An Effective end-user development approach through domain-specific mashups for Research Impact Evaluation. (Executive Summary)

Muhammad Imran, Qatar Computing Research Institute
-Executive Summary-

AN EFFECTIVE END-USER DEVELOPMENT APPROACH THROUGH DOMAIN-SPECIFIC MASHUPS FOR RESEARCH IMPACT EVALUATION

Muhammad Imran

Advisor:
Prof. Maurizio Marchese
Università degli Studi di Trento

Co-Advisor:
Prof. Fabio Casati
Università degli Studi di Trento

March 2013
Abstract

Over the last decade, there has been growing interest in the assessment of the performance of researchers, research groups, universities and even countries. The assessment of productivity is an instrument to select and promote personnel, assign research grants and measure the results of research projects. One particular assessment approach is bibliometrics i.e., the quantitative analysis of scientific publications through citation and content analysis. However, there is little consensus today on how research evaluation should be performed, and it is commonly acknowledged that the quantitative metrics available today are largely unsatisfactory. The process is very often highly subjective, and there are no universally accepted criteria.

A number of different scientific data sources available on the Web (e.g., DBLP, Microsoft Academic Search, Google Scholar) that are used for such analysis purposes. Taking data from these diverse sources, performing the analysis and visualizing results in different ways is not a trivial and straightforward task. Moreover, the data taken from these sources cannot be used as it is due to the problem of name disambiguation, where many researchers share identical names or an author different name variations appear in the data. We believe that the personalization of the evaluation processes is a key element for the appropriate use and practical success of these research impact evaluation tasks. Moreover, people involved in such evaluation processes are not always IT experts and hence not capable to crawl data sources, merge them and compute the needed evaluation procedures.

The recent emergence of mashup tools has refueled research on end-user development, i.e., on enabling end-users without programming skills to produce their own applications. Yet, similar to what happened with analogous promises in web service composition and business process management, research has mostly focused on technology and, as a consequence, has failed its objective. Plain technology (e.g., SOAP/WSDL web services) or simple modeling languages (e.g., Yahoo! Pipes) do not convey enough meaning to non-programmers. We believe that the heart of the problem is that it is impractical to design tools that are generic enough to cover a wide range of application domains, powerful enough to enable the specification of non-trivial logic, and simple enough to be actually accessible to non-programmers. At some point, we need to give up something. In our view, this something is generality since reducing expressive power would mean supporting only the development of toy applications, which is useless, while simplicity is our major aim.

This thesis presents a novel approach for an effective end-user development, specifically for non-programmers. That is, we introduce a domain-specific approach to mashups that “speaks the language of users”, i.e., that is aware of the terminology, concepts, rules, and conventions (the domain) the user is comfortable with. We show what developing a domain-specific mashup platform means, which role the mashup meta-model and the domain model play and how these can be merged into a domain-specific mashup meta-model. We illustrate the approach by implementing a generic mashup platform, whose capabilities are based on our proposed mashup meta-model.
# Contents

1 **Introduction**  
  1.1 Problems, Challenges and Objectives .................................. 2  
  1.2 Solution Overview and Contributions ............................... 3  
    1.2.1 Overview ........................................................................... 3  
    1.2.2 Contributions ................................................................. 4  

2 **End-user Development & Mashups: State of the Art** .................. 7  
  2.1 Overview .............................................................................. 7  
  2.2 End-user Development .......................................................... 8  
  2.3 Enabling Practices and Techniques ................................. 9  
    2.3.1 Domain-specific Modeling ........................................... 9  
    2.3.2 Domain-specific Languages (DSLs) ............................... 10  
    2.3.3 Intuitive Interaction Paradigms .................................... 10  
  2.4 Mashups from an End-User Development Prospective .......... 10  
    2.4.1 Web 2.0 & Enabling Technologies ................................ 10  
    2.4.2 Tool-Assisted Mashup Development ............................. 12  

