	4.818533339	2.960	4.880

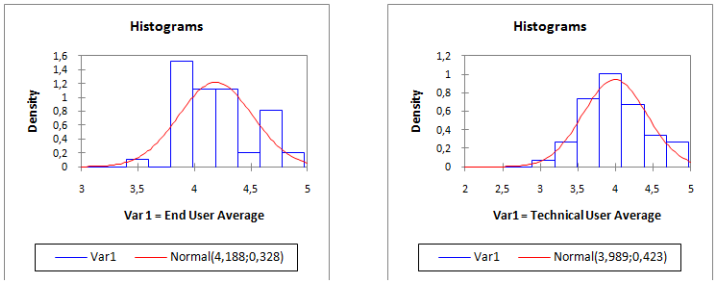

Table IV. Descriptive statistics for the overall rating

The calculated σ and S_E values are related by the fact that partial deviations are overly variable depending on the selected subsamples of the population under study. This is because the characterization of the population includes, as the ANCOVA of the regression model of the mean rating variable (see Table V) shows, a variable that is significant for the study namely whether or not the user has programming expertise.

Table IV (see rows 2 and 3) also shows the descriptive statistics of the distribution of ratings given by end users and technical users, and the normalized distribution of the two samples (end-user and technical-user ratings). There is in fact a sizeable difference in the mean rating variable depending on the qualitative variable measuring programming expertise. Looking at the results in Table V, programming illiterate end users rated the end-user composition model more positively than programmers.

We conducted an ANCOVA analysis (table V) in an attempt to explain the quantitative "final rating" variable depending on the other quantitative and qualitative variables gathered to characterize the sample. This way, we aimed to empirically check whether age, educational attainment, employment or previous EUD expertise cause the rating to vary. This analysis will be able, on the one hand, to check that the selected sample is not biased and, therefore, does not contaminate the conducted survey and, on the other, to verify that the only variable that appears to have a direct effect on user satisfaction with the end-user composition model is previous programming expertise.

Goodness of fit statistics

Observations	Sum of weights	Df	R^2	Adjusted R^2	MSE	RMSE	MAPE	DW	Cp	AIC	SBC	PC
100.000	100.000	64.000	0.395	0.065	0.142	0.377	5.462	1.157	36.000	-167.99	-74.21	1.285

Analysis of variance:

Source	df	Sum of squares	Mean squares	F	Pr > F
Model	34	5.932	0.169	1.196	0.264
Error	65	9.072	0.142		
Corrected Total	99	15.004			

Computed against model Y=Mean(Y)

Type I sum of squares analysis:

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

Type III sum of squares analysis:

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

Table V. ANCOVA analysis of the sample

Analysing the study, we find that the coefficient of determination R^2 is very low (0.395). This indicates that there is a high percentage of variability in the modelled variable so that gender, age, educational attainment, employment and previous experience appear to explain only 30% of the rating data. The other values are due to other unknown variables. This value of R^2 and adjusted R^2 suggests that the rating of

the end-user composition model is largely (70%) independent of the characteristics of the users rating the model (Figure 6). First, this result validates the sample, indicating that there is no bias related to the qualitative and quantitative variables characteristic of the users and to their recruitment for the study. The regression model shows a horizontal and vertical dispersion of predictions, with error ranges from 2.5 to 5 points (out of 1 to 5 points), meaning that the *mean rating* variable is completely independent.

Having validated the surveyed sample, it is worth mentioning that the ANCOVA analyisis (Table V) indicates that the selected explanatory variables cannot be considered to be the source of a significant amount of model information (Pr > F = 0.264 >> 0.01). The model is not significant because the rating of the model is independent of the characterization of the sample, and this means that we can again assume that the a priori identified survey bias does not mean that either the rating or the results are biased a posteriori

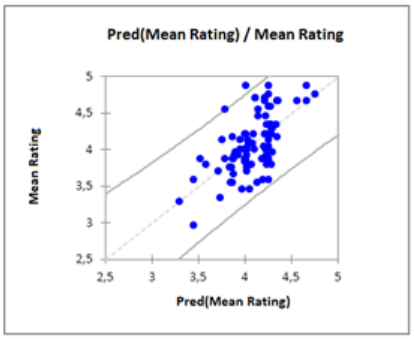

Fig. 6. Fit of the end-user composition model rating based on the regression model

