On Development Practices for End Users

Alessandro Bozzon[1], Marco Brambilla[1],
Muhammad Imran[2], Florian Daniel[2], Fabio Casati[2]

[1] Politecnico di Milano, Dipartimento di Elettronica e Informazione, 20133 Milano, Italy
{bozzon,mbrambil}@elet.polimi.it

[2] University of Trento, Via Sommarive 14, 38123 Povo (TN), Italy
{imran,daniel,casati}@disi.unitn.it

Abstract. The paper discusses some trends in end user programming (EUP) and takes inspiration from the discussions in a panel and in a vertical session on research evaluation within the second Search Computing workshop. We discuss the controversial successes and failures in this field and we elaborate on which facilities could foster adoption of end user programming. We discuss various dimensions of end user programming, including vertical versus horizontal language definition, declarative versus imperative approaches. We exemplify our discussion in the realistic scenario of research evaluation by comparing the Search Computing and ResEval approaches.

Keywords: crowd programming, end user development, mashup, conceptual modeling, declarative programming.

1 Introduction

In recent years, several research projects such as Search Computing, ResEval[1], and FAST[2] spent substantial effort towards empowering end users (sometimes called expert users, to distinguish them from generic, completely unskilled users), with tools and methods for software development. However, the success of **end user development** (EUD) and its potential adoption are still controversial. In this chapter we look at this field from a *search* perspective and we elaborate on which paradigms and ingredients best aid end users in performing development tasks, and most notably formulating complex search queries. We also discuss various dimensions of end user programming, including vertical versus horizontal language definition, declarative versus imperative approaches.

[1] http://reseval.org - ResEval is a Web-based tool for evaluating the research impact of individual researchers and groups by integrating multiple scholarly data sources.

[2] http://fast-fp7project.morfeo-project.org - FAST aims at the development of a new visual programming environment for business processes based on semantic Web services.

The chapter is organized as follows: Section 2 discusses the problem of identifying the developer classes that could be addressed by EUD approaches; Section 3 presents some practices that could foster the adoption of EUD and Section 4 discusses some relevant dimensions of Domain Specific Languages within EUD approaches. Section 5 shows a declarative approach (namely, Search Computing) and an imperative approach (namely, ResEval) at work on a realistic scenario; and Section 6 draws some conclusions.

2 Target User Classes

End user development comprises several alternative approaches, spanning from mashup development, to software configuration, to simple programming tasks and search. These approaches are often antithetic, but sometimes they can be combined together to exploit the respective strength points.

For instance, while users are getting more and more used to *configuring* applications, also thanks to the pervasiveness of mobile and gaming software, *mashup platforms* for the development of simple Web applications are also gaining popularity. Yet, mashups were actually born as a hacking phenomenon, where very expert developers build applications by integrating reusable content and functionality sourced from the Web (for instance, see www.programmableweb.com), and – despite the numerous attempts – mashup development is still for skilled programmers only.

Actually, mashup tools initially targeting end users slowly moved towards the expert user, then to the developer, and finally to the expert developer. In fact, our experience on both model-driven web engineering [13] and mashup development [8] has shown that there are basically only two target users in the real world:

- **Developers**, who want to see the source code and to write imperative code. These users do not trust model-driven approaches, because they feel this can reduce their freedom in application development;
- **Non-developers**, who want to ignore all the technical issues and have simple, possibly visual or parameter-based configuration environments for setting up their applications.

The rest of the stratification of users into expert users, entry-level developers, developer/designer that can be theoretically defined does actually not exist. Recognizing the distinction of only two major user classes, empowering non-developers becomes more focused, but also non-trivial.

3 Enabling Practices and Techniques

Enabling end users to develop own applications or compose simple mashups or queries means simplifying current development practices. A variety of options may help simplifying the user development; we discuss the most important ones in the follow-

ing, in order to use them in the next section to analyze two approaches that partly aim at supporting end users in composing complex queries.

Simple programming models. The first issue is to understand which programming paradigms are best suited for end user programming. The solution to this issue can take inspiration from existing experiences in orchestration and mashup languages which are targeted at process automation and at relatively inexperienced users (although they have not been that successful in reaching out to non-IT experts, as yet). The aim is to find programming abstractions that are simple enough to appeal to domain experts and at the same time complex enough to implement enterprise procedures and Web application logic.

For instance, some mashup approaches heavily rely on connections between components (this is the case of Yahoo! Pipes[3] and IBM Damia [1], for instance), and therefore are inherently imperative; other solutions completely disregard this aspect and only focus on the components and their pre- and post-conditions for automatically matching them, according to a declarative philosophy like the one adopted in choreographies (for instance, see the proposal of the FAST European project [9]).