3 **End-User Oriented Mashup Platform Development Methodology** 15  
  3.1 Overview .............................................................................. 15  
  3.2 Concepts & Definitions ........................................................ 15  
  3.3 Challenges and problems .................................................... 16  
  3.4 Methodology ....................................................................... 17  
  3.5 The Domain Concept Model ............................................. 19  
  3.6 The Generic Mashup Meta-Model ...................................... 21  
    3.6.1 The mashup meta-model ............................................. 22  
    3.6.2 Operational semantics .............................................. 24  
    3.6.3 Generic mashup syntax ............................................. 24  
  3.7 The Domain-Specific Mashup Meta-Model ....................... 24  
    3.7.1 Domain process model ............................................. 25  
    3.7.2 Domain rules .............................................................. 26  
    3.7.3 Domain syntax .......................................................... 27  
    3.7.4 Domain instances ...................................................... 27
# Domain-Specific Mashup Platform Development

## 4.1 Overview
### 4.2 Components & Compositions Execution Insights
- 4.2.1 Orchestration style
- 4.2.2 Data-passing style
- 4.2.3 Compositions execution

## 4.3 Components Definitions
### 4.3.1 Component Definition Language (CDL)

## 4.4 Mashup Compositions Definitions
### 4.4.1 Mashup Definition Language (MDL)

## 4.5 The Mashup Engine
### 4.5.1 Mashup Engine Architecture
### 4.5.2 The Repository Module
### 4.5.3 Component- and Composition Mapper
### 4.5.4 Component Descriptor and Component
### 4.5.5 Composition
### 4.5.6 Data Mapper
### 4.5.7 Data Processor
### 4.5.8 Configuration Interface

# ResEval Mash: A Domain-Specific Mashup Tool

## 5.1 Overview
### 5.2 Design Principles
- 5.2.1 Intuitive graphical user interface
- 5.2.2 Hidden data mappings
- 5.2.3 Data-intensive processes
- 5.2.4 Platform-specific services
- 5.2.5 Runtime transparency

## 5.3 ResEval Mash Architecture
### 5.3.1 Overview
### 5.3.2 Mashup Engine
### 5.3.3 Composition editor
### 5.3.4 Component Registration Interface
### 5.3.5 Server-Side Services
### 5.3.6 CDM Memory Manager, CDM Module & Shared Memory
### 5.3.7 Local Database and the Web

## 5.4 Intelligent Switching between Data-flow and Control-flow

## 5.5 Components Models and Data Passing Logic

## 5.6 The Domain-Specific Service Ecosystem

# User Studies and Evaluation

## 6.1 Overview
### 6.2 Comparative & Usability Evaluation: User Study-1
- 6.2.1 Task Design
| 6.2.2  | Evaluation Procedure                   | 60 |
| 6.2.3  | Questionnaires                         | 62 |
| 6.2.4  | Results                                | 65 |
| 6.2.5  | Evaluation Analysis & Discussion       | 71 |
| 6.3    | Usability Evaluation: User Study - 2   | 75 |
| 6.3.1  | Evaluation Procedure                   | 76 |
| 6.3.2  | Participants Description               | 78 |
| 6.3.3  | Analysis: Usability Evaluation         | 78 |
| 6.4    | General Analysis & Discussion          | 79 |
| 7      | Conclusions and Future work            | 81 |
| 7.1    | Overview                               | 81 |
| 7.2    | Contributions Summary of the Thesis    | 82 |

**Bibliography** 85
Chapter 1

Introduction

The concepts of *scientometrics* (i.e., the science of measuring and analyzing science) and *informetrics* (i.e., the study of the quantitative aspects of information in any form) [1][2] are increasingly popular. More specifically, among the other fields that informetrics encompasses, the field of *bibliometrics*, which deals with the quantitative analysis of disseminated information of all forms, has received considerable interest over the last few years. The quantitative analysis of scientific and technological information, under bibliometric field, typically use citation and content analysis techniques. The ultimate goal of such an analysis is to determine the impact of a research work that then contributed to productivity and the impact of researchers (i.e., who actually conduct the research work). Bibliometrics has changed out the way the research assessment practices were following, and as it is now bibliometrics methods are widely being used to evaluate research groups, individual research’s, departments, universities and many more.

However, evaluating someone’s research output quality is a notoriously challenging problem which, so far, has no well accepted solution. The field of research is a competitive struggle for a researcher. These researchers throughout their career are evaluated on the basis of their research work, especially the disseminated work, which could be of different forms. For example, to name a few, among traditional quantitative indicators include *journals publications count*, or *top tier conference publications count* etc., and among *citation-based* methods include, *journal impact factor*, *h-index* value, or *g-index* value etc. Often times, the choice of an evaluation criterion depends on the purpose behind that evaluation practice.