Analysing the results of the sum of squares analysis in Table V, we find the variable that has most impact on the rating. Of the studied variables (age, gender, educational attainment, employment, previous EUD experience and programming expertise), the variable with the greatest Fisher F-distribution is previous programming expertise (F=1.407). Pr > F is equal to 0.146 (the closest to 0.01) for that variable. Therefore, we can infer that the aspect of sample characterization that is most statistically significant for the rating is whether or not the user has programming expertise [Lehmann, 05]. The other variables have a weaker Fisher F-distribution (and, therefore, less impact on the rating). The variable with the least impact on the model is gender, followed by previous EUD experience and then employment and educational attainment. Judging by the probabilistic values Pr > F, everything appears to indicate that previous EUD experience does not affect the rating of the new model at all. This way, respondent age, employment, educational attainment, etc., will not alter their rating of the end-user composition model.

Finally, the extent to which each variable has an impact on the end-user composition model rating can be quantified using a regression model and its standardized coefficients. Figure 7 lists and plots the model coefficients.

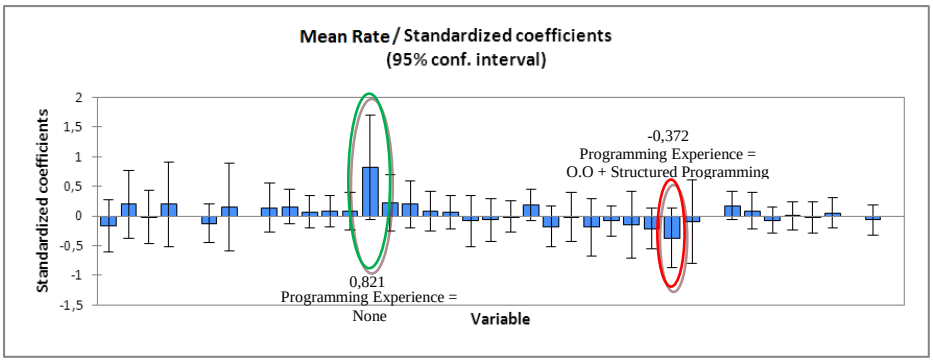

Fig. 7. Impact of each sample characterization variable on rating

The described end-user composition model does meet the needs of users, especially end users. The factor that most positively affects the end-user composition model rating is that users have no type of programming expertise, something that has already been verified, reasoned and proven. On the other hand, the factor that most negatively affects the rating is that users are experienced in other composition models (like object-oriented and structured programming (see Fig. 7)), that is, users with a lot of programming expertise are the ones that rate the end-user composition model worst. This is because their cognitive model is oriented to components proper to the programming world, and they are less familiar and at ease with components used by domain experts.

4.2.2 RQ2: How long does it take a user to develop a valid solution?

Apart from a survey-based evaluation of the views of the users of the end-user composition model, the statistical survey also measured the time that it took users to develop a full software solution to solve the set problem. The focus was on ascertaining whether it took technical users less time to develop an EUD solution than end users without any type of programming expertise.

The mean development time using EUD was 8.39 minutes, whereas it took end users and technical users on average 8.32 and 8.46 minutes, respectively, to develop the application.

According to an ANOVA study, it does not take end users any longer to develop their solution than technical users.

To confirm these results, we set a series of six standard problems (see http://apolo.ls.fi.upm.es/eud/ problems_description.pdf), which could be solved using three to five different types of services, data or heterogeneous resources. After asking the technical users to solve these problems with and without the end-user composition model, the statistically significant results indicated that the model takes at least 100 times less time than is required for traditional development, provided that the necessary resources are packaged and consistently aligned with the end-user composition model (Figure 8). Additionally, the model manages to simplify the process and, most importantly, agglutinate programmers in much shorter and focused development time spans than any of the programming techniques.

The graph shows that the use of the end-user composition model empowers all users (technical or otherwise) to complete the development in about eight minutes, whereas traditional programming is only an option for technical users, whom it takes 1220 minutes to solve the set problem. Although these data are problem dependent, the saving in time and effort is notable in all cases. Note also that there are very large variations in development time without EUD, ranging from 1100 to over 1600 minutes (Figure 8) for the proposed problem. This suggests that the traditional programming puts user ability, intellect and initiative more to the test, and whether or not the user is inspired by the particular problem can lead to variations of up to 500 minutes in development time (more than an eight-hour working day). However, the end-user composition model reduces the development time span enormously, and any user (even non-programmers) can finish the solution within a time range differing by only 15 minutes (Figure 8) at most (between the maximum and minimum development time observed in the study using the end-user composition model).