Domain-specific languages (DSLs). Simple programming models are not enough. Typically, end users simply don't understand what they can do with a given development tool, a problem that is basically due to the fact that the development tools does not speak the language of the user and, hence, programming constructs don't have any meaning to the user. Domain-specific languages aim at adding domain terminology to the programming model, in order to give constructs domain meaning.

In some fields, such as database design, domain-specific languages are a consolidated practice: declarative visual languages like the ER model are well accepted in the field. Other, more imperative approaches, like WebML, address developers that are willing to embrace conceptual modeling. Business people, on the other hand, are well aware of workflow modeling practices and are able to work with formalisms like BPMN, completely ignoring what happens behind the scenes both in terms of technological platform and of transformations applied to get to a running application. Another example in this category is Taverna[4], a workflow management system well known in the biosciences field. A more precise classification of DSLs is provided in Section 4.

Intuitive interaction paradigms. User interfaces of development tools may not be a complex theoretical issue, but acceptance of programming paradigms can be highly influenced by this aspect too. The user interface comprises, for instance, the selection of the right graphical or textual development metaphor so as to provide users with intelligible constructs and instruments. It is worth investigating and abstracting the different kinds of actions and interactions the user can have with a development environment (e.g., selecting a component, writing an instruction, connecting two components), to then identify the best mix of interactions that should be provided to the developer.

---

[3] http://pipes.yahoo.com/pipes/

[4] http://www.taverna.org.uk/

Reuse of development knowledge. Finally, even if a tool speaks the language of the user, it may still happen that the user doesn't speak the language of the tool, meaning that he/she still lacks the necessary basic development knowledge in order to use the tool profitably. Such a problem is typically solved by asking more expert users (e.g., colleagues or developers) for help – if such are available. The challenge is how to reuse or support the reuse of development knowledge from more expert users in an automated fashion inside a tool, e.g., via recommendations of knowledge [12].

Recommendations can be provided based on several kinds of information, including components, program specifications, program execution data, test cases, simulation data, and possibly mockup versions of components and program fragments used for rapid prototyping. Information may or may not be tagged with semantic annotations. When present, the annotations can be used to provide better/more accurate measures of similarity and relevance. In a general sense, the approach we envision is an alternative to design patterns for exploiting the expertise of good developers, thus allowing reuse of significant designs.

Programming, testing, and prototyping experiences of peers or of more experienced developers may support the entire development lifecycle. If knowledge is harvested and summarized from peers (e.g., by analyzing their mashup definitions), this opens the door to what we can call "implicit collaborative programming" or "crowd programming", where users, while going through a software engineering lifecycle for implementing procedures of their own interest, create knowledge that can be shared and leveraged by other domain experts for their own work.

4 Domain-Specific Languages: Assessment Dimensions

We have seen that **Domain-Specific Languages** (DSLs), i.e., design and/or development languages that are designed to address the needs of a specific application domain, are important to provide the end user with familiar concepts, terminology and metaphors. That is, DSLs are particularly useful because they are tailored to the requirements of the domain, both in terms of semantics and expressive power (and thus do not enforce end users to study more comprehensive general-purpose languages) and of notation and syntax (and thus provide appropriate abstractions and primitives based on the domain).

While a broad discussion on DSLs is outside the scope of this paper[5], we wish to highlight a few possible classifications of these languages, which can become handy for EUD. In particular, we describe the dimensions of focus, style and notation.

The **focus** of a DSL can be either vertical or horizontal. *Vertical DSLs* aim at a specific industry or field. Examples of vertical DSLs may include: configuration languages for home automation systems, modeling languages for biological experiments, analysis languages for financial applications, and so on. On the other side, *horizontal DSLs* have a broader applicability and their technical and broad nature allows for

[5] See a thorough discussion on this topic here:
http://lostintentions.com/2009/08/15/a-look-into-domain-specific-languages/

concepts that apply across a large group of applications. Examples of horizontal DSLs include SQL, Flex[6], WebML[7], and many others.

The **style** of a DSL can be either declarative or imperative. *Declarative DSLs* adopt a specification paradigm that expresses the logic of a computation without describing its control flow. In other words, the language defines what the program should accomplish, rather than describing how to accomplishing it. *Imperative DSLs* instead specifically require defining an executable algorithm that states the steps and control flow that needs to be followed to successfully complete a job.

The **notation** of a DSL can be either graphical or textual. The *graphical DSLs* (also known as Domain Specific Modeling Languages, DSML) imply that the outcomes of the development are visual models and the development primitives are graphical items such as blocks, arrows and edges, containers, symbols, and so on. The *textual DSLs* comprise several categories, including XML-based notations, structured text notations, textual configuration files, and so on.