Over the last few years, research impact evaluation received a substantial focus as the amount of contribution to science is increasing heavily, and the competition becomes tougher among researchers, and at large extent among research groups, departments, universities as well as research institutions. As the research landscape evolves, assessing the impact of researchers and their disseminated research outputs is in high demand for a variety of reasons, such as the self-assessment of researchers, evaluation of faculties or universities, faculty recruitment and promotion, funding, awards [3] as well as to support the search for attractive content within an ocean of scientific knowledge. An evaluation task, which determines the impact and the productivity of researchers, requires the selection one or more information sources, appropriate evaluation indicators, and an uncontroversial evaluation procedure. To this end, a vast collection of such evaluation indicators, information sources and procedures are becoming available, which make the evaluation exercise more subjective.
1.1 Problems, Challenges and Objectives

Despite the fact that, the researchers must be evaluated on the basis of their research work; however, there is little consensus today on how an evaluation procedure should be designed and performed, and it is commonly acknowledged that the quantitative metrics available today are largely unsatisfactory. Indeed, today people judge research contributions mainly through publication in venues of interest and through citation-based metrics (such as the h-index), which attempt to measure research impact. However, there is little consensus on how citation statistics should be used, and they have well-known flaws. For instance, [4] pointed out shortcomings, biases, and limitations of citation analysis (e.g., name disambiguation [5]). In another work [6], authors criticize the use of journal impact factor for evaluating research.

Furthermore, current metrics are limited to papers as the unit of disseminated scientific knowledge, while today there are many other artifacts that do contribute to the Science, such as blogs, datasets, experiments, or even reviews, but that are not considered in research evaluation. Besides the flaws of current metrics, the fact remains that people have - and we believe will always have - different opinions on which criteria are more effective than others, also depending on the task at hand (that is, the reason why they are conducting the evaluation) [7]. For example, in our department, the evaluation criteria for researchers are defined in a detailed document of 10 pages full of formulas and are mostly based on publications in venues that considered important in the particular community and are normalized following a particular agreed criteria. For instance, other institutions use citation counts normalized by the community to which the authors belong and then grouped by research programs to evaluate each research group, not individuals. Examples are numerous and, much like in the soccer world cup, everybody has an opinion on how it should be done.

Not only individuals may choose different metrics, but also different sources (e.g., Google Scholar vs. Scopus), different normalization criteria (e.g., normalizing the value of metrics with respect to averages in a given community), different ways to measure individual contributions (e.g., dividing metrics by the number of authors), or different ways to compare (e.g., compare a candidate with the group that wants to hire them to determine the autonomy and diversity of the candidate from the group), with different aggregation functions (e.g., aggregated h-index of a scientists co-authors, aggregated citation count, etc.).

We believe that this kind of personalization of the assessment processes (as well as many other personalization of the evaluation process, like for instance, the need of normalizing a traditional metric for a specific community) is a key element for the appropriate use and practical success of the various evaluation tasks. Moreover, people involved in such evaluation processes, most of the time are not IT experts, and not capable of building appropriate software for crawling data sources, automatically parsing relevant information, merging data and computing the required personalized metrics. Therefore, in order to empower the interested end-users, we need to design an appropriate and possibly easy-to-use IT platform, which could make life easier of those domain-experts who do not expert in IT. Indeed, supporting custom metrics for research evaluation is a non-trivial issue and requires addressing interesting research questions like:

- What is the set of key features that may enable a user to express its own evaluation metrics, i.e., what is the expressive power needed to do so? For instance, assessing the independence of a set of young researchers requires fetching all publications by the researchers, cleaning
out papers that have been co-authored by the researchers’ PhD supervisor, computing their h-index metrics, and ranking them according to their h-index.

- How to enable less technical end-users to perform both easy and more complex data integration tasks? We have seen that being able to access an evaluation body (e.g., a set of papers) that is as complete as possible is at least as important as expressing custom metrics over the evaluation body. For example, fetching all publications of the young researchers may imply fetching data from Google Scholar, DBLP, and Scopus as well as fusing the obtained data and cleaning it.