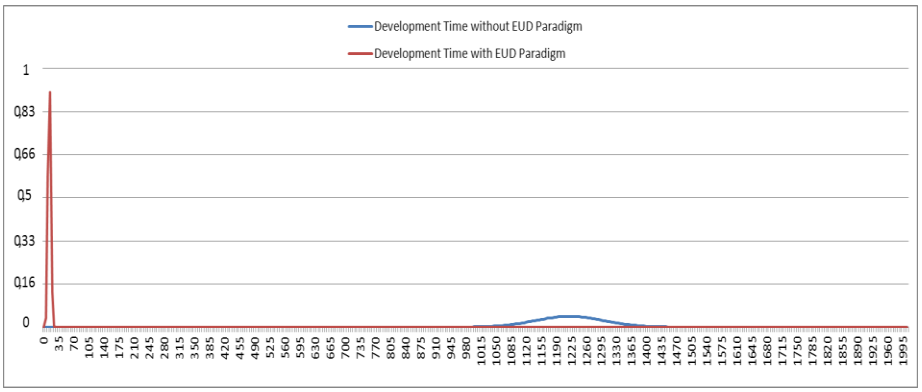

Fig. 8. Development time with and without EUD

4.3 Discussion

The results obtained with respect to RQ1 show that the end-user composition model meets end users' needs. These users will have access to the option of building their own solutions to meet the problems that they encounter in their routine knowledge work, without having to have programming expertise. To do this, it is necessary to provide users with catalogues fed with end-user components, as well as frameworks for accessing these catalogues and implementing the end-user composition model.

A relevant result was that the end-user composition model is better suited to the cognitive model of programming illiterate users than to people used to programming. This is a first among existing composition models. This suggests that the steps of the scientific method applied to build the composition model correctly deduced what elements end users envisage using and what components they understand in order to transform an imagined solution based on their expertise into real software.

This model also achieves comparable results among both young and older users, men and women and people with different educational attainment and jobs. This is a sound enough empirical basis to claim that the approach helps users of all types to build their own low-cost solutions without having to resort to off-the-shelf software (which, being general-purpose, is not tailored to their changing and complex problems) or to pay out large sums of money for ad-hoc software built by traditional software engineers.

The results for RQ2 show that the end-user composition model uses components and a development model that is equally accessible for all users and requires similar effort irrespective of programming expertise.

The end-user composition model offers end users a solution for developing, testing and debugging, and using software that would be out of the question with traditional paradigms. Additionally, programmers will be able to build lightweight developments more effectively, quickly and cheaply thanks to the end-user composition model, developing the solution in one hundredth of the development time that it would take without prefabricated user-centred components. For these premises to be true,

however, software providers have to populate the collaborative catalogues underpinning the end-user composition model.

It also has another benefit for these users: thanks to component simplicity they will be able to recover new resources and services not previously adapted to the model, thereby solving problems based on partial EUD solutions without having to tackle the whole problem traditionally from scratch and also extending the end-user components available to the other users.

As the number of model users grow, the number of components, partial and full solutions should also increase, thereby potentially attracting more and more users. In face of spiralling component use, software providers should, likewise, set about recovering and publishing services as end-user components, thereby leading to an ideal ecosystem for getting millions of end users from all over the world to develop useful and effective solutions.

The complete statistical study is available at [Lizcano, 11b], including the original survey, a description of components used in the experiment, how surveyed users manage their composition process, an evaluation of solutions quality (robustness and security) and so on.

In addition, the framework that we employed to run the study is available at [EzWeb Demo, 11] and [FAST Demo, 11] and can be exploited after registration. This framework and the proposed end-user composition model are now being used by Spanish public administrations to promote new digital spaces for citizen interaction. Saragossa Town Council (see [Tejo-Alonso, 11]) is using the end-user composition model and its software components to empower its citizens to compose their own software solutions to complete bureaucratic formalities, access citizens' services, report breakdowns or incidents on public thoroughfares, etc.