Despite the various experiences in DSL design and application, there is no general assessment on the preferences of the developers for one or the other kind of language depending on the user profile. However, typically languages oriented to the end users tend to be more visual and declarative, while the ones for developers are often textual and imperative.

5 EUD in Practice: Two Examples for Research Evaluation

To exemplify how EUD can be supported in practice, we describe two approaches to a domain-specific problem, namely **research evaluation**. Research evaluation has received a lot of interest in the last years since everybody is producing research artifacts at his best, and in this race everyone wants to lead. Assessing the impact of researchers and publications is highly demanded and important [1]. Yet, the very problem of finding experts or high-profile people in some specific area is still a challenging endeavor: simply imagine you want to assess the *independence of young researchers* or to evaluate the *quality of a supervisor*: different metrics should be defined and used.

There are attempts of *applications* that aim to help assessment problems like the above. Typically, they support citation-based metrics for the evaluation of research impact. Examples are Web of Science[8], Scopus[9], Publish or Perish[10], and similar. However, all the currently available tools lack some key features, such as completeness of data, data cleaning options, or comparison features, which imply that their outputs are not always satisfying and reliable.

In the rest of this section we discuss the imperative approach proposed in ResEval and the declarative approach proposed in Search Computing for addressing the issue,

[6] http://www.adobe.com/products/flex/
[7] http://www.webml.org/
[8] http://scientific.thomson.com/products/wos/
[9] http://www.scopus.com/home.url
[10] http://www.harzing.com/

and, finally, we provide a possible combination of the two for leveraging on the respective strength points.

5.1 Research Evaluation in ResEval

ResEval is a research evaluation platform that is currently being developed at the University of Trento. ResEval is a tool that is based on citation-based indicators like h-index, g-index, noise ratio, and citation count. ResEval provides self-citation analysis, the possibility to find top co-authors, and top citers of a researcher.

ResEval is mainly based on *citation analysis*, which is today's de-facto standard practice. Despite its widespread use, citation analysis does however not come without controversy [6]. Everyone has his own, sometimes very *subjective logic* for assessing research impact. Also ResEval comes short if we want to assist sophisticated, user-defined assessment metrics, since – as in all the other tools – the supported features are pre-defined and custom metrics cannot be expressed. In order to support users in defining their own research evaluation logic, we introduce a new ingredient that we think will allow us to further lower the complexity of the mashup process: we propose a *domain-specific information mashup approach* [7] through which users (possibly with no programming skills) can define, execute, and visualize a metric's combination logic and its result.

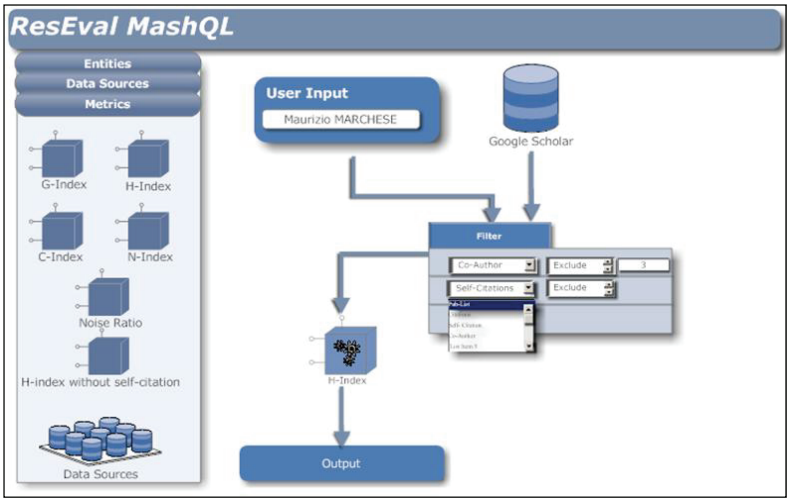

Fig. 1 Mockup of a research evaluation mashup in ResEval

Specifically, ResEval is currently being extended with a ***mashup platform*** (an adaptation and extension of the mashArt platform [8]), allowing users to combine both UI components (widgets that can show charts and trends) and information from a variety of (Web) sources. ResEval features a ***domain-specific language*** that constrains the concepts and the functionality of the platform for the sake of ease of use [10], in order to have a mashup platform that can be used by non-programmers. The focus of the proposed DSL is ***vertical*** (research evaluation), while its style is ***impera-***

tive, and its notation is ***graphical*** so as to fine-tune the ***development metaphors*** to the domain. ***Reuse*** of development knowledge mainly comes in the form of ready components that can be composed into value-adding manners, while keeping large part of the complexity inside the components.