- Which is the best paradigm or formalism that may allow users to model/express their custom evaluation metrics? A metric may, for example, be expressed in text form via a dedicated domain-specific language, or modeled visually by means of suitable graphical modeling constructs, composed with the help of a guided wizard, and so on.

- What type of software support does the computation of custom evaluation metrics need? Depending on the logic needed, the actual computation of a metric may be achieved via generated code, a dedicated evaluation engine, a query engine, or similar.

One of the most important issues that need especial consideration while addressing the problem is the kind of target end-users. We intend non-IT experts (i.e., non-programmers) as our end-users, who will get benefited from our research work. In following we introduce our proposed solution for all the aforementioned problems. It must be well-understood that throughout the different stages of our work, we always refer and give examples from the selected domain to convey understanding whenever needed. However, this does not mean that the proposed solution is only valid for the selected domain. Instead, we aim at to keep separate those aspects that purely based on the chosen domain from those of generic type. In essence, we first aim at proposing a generic approach, and a methodology that then given a set of domain-specific aspects we show how to adapt it for that particular domain.

1.2 Solution Overview and Contributions

1.2.1 Overview

After about two decades of research in workflow management and more or less one decade of web service composition, two research streams whose initial ambitious goal was to enable non-technical users to design processes or compose services with little or no help from developers, we are still in a situation in which these forms of process modeling and execution technologies can only be mastered by specifically trained developers. One of the best examples of this situation is probably the recent standardization of Version 2.0 of the Business Process Modeling Notation (BPMN) [8], which brings together the two worlds of BPM and service composition, but that also has become much more like a programming language and less like a modeling instrument targeted at non-programmers (as the size of the documentation also indicates). As a result, people that are not fully familiar with the modeling notation are reluctant to use it since they know that they will not be able to draw a correct and consistent process model.
While this is a concrete issue in business process modeling and service composition, it is even more so in a relatively new, yet highly-related area: web mashups. The recent emergence of mashup tools has refueled research on end-user development, i.e., on enabling end-users without programming skills to compose their own applications.

**Mashups** are typically simple web applications (most of the times consisting of just one single page) that, rather than being coded from scratch, are developed by integrating and reusing available data, functionalities, or pieces of user interfaces accessible over the Web. For instance, [housingmaps.com](http://www.housingmaps.com) integrates housing offers from Craigslist with a Google map adding value to the two individual applications. Likewise, **Mashup tools**, i.e., online development and runtime environments for mashups, ambitiously aim at enabling non-programmers (regular web users) to develop their own applications, sometimes even *situational* applications developed ad hoc for a specific immediate need [9].

However, we think that doing so is even harder than enabling non-programmers to model an own process or service composition, because developing full applications is simply complex. While the component-based reuse approach is certainly lowering part of the complexity, developing an own application, however, also means dealing with data integration, application logic, and content presentation issues, all aspects the common web user is not even aware of. Yet, similar to what happened in web service composition, the mashup platforms developed so far tend to expose too much functionality and too many technicalities so that they are powerful and flexible but suitable only for programmers. Alternatively, they only allow compositions that are so simple to be of little use for most practical applications.

For example, mashup tools typically come with **SOAP services**, **RSS feeds**, **UI widgets**, and the like. Non-programmers do not understand what they can do with these kinds of compositional elements [10] [11]. We experienced this with mashup tools in our own group, mashArt [12] and MarcoFlow [13], which we believe to be simpler and more usable than many composition tools, but that still failed in being suitable for non-programmers [14].

Yet, being amenable to non-programmers is increasingly important as the opportunity given by the wider and wider range of available online applications and the increased flexibility that is required in both businesses and personal life management raise the need for situational (one-use or short-lifespan) applications that cannot be developed or maintained with the traditional requirement elicitation and software development processes.

We believe that the **heart of the problem** is that it is impractical to design tools that are *generic enough* to cover a wide range of application domains, *powerful enough* to enable the specification of non-trivial logic, and *simple enough* to be actually accessible to non-programmers. At some point, we need to give up something. In our view, this something is generality, since reducing expressive power would mean supporting only the development of toy applications, which is useless, while simplicity is our major aim. Giving up generality in practice means narrowing the focus of a design tool to a well-defined *domain* and tailoring the tool’s development paradigm, models, language, and components to the specific needs of that domain only.