5 Conclusions and Future Work

Large companies like Amazon, Google, Yahoo!, IBM, HP, Sun Microsystems, SAP, Apple and so on have realized that their future on the Internet hinges on adopting a series of basic business principles, such as offering SaaS, ensuring that these services run efficiently in the cloud and can also be used straightforwardly, naturally and simply by the Long Tail. This philosophy is today empowering millions of non-professional programmers to use repositories as a sandbox for finding, remixing, hacking and even exploiting services, resources and wrapped data feeds to thus compose solutions and end user developments. The end-user composition model is also a structured and standardized way of offering such components to programming illiterate end users. The use of this composition model would thus further expand the target audience capable of exploiting the ecosystem of user-centered services that many companies are producing.

This could lead to a shift in how software is developed to support knowledge workers, opening the doors of the cathedrals of traditional software and SOA-type architecture engineering and letting in the everyday hustle and bustle of end users working in a hierarchyless and barrier-free bazaar.

And the support for this community of knowledge workers to cooperate and exchange solutions and expertise is a catalogue where both software providers and users of all types and programming abilities give free rein to their collective intelligence and innovativeness. This way, resources and components can be used in ways that their creators would never even have imagined, and individual end users will find more and more parts that fit their problem-solving approach and devise their own particular and changing solution as the best way of getting their job done. The statistical study of the end-user composition model suggests that a properly fed catalogue will achieve sufficient network externality for end users and programmers to gain enormous benefits from the EUD approach.

But, is it possible to incentivize users, groups and providers to publish their creations and spend time populating the catalogue? If users and providers find the catalogue to be useful and the foundations for compensating the reputation and hard work of anyone publishing in such repositories are properly laid, the gift culture will assure that users and providers go about homesteading the noosphere.

With this catalogue, end users will be able to create solutions to their everyday problems, giving up the tedious practices of manually establishing the data flow between applications, Web pages, calls to resources, etc. Also, small- and medium-sized enterprises, which do not have the funds to commit major software development investments, will gain access to tools for developing ad hoc solutions tailored to their problems. And large corporations will be able to publish some of their products and business process management applications for users, whether they are company customers or employees, to exploit, adapt and parameterize to their needs, setting up a feedback cycle that would be unthinkable in traditional software development processes.

Acknowledgements

We would like to thank the experiment participants, who although they remain anonymous, played a fundamental role in this study.

References

[Scaffidi, 05] Scaffidi, C., Shaw, M., and Myers, B.: Estimating the Numbers of End Users and End User Programmers, Proc. 2005 IEEE Symp. Visual Languages and Human-Centric Computing, 2005, Dallas, TX, USA. pp. 207-214.

[Lieberman, 06] Lieberman, H., Paternò, F., Klann, M., and Wulf, V.: End-user development, Germany: Kluwer/Springer, pp. 1–8.

[Boehm, 01] Boehm, B. and Basili, V.R.: Software Defect Reduction Top 10 List, Computer, vol. 34, no. 1, pp. 135-137, Jan. 2001.

[Hilzenrath, 03] Hilzenrath, D.S.: Finding Errors a Plus, Fannie Says; Mortgage Giant Tries to Soften Effect of $1 Billion in Mistakes, The Washington Post, 31 Oct. 2003.

[Panko, 95] Panko, R.: Finding Spreadsheet Errors: Most Spreadsheet Errors Have Design Flaws that May Lead to Long-Term Miscalculation,"Information Week, p. 100, May 1995.

[Robertson, 03] Robertson, G.: Officials Red-Faced by $24m Gaffe: Error in Contract Bid Hits Bottom Line of TransAlta Corp., Ottawa Citizen, 5 June 2003.

[Davenport, 05] Davenport, T.H.: Thinking for a living: How to get better performance and results from knowledge workers, Boston, MA: Harvard Business Press, 2005.

[Cook, 97] Cook, C., Burnett, M. and Boom, D.: A Bug's Eye View of Immediate Visual Feedback in Direct-Manipulation Programming Systems, Proc. 7th Workshop on Empirical Studies of Programmers, Oct. 1997, Alexandria, Virginia, USA. pp. 20-41.