For instance, **Error! Reference source not found.** shows a simple mashup example where data sources, operators, filters, and UI components are used to compute a metric. Specifically, the mashup fetches all data from Google Scholar (other sources like DBLP and Microsoft Academic and unions thereof are also supported), filters out only those publications by a given author, and applies a set of filter conditions (e.g., we exclude self-citations). Then we compute the conventional h-index metric over to so cleaned list of publications and render the result in a UI component so that the user can inspect the output of the computation.

5.2 Research Evaluation with Search Computing

Search Computing complies with all the EUD practices described in Section 3. In particular, it provides ***simple programming models*** (based on visual registration of search services and configuration of queries expressed on such services) defined on a set of domain-specific languages that cover service registration, query design, and query plan refinement (expressed using the Panta Rhei notation).

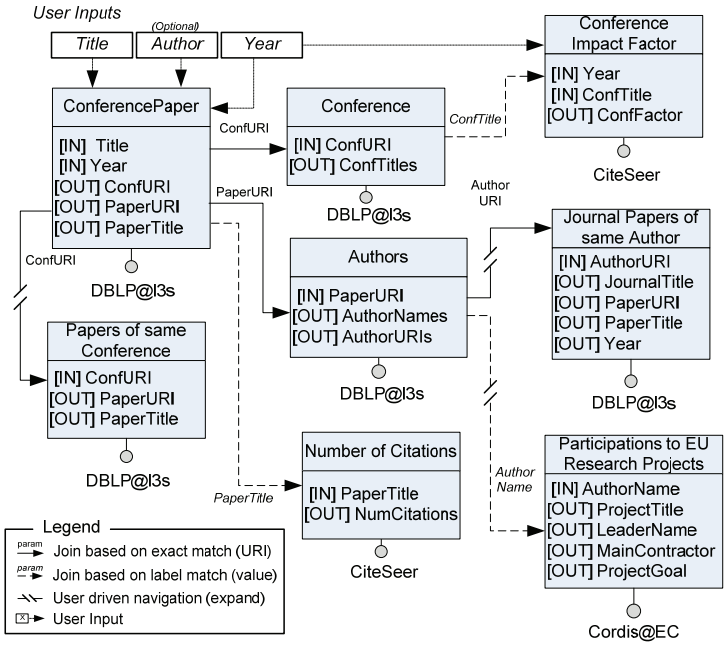

Fig. 2 A liquid query template for a research products and venues search application.

A set of ***graphical design tools*** has been devised to support developers in their work. These tools also comprise support to ***design reuse*** (in terms of registered services and queries), together with additional facilities for design validation (e.g., checking the correctness of the query against the properties of the registered services). At the moment no recommendation features are provided, although some basic ones are scheduled as future work. Generally speaking, Search Computing provides a set of ***horizontal*** DSLs, being focused on search applications but applicable to any industrial field. Most of them are ***declarative***, except for Panta Rhei, which specifies the query plans as an executable orchestration of search services and therefore is ***imperative***. The notations are graphical, except for SeCoQL and the Liquid Query configuration language [3], which are textual notations.

Being a general-purpose approach (i.e., a horizontal DSL), search computing can be easily applied to the research productivity field. To provide a better understanding of our approach, consider a scenario in which a research evaluation application is built at design time [4] and then consumed at runtime by a user through the Liquid Query interface [1].

For instance, we assume the final application expects the user to search for a conference paper and the systems to provide him the list of matching papers, the number of citations for each paper, the authors, the and the details of conference where it was presented, including its impact factor. Subsequently, the user can decide to extend the results by navigating toward other papers published in the same conference, the journal papers of each author, and the European research projects the authors have been responsible for. The declarative specification of such a query template can be displayed visually, as shown in **Error! Reference source not found.**, while the results are shown in the Atom View interface [3] represented in Fig. 3[11].

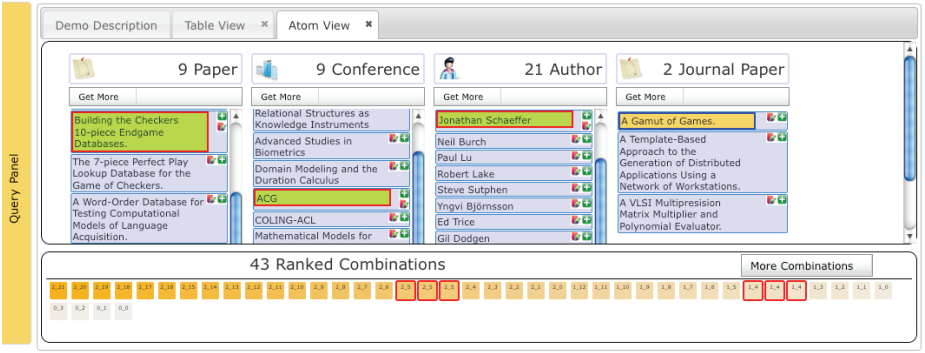

Fig. 3 Structure of the query upon research products and venues.