1.2.2 Contributions

This chapter presented an introduction of the reference domain and the problems and challenges faced by the users. In following we summarize contributions of this thesis.
1. First of all, we present the novel idea of **domain-specific mashups** and describe what they are composed of, how they can be developed, how they can be extended for the specificity of any particular application context, and how they can be used by non-programmers to develop complex mashup logics within the boundaries of one domain.

2. We detail and exemplify all **design artifacts** that are necessary to implement a domain-specific mashup tool, in order to provide expert developers with tools they can reuse in their own developments.

3. We show what developing a domain-specific mashup tool means, which role the **mashup meta-model** and the **domain concept model**, the **domain syntax model** play and how these can be merged into a **domain-specific mashup meta-model**.

4. We describe a **methodology** for the development of domain-specific mashup tools, defining the necessary concepts and design artifacts. As we will see, one of the most challenging aspects is to determine what is a domain, how it can be described, and how it can both constrain a mashup tool (to the specific purpose of achieving simplicity of use) and ease development. The methodology targets expert developers, who implement mashup tools.

5. We apply the methodology in the context of a **mashup platform** that supports the development of domain-specific mashup tools. To achieve this, we present a baseline platform, which is then used to develop and tailor a mashup tool to support a domain most scientists are acquainted with, i.e., research evaluation. This mashup platform targets domain experts (i.e., non-programmers).

6. In this thesis, we also present an efficient approach for mashup-based web application, those communicate big data between client and server. The proposed approach prevents heavy data communication using suitable communication-pattern (i.e., among the four proposed patterns) and a server-side cache.

7. To evaluate our work, we performed twofold validations. First, we performed a **usability and comparative evaluation**, which is to understand end-users preference between a generic versus a domain-specific mashup tool and to learn the right balance a mashup tool should offer in terms of complexity, flexibility, and expressiveness. Second, we performed a user studies in order to assess advance usability aspects of the developed platform and the viability of the respective development methodology.

While we focus on mashups, the techniques and lessons learned in the thesis are general in nature and can easily be applied to other domains and to other composition or modeling environments, such as web service composition or business process modeling.
Chapter 2

End-user Development & Mashups: State of the Art

2.1 Overview

By and large, in the current era, most people are familiar with the use of computers, at least with the basic functionalities and user-experience that computers provide. These computer users include engineers, teachers [15], doctors, salesmen, scientists [16], managers, and children [17]. Based on a survey conducted by the U.S. Bureau of Labor and Statistics, Boehm et al. in his paper [18], predicted that in 2005 there would be 55 million such end-users (i.e., computer users using spreadsheets, databases, writing formulas, and queries for their daily work requirements). In another work [19], which was also based on a survey conducted in 2005 by the U.S. Bureau of Labor and Statistics, reported that these end-users population already increased to 80 million. Moreover, in the same work, based on the rate of increase from 1995 to 2005, they also predicted that this number will be 90 million in 2012.

The nature of work that many of these users involved - vary - rapidly on the basis of months or even days. Thus, the requirements for more intuitive, easy-to-use and flexible enabling development environments increased as with the growth of end-users. Despite many efforts, it is still a challenging endeavor for the end-users to develop or modify applications that support and fulfill their goals. As this process requires considerable expertise in programming languages that these users lack. On the other hand, traditional requirement elicitation methods and computer programmers simply cannot anticipate and meet all of these requirements.

End-user development (EUD) is a way to solve this problem. EUD helps to empower less skilled users in such a way that they can easily and effectively be involved in development processes so to develop and tailor applications by their own. More specifically, EUD provides different techniques, methods, and tools that allow users to easily cope with the new requirements within the boundaries of a particular user’s expertise [20]. Over the time, different EUD techniques emerged that target different classes of end-users having different expertises [21], [22], [23], [24], [25].

In this chapter, we present state of the art methods that have been proposed in the field of end-user development and we also present an analysis of the major techniques, methods, and tools used for this purpose. We also discuss major paradigms those considered as a funda-
mental base for EUD. Moreover, this chapter introduces the newly emerging field of Mashups, especially in the context of EUD along with various developments in mashups field that have been proposed. In the end we discuss on how mashups can be better choice for less-skilled users.