[Fischer, 09] Fischer, G., Nakakoji, K. and Ye, Y.: Metadesign: Guidelines for Supporting Domain Experts in Software Development, IEEE Software, vol. 26, no. 5, pp. 37-44, Sept./Oct. 2009.

[Erwig, 09] Erwig, M.: Software Engineering for Spreadsheets, IEEE Software, vol.26, no.5, pp.25-30, Sept./Oct. 2009.

[Fisher, 06] Fisher, M., Rothermel, G., Brown, D., Cao, M., Cook, C. and Burnett, M.: Integrating automated test generation into the WYSIWYT spreadsheet testing methodology, ACM Trans Softw. Eng. Meth., vol. 15, no. 2, April, 2006.

[Ruthruff, 06] Ruthruff, J. R., Burnett, M., Rothermel, G.: Interactive Fault Localization Techniques in a Spreadsheet Environment, IEEE Trans. Software Eng., pp. 213-239, April, 2006

[Brandt, 09]Brandt, J. P., Guo, J., Lewenstein, J., Dontcheva, M., Klemmer, S. R.: Opportunistic Programming: Writing Code to Prototype, Ideate, and Discover, IEEE Software, pp. 18-24, Sept./Oct., 2009.

[Blackwell, 99] Blackwell, A. and Green, T. R. G.: Investment of Attention as an Analytic Approach to Cognitive Dimensions, in Collected Papers of the 11th Annu. Workshop Psychology of Programming Interest Group (PPIG-11), T. R. G. Green, R. H. Abdullah & P. Brna, Eds. , 1999, Leeds, UK. pp. 24-35.

[Curtis, 88] Curtis, B., Krasner, H., and Iscoe, N.: A Field Study of the Software Design Process for Large Systems, Commun. ACM, vol. 31, no. 11, pp. 1268–1287, 1988

[Lizcano, 09] Lizcano, D., Soriano, J., Reyes, M. and Hierro, J.J.: A user-centric approach for developing and deploying service front-ends in the future internet of services, Int. J. Web Grid Serv, 2009.

[Soriano, 07] Soriano, J., Lizcano, D., Cañas, M.Á., Reyes, M. and Hierro, J.J.: Fostering innovation in a mashup-oriented enterprise 2.0 collaboration environment, System and Information Science Notes, vol. 1, no. 1, pp. 62-69 and SIWN Conf. Adaptive Business Systems, ICABS2007, Chengdu, China.

[Lizcano, 08] Lizcano, D., Jiménez, M., Soriano, J., Cantera, J. M., Reyes, M and Hierro, J. J.: Leveraging the upcoming internet of services through an open user service front-end framework, Towards a Service-based Internet, Proc. ICSOC/ServiceWave 2008 Conf. Berlin, Germany: Springer Verlag, LNCS, vol. 5377.

[Floyd, 79] Floyd, R. W.: The paradigms of programming. Commun. ACM, vol. 22, no. 8, pp. 455-460, August, 1979.

[Burnett, 01] Burnett, M., Atwood, J., Walpole Djang, R., Reichwein, J., Gottfried, H. and Yang, S.: Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm, J. Funct. Program., vol. 11, no. 2, pp. 155–206, March, 2001.

[Myers, 06] Myers, B. A., Ko, A. J. and Burnett, M. M.: Invited research overview: end-user programming, CHI '06 Extended Abstracts on Human Factors in Computing Systems, ACM, New York, NY, 2006, pp. 75-80.

[Chin, 06] Chin, J.S., Callaghan, V., Clarke, G.: An End-User Programming Paradigm for Pervasive Computing Applications, 2006 ACS/IEEE Int. Conf. Pervasive Services, 2006, pp.325-328.

[Riecken, 94] Riecken, D.: VTP: an end-user programming paradigm based on tool-based language constructs, in IEEE Int. Conf. Sys., Man, and Cybern., vol.3, 2-5, pp.2498-2504.

[Chengchun, 05] Chengchun, S., Haiyan, Y., Lijuan, X., Haozhi, L. and Zhiwei, X.: Towards an End-User Programming Environment for the Grid, in Grid and Cooperative Computing - GCC 2005, LNCS, Berlin/Heidelberg: Springer, 2005.

[Anderson, 06] Anderson, C.: The Long Tail: Why the Future of Business Is Selling Less of More, J. Prod. Innovat. Manag., vol. 24, 2006, pp. 274–276.