5.3 On the Combined Potentials of ResEval and Search Computing

The previous two sections show that advanced research evaluation scenarios can be approached from at least two different perspectives. ResEval proposes an *imperative* paradigm for the specification of custom *data processing logic*; it is particularly

[11] See the prototype and a video at: http://www.search-computing.org/demo/ui.

strong in the *flexibility* with which evaluation logic can be expressed. Search Computing proposes a *declarative* paradigm for the specification of *data integration logic*; it is particularly strong for its *ease of use*. Although the two instruments approach the same problem from a different perspective, it is important to note that, rather than representing alternatives, the two instruments complement each other and that, hence, an integration of the two may be beneficial to both.

In particular, Search Computing allows for easy data integration, by natively providing join mechanisms on heterogeneous sources, and efficient data retrieval by means of parallel and asynchronous invocation of services, with non-blocking behavior with respect to the overall query plan.

On the other side, ResEval can contribute pre-defined complex ranking indexes for the specific domain of research evaluation, and composition flexibility thanks to the mashup-based development.

6 Conclusions

This chapter discussed the problem of end user development and investigated the most important enabling features and classification dimensions. The Search Computing and ResEval approaches have been considered as representative examples. While the future of EUD is still uncertain, we strongly believe that combinations of declarative and imperative approaches can bring value to end users, who can be empowered with basic development capabilities, at least in limited domain scenarios.

References

1. M. Altinel, P. Brown, S. Cline, R. Kartha, E. Louie, V. Markl, L. Mau, Y.-H. Ng, D. Simmen, A. Singh. Damia – A Data Mashup Fabric for Intranet Applications. In *Proceedings of VLDB '07*, Vienna, Austria.
2. A. Bozzon, M. Brambilla, S. Ceri, P. Fraternali. Liquid query: multi-domain exploratory search on the Web. In *Proceedings of WWW '10*, ACM, New York, NY, USA, pp. 161-170.
3. A. Bozzon, M. Brambilla, S. Ceri, P. Fraternali. Exploring the Web with Search Computing. In *New Trends on Search Computing*, Stefano Ceri and Marco Brambilla (eds.), Springer LNCS, Vol. 6585, March 2011.
4. M. Brambilla, L. Tettamanti. Search computing processes and tools. In *New Trends on Search Computing*. Stefano Ceri and Marco Brambilla (eds.), Springer LNCS, Vol. 6585, March 2011.
5. Informatics Europe Report. 2008. Research evaluation for computer science. Available at *http://www.informatics-europe.org/docs/research_evaluation.pdf*
6. A.J. Chapman. Assessing research: citation count shortcomings. *The Psychologist*, 1989, pp. 336-44.
7. M. Imran, F. Daniel, F. Casati, M. Marchese. ResEval: A Mashup Platform for Research Evaluation. In *Proceedings of ECSS'10*, 2010, Prague, Czech Republic.
8. F. Daniel, F. Casati, B. Benatallah, M.-C. Shan. Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In *Proceedings of ER'09*, pp. 428-443.
9. Hoyer, V., Janner, T., Delchev, I., Fuchsloch, A., López, J., Ortega, S., Fernández, R., Möller, K.H., Rivera, I., Reyes, M., Fradinho, M. (2009). The FAST Platform: An Open

and Semantically-enriched Platform for Designing Multi-Channel and Enterprise-Class Gadgets. In P*roceedings of ICSOC 2009*, Stockholm, Sweden.
10. S. Soi, M. Baez. Domain-specific Mashups: from all to all you need. In *Current Trends in Web Engineering – ICWE 2010 Workshop Proceedings*, July 2010, Springer, pp. 384-395.
11. M. Baez, A. Birukou, F. Casati, M. Marchese. Addressing Information Overload in the Scientific Community. *IEEE Internet Computing*, vol. 99, no. PrePrints, 2010.
12. S. Roy Chowdhury, C. Rodríguez, F. Daniel, F. Casati. Wisdom-Aware Computing: On the Interactive Recommendation of Composition Knowledge. *Proceedings of WESOA'10*, December 2010, Springer.
13. S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, M. Matera. Designing Data-Intensive Web Applications. Morgan Kaufmann, 2002.