2.2 End-user Development

The term end-user typically refers and uses for a user of computer applications. The user in this context considered a non-technical or less skilled and a non-programmer. The intentions of these users are to use the computer applications to fulfill their daily life work requirements. While the term end-user development refers to, when an end-user, who is not an expert on conventional computer programming languages, writes computer programs using either declarative or imperative programming techniques. Thus, end-user development, for these kinds of users (i.e., end-user), provides enabling techniques, method and tools that facilitate them to configure, tailor, modify or write new computer programs. Among various forms of end-user development, to name a few, include use of spreadsheets, writing database queries, configuring software programs, visual programming, use of Wikis etc.

Early efforts in the field of EUD were focused around the concepts like customization, parameterization of software programs and some other on tailoring and writing small scripts [26], [27]. These enabling techniques allow end-users, for example, to write scripts in the form of macros for MS Word using Visual Basic syntax, or to perform complex computations or data processing with the help of spreadsheets (e.g., MS Excel), or configuring a software settings using different parameters (e.g., use of various graphical settings). With the passage of time and in parallel the increase in more complex users’ requirements made some of these technologies (e.g., writing scripts or macros), due to their richer technical usage demands, off-track and out of non-programmers technical expertise domain and others (e.g., use of spreadsheets) become simply useless for performing non-trivial tasks.

However, new ways emerged and among those, for instance, programming by example also known as programming by demonstration, to some extent, reduces the efforts a user needed to learn traditional programming abstractions [21]. In this approach a computer program records the user’s action and after generalizing those set of actions it performs the same actions (not necessarily exactly same) in some other similar situations. With the passage of time, the presence of the Internet, especially with the growth of newly emerged Web 2.0 technologies, made it possible to provide a common platform for everyone to produce and consume resources at any time, in any form and from anywhere. For example, among these resources, open data access, Web Services, Online APIs, feeds (i.e., RSS/ATOM feeds) are the most popular. Although the requirement for more intuitive development environments and design support for end-users clearly emerge from research on end-user development, for example for web services [10], [11], not many tools and frameworks are yet available to satisfy this need. From a conceptual point of view, there are currently two main approaches to enable less skilled users to develop programs, which are simplifying development practices and enabling reusability. That is, in general development can be eased either by simplifying it (e.g., limiting the expressive power

---

1 More details on declarative and imperative techniques will be presented in the next section
of a programming language) or by reusing knowledge (e.g., copying and pasting from existing algorithms).

Among the simplification approaches, the workflow and Business Process Management (BPM) community was one of the first to propose that the abstraction of business processes into tasks and control flows would allow also less skilled users to define their own processes. Yet, according to our opinion, this approach achieved little success and modeling still requires training and knowledge. The advent of the Service-Oriented Architecture (SOA) substituted tasks with services, yet the composition is still a challenging task even for expert developers \cite{11} \cite{10}. The reuse approach is implemented by program libraries, services, or templates (such as generics in Java or process templates in workflows). It provides building blocks that can be composed to achieve a goal, or the entire composition (the algorithm -possibly made generic if templates are used), which may or may not suit a developer’s needs.

In recent years, several research projects such as Search Computing\cite{28}, mashArt \cite{29}, FAST\cite{30} and even our own old tool ResEval \cite{31} spent substantial effort towards empowering end-users (as for some of these tools refer end-users sometimes as expert users, to distinguish them from generic, completely unskilled users), with tools and methods for software development \cite{32}. In the following we look at this field from a different perspective and we elaborate on which paradigms and ingredients best aid end-users in performing development tasks, and most notably formulating complex tasks. We also discuss various dimensions of end-user programming, including vertical versus horizontal language definition, declarative versus imperative approaches.

### 2.3 Enabling Practices and Techniques

Enabling end-users to develop own applications or compose application programs by combing together the different pieces available online in the form of public web services, APIs or data in various forms, requires simplifying current end-user development practices. To this end, a variety of approaches may help simplify the end-user development, as also discussed a few of these approaches in the previous section. However, in this section we discuss in detail the most important ones, in order to use them in the next section to analyze these approaches that partly aim at supporting end-users for composing complex applications.

#### 2.3.1 Domain-specific Modeling.

The idea of focusing on a particular domain and exploiting its specificities to create more effective and simpler development environments is supported by a large number of research works \cite{33} \cite{34} \cite{35} \cite{36}. Mainly these areas are related to Domain Specific Modeling (DSM) and Domain Specific Language (DSL).