[ProgrammableWeb, 11] Programmable Web [Online]. Available: http://www.ProgrammableWeb.com

[Chrome Web Store, 11] Google Chrome Web Store. [Online]. Available: http://www.google.com/chrome/intl/en/more/webstore.html

[Schroth, 07] Schroth, C. and Christ, O.: Brave new web: Emerging design principles and technologies as enablers of a global SOA. Proc. IEEE Int. Conf. on Services Computing, Los Alamitos, CA, 2007, pp. 597–604.

[Schroth, 07b] Schroth, C. and Janner, T.: Web 2.0 and SOA: Converging concepts enabling the internet of services, IT Prof., vol. 9, no. 3, pp. 36–41, 2007.

[Lizcano, 11] Lizcano, D., Alonso, F., Soriano, J. and López, G.: End-User Development Success Factors and their Application to Composite Web Development Environments, Proceedings of The Sixth International Conference on Systems (ICONS 2011), IEEE Computer Society Press, 2011. ICONS 2011, St. Maarten, The Netherlands Antilles.

[Wu, 04] Wu, J.-H., Chen, Y.-C. and Lin, L.-M.: Empirical evaluation of the revised end user computing acceptance model, Comput. Hum. Behav., vol. 23, no. 1, 2004. pp. 162 –174.

[Jones, 03] Jones, S.P., Blackwell, A. and Burnett, M.: A user-centred approach to functions in Excel, Proc.8thACM SIGPLAN Int. Conf. Functional Programming, Sweden, 2003, pp. 165–176.

[Lizcano, 08] Lizcano, D., Soriano, J., Reyes, M. and Hierro, J.J.: EzWeb/FAST: Reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services, Proc. 10th Int. Conf. Information Integration and Web-Based Applications and Services, Austria, 2008, pp. 15–24.

[Lizcano, 09b] Lizcano, D., Fernández, R., Ortega, S. and Soriano, J.: Towards a user-centred composition system for service-based composite applications, Proc. 11th Int. Conf. Information Integration and Web-Based Applications and Services, Malaysia, 2009, pp. 319–328.

[OMG, 06] Object Management Group Inc.: Meta object facility (MOF) core specification., MG Modeling and Metadata Specification, USA, 2006.

[Sobek, 05] Sobek. R.: Official mof specification from omg. Object Management Group, Inc., USA, 2005

[EzWeb, 11] Morfeo EzWeb Project. (2011). [Online]. Available: http://ezweb.morfeo-project.org/lng/en

[FAST, 11] Morfeo FAST FP7 Project. (2011). [Online]. Available: http://fast-fp7project.morfeo-project.org/lng/en

[EzWeb Demo, 11]EzWeb Demo. [Online]. Available: http://demo.ezweb.morfeo-project.org/

[FAST Demo, 11] FAST Demo. [Online]. Available: http://demo.fast.morfeo-project.org/

[EzWeb Catalogue, 11] EzWeb Catalogue Video. [Online]. Available: http://ezweb.tid.es/ezweb/videos/catalogo/catalogo.htm

[FAST Manual1, 11] FAST GVS Manual – Part 1. [Online]. Available: http://www.youtube.com/watch?v=qFt2LBlxkwU

[FAST Manual2, 11] FAST GVS Manual – Part 2. [Online]. Available: http://www.youtube.com/watch?v=dpoRhnF8_1A

[Lehmann, 05] Lehmann, E.L. and Romano, J. P.: Testing Statistical Hypotheses, 3rd ed. New York, Springer, 2005.

[Jessen, 78] Jessen, R.J.: Statistical Survey Techniques, New York, NY: John Wiley and Sons, Inc., 1978.

[Lizcano, 11b] Lizcano D.: Statistical Survey of the EUD model, 2011. [Online]. Available: http://apolo.ls.fi.upm.es/eud

[Tejo-Alonso, 11] Tejo-Alonso, C., Fernández, S., Berrueta, D., Polo, L., Fernández, M. J. and Morlán, V.: eZaragoza, a tourist promotional mashup. [Online]. Available: http://idi.fundacionctic.org/eZaragoza/ezaragoza.pdf)

Appendix I

The set problem is to be solved using:

1) The EUD model through the components and components available in the EzWeb/FAST catalogues (see http://ezweb.tid.es/ezweb/videos/catalogo/catalogo.htm), publishing the final solution (see http://ezweb.tid.es/ezweb/videos/publish/publish.htm) and finally sharing this solution with other end users (see http://ezweb.tid.es/ezweb/videos/share/share.htm)
2) Traditional programming paradigms with which the user is acquainted.