In DSM, domain concepts, rules, and semantics are represented by one or more models, which are then translated into executable code. Managing these models can be a complex task that is typically suited only to programmers but that, however, increases users’ productivity.

\begin{footnotesize}
\textsuperscript{2}http://www.search-computing.it/
\textsuperscript{3}http://fast-fp7project.morfeo-project.org
\end{footnotesize}
This is possible thanks to the provision of domain-specific programming instruments that abstract from low-level programming details and powerful code generators that "implement" on behalf of the modeler. Studies using different DSM tools (e.g., the commercial MetaEdit+ tool and academic solution MIC [33]) have shown that developers’ productivity can be increased up to an order of magnitude.

2.3.2 Domain-specific Languages (DSLs)

Simple programming models are not enough. Typically, end-users simply do not 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 do not 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 the DSL context, although we can find solutions targeting end-users (e.g., Excel macros) and medium skilled users (e.g., MatLab), most of the current DSLs target expert developers (e.g., Swashup [37]). Also here the introduction of the "domain” raises the abstraction level, but the typical textual nature of these languages makes them less intuitive and harder to manage and less suitable for end-users compared to visual approaches. A number of benefits and limits of the DSM and DSL approaches are summarized in [36] and [35].

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 [38], 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 [39], a workflow management system well known in the biosciences field.

2.3.3 Intuitive Interaction Paradigms

The 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.

2.4 Mashups from an End-User Development Prospective

2.4.1 Web 2.0 & Enabling Technologies

During the last decade, the advent of Web 2.0 has been drastically and successfully proved as an enabling environment for normal web users to enable them to involve into the creation and consumption of Web resources of various types, like blogs, Wikis, Social Media etc. In
respect to Web 1.0 which was known as "web as information source", web 2.0 is called "web as participation platform". Of the major key features of Web 2.0 from EUD point of view include "rich user experience", "user as a contributor", and "user participation".

Among relevant Web 2.0 technologies, the Service-Oriented Architecture (SOA) field emerged as a paradigm in software development. The emerging visions of an Internet of Services (IoS) and a Web Service Ecosystems [40] [41] supported SOA and have shown much potential in the field. However, the major focus of these technologies remained on the technical level of a service to service based interactions systems [42] and a little on service to user (i.e., non-programmer) communication. Due to the high technical complexity of the relevant standards (e.g., WSDL, SOAP, UDDI, REST), we think that doing so is even harder than enabling non-programmers to model an own process or service composition, because developing full applications is simply complex as it require a lot of programming knowledge to deal with data, application and presentation issues [43].

In parallel to Web 2.0 technologies evolution, the Web mashup [44] phenomenon emerged, which provided easier ways to glue these services and data together [45] and claiming to enable also non-programmers to use and mash pre-built components that provide an abstraction of complex programming concepts. Before investigating further on mashups and to set the context, lets just introduce the terminologies that are mostly used. Typically the term mashup refers to those web applications that, rather than being developed from scratch, are developed using various available data, functionalities or user interfaces over the Web. While, Mashup tools, provide development and runtime environments for the composition and execution of applications (i.e., mashup applications) to non-programmers to enable them to create their own situational applications [46].

Based on the Web 2.0 philosophy, a new type of mashup-based approach emerged, which is known as Enterprise Mashups [47]. As it is more adapted and evolved in large companies where more rapid requirements require employees to be dealt with more sophisticated information technologies. Within an organization the key components of Enterprise Mashups include "resources", "widgets" and "mashups", that deal with data (i.e., actual content), application logic (i.e., implementing actual business logic) and mashup application (i.e., assembling together a collection of widgets) respectively. In [48] author introduces mashup concepts and present a mashup model for syntactically composing mashups. In this model, a mashup is defined as a network of mashlets. These mashlets are the main mashup components and consist of a set of relations, e.g. internal relations, I/O relations and web service relations. They can be GUI-based and can be organized in a hierarchical way, i.e., complex mashlets can contain simpler ones. In this model mashlets are defined by means of rules that state which the input, the output and the possible services calls are. The authors also explain the necessity of allowing the user to query and update the data dynamically in the mashup, as well as to add, update or remove mashlets at run time.