Problem Statement:

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 programming illiterate, meaning that he has never thought of the possibility of building a solution that meets his needs and improves task performance.

This problem requires the use of six resources and/or services.

Appendix II

No.	Item	General Topic	Survey Section
Q7	EzWeb/FAST is a satisfactory means for creating solutions to meet personal needs when it is not feasible to develop a traditional solution due to time and/or budget constraints.	Real expected use of the EUD model by the respondent	Real expected use of the EUD model
Q8	It is rewarding to use tools like EzWeb/FAST and be able to rapidly and simply create mashups.	Personal realization	
Q9	Domain experts, web programmers and service providers should consider the EUD model as a design vision to be taken into account.	EUD's future, real use and success	
Q10	The more people that adopt the EUD model the easier it will be to find useful design components and create end-user solutions.	Forecast network externality of EUD	
Q11	The EUD model enormously simplifies the stages of implementation, testing, debugging and any modifications to account for changing requirements of the EUD solution development process.	EUD vs. traditional programming	
Q12	It was complicated to create a solution to the stated problem using EzWeb/FAST.	Personal realization	EUD problem-solving validity
Q13	The design components available in the EUD model do not meet the needs of real-world problems	EUD component abstraction	
Q14	The communication mechanism between the design elements is not suitable for solving the problems that end users are likely to have.	Pre-/postconditions as an EUD composition technique	
Q15	The solution created using EzWeb/FAST can be straightforwardly evaluated in a stepwise manner to check that it is error free and be able to create increasingly complex solutions.	EUD solution testability and mantainability	
Q16	Using EzWeb/FAST, a change in the end-user requirements leads to major rework to tailor the solution to the new problem.	EUD solution testability and mantainability	
Q17	The EzWeb/FAST EUD platform is easy to use even first time round.	EUD usability	Usability
Q18	Most people could learn to use EzWeb/FAST to develop end-user solutions.	EUD validity for programming illiterate users	
Q19	I get the feeling that it is not easy to create real-world solutions using EzWeb/FAST.	EUD usability	
Q20	The development model interface and support built into EzWeb/FAST are too complex for end users to be able to create solutions.	EUD's future, real use and success	
Q21	Users need a lot of additional training before they will be able to use EzWeb/FAST effectively to develop their own solutions.	Real expected use of the EUD model by the respondent	
Q22	It is easy to link several components in the EzWeb/FAST using pre- and postcondition mechanisms.	Pre-/postconditions as an EUD composition technique	Functionality
Q23	Useful design components are easy to locate thanks to EzWeb/FAST catalogues.	EUD component abstraction	
Q24	It is hard to publish new design components as gadgets for use in composite applications.	Design element publication and catalogue	
Q25	The composite system built did not respond as expected.	Solution conformity to requirements using the EUD model	
Q26	It is hard to create a composite solution to a specific problem using EzWeb/FAST (considering that the catalogue is well enough populated with design components).	Forecast network externality of EUD	
Q27	Which of the following do you think is the most realistic development time ratio considering two development options for a real problem: a) implement a solution from scratch and b) use the EUD model? 1. The EUD model can reduce development time/workload enormously 2. The EUD model can reduce development time/workload appreciably 3. The workload for the EUD model and for programming a solution from scratch is similar. 4. The EUD model takes more development time than traditional programming. 5. The EUD model does not always manage to produce a valid solution to a set problem, even if the catalogue contains the necessary components.	EUD vs. traditional programming	Overall rating
Q28	Using the EUD model and tools like EzWeb/FAST, any user (no matter how much programming knowledge they have) can create their own solution to a particular problem.	Solution conformity to requirements using the EUD model	
Q29	Users need to know how to program to create functional and stable solutions using EzWeb/FAST.	EUD validity for programming illiterate users	
Q30	Developing and tailoring new design components for EUD platforms like EzWeb/FAST will be key occupation of information technology enterprises in the future.	Design element publication and catalogue	

Table II. Survey for measuring RQ1