As mashups aim to bring together the benefits of both simplification and component reuse. We believe that, in order to make application development from programmers-centric to end-user centric, we need to achieve simplicity from both ends (i.e., from the technology as well as from end-user ends). While the component-based reuse approach is certainly lowering part of the complexity, developing an own application, however, also means dealing with data integration, application logic, and content presentation issues, all aspects the common web user
is not even aware of. However, in the case of domain-specific mashup environments, as also in our case, we aim to push simplification even further compared to generic mashup platforms by limiting the environment (and, hence, its expressive power) to the needs of a single, well-defined domain only. Reuse is supported in the form of reusable domain activities, which can be mashed up.

### 2.4.2 Tool-Assisted Mashup Development

In this section we present and review a number of representative mashup tools, and evaluate them based on those main aspects we consider fundamental for addressing real-life end-user needs. Of the main assessment aspects, the support for the integration of data, services and user interface is fundamental. This functionality is known as universal integration. Moreover, we also present our analysis based on the requirements we gathered during our domain analysis, those best suited for end-users, which are intuitiveness of UI, modeling constructs, execution paradigm, and data-mappings.

**Yahoo Pipes**[^4] is a well-known mashup tool by Yahoo. It provides a number of built-in components and a visual composition editor that allows to design data processing logics. The Yahoo Pipes composition editor offers a set of components and works in a drag-drop fashion, where a user can drop, connect and configure components. Yahoo pipes is quite an attractive with its composition environment, which allows web users to make data centric compositions. The components follow data-flow based approach, as each component waits until data becomes available at its input port. The data-flow based approach is more intuitive for an end-user as compared to a control-flow approach as long as it stays trivial. As in case of Yahoo pipes, it mainly offers a very technical set of components (i.e., modeling constructs) like loops, regular expression, URL builder, RSS feeds etc. that makes composition task complex for a non-technical users. A non-technical user by no means can understand these components and consequently unable to make compositions. Moreover, these programming-related components, which require the basic expertise of programming concepts, may have multiple input and configuration parameters through which connection between two components take place. In essence, the complex data mappings have to be performed to compose a valid mashup. Moreover, Yahoo pipes does not support UI integration, and support for service integration is still poor and is out of an end-user technical reach.

Likewise, instead of domain-specific, the generic nature of the components that Yahoo Pipes offers are only understandable by programmers. However, we believe that a domain expert (i.e., a non-IT user) is still not able to get fruitful results from this tool. Because, one of the main reasons is that it restricts domain experts by not offering those domain constructs and terminologies they are familiar with. It is almost as generic as only understandable by IT experts as it exposes programming notations.

**Intel Mash Maker**[^19] provides a different kind of mashup approach which mainly focuses on data (i.e., online content) of a user’s interest. Intel mash maker is a Firefox extension that runs on the client-side browser and adds a toolbar to the browser with a set of buttons representing various functionalities. It basically monitors the user’s behavior that checks what information a user visit or is interested in and automatically builds a mashup application that could be of

[^4]: [http://pipes.yahoo.com](http://pipes.yahoo.com)
interest to the user, even when the user was not aiming at building a mashup application. The tool mainly extracts relevant data from Web pages but does not provide any data integration functionality. Moreover, no UI or data presentation features are provided, likewise it does not allow service composition. The proper use of the tool, especially use of the advanced features, requires programming skills that non-technical end-users lack.

**mashArt** Among the academic projects a noticeable example is mashArt \[50\] project. The tool mainly aimed at a universal integration approach for UI components and end-user centric development. Aiming at these objectives, mashArt comes with models and languages able to accommodate all the three types of needed components (i.e., data, services and UIs) and with a simple web-based editor and an integrated lightweight runtime environment (allowing for instantaneous previewing) targeted at non IT-expert skilled web users. Although mashArt achieved universal integration, yet it is not able to effectively target end-users. The tool does not solve the problem of complex data mappings, and also mainly the components that mashArt provides are of generic types.

**JackBe Presto** JackBe Presto is one of the popular commercial products. The Presto suite is constituted by several distinct tools. One constitutes the composition development environment, Presto Wires, which adopts a Pipes-like approach for mashing up data from enterprise internal and external sources. It also allows a portal-like aggregation of UI widgets (so-called mashlets developed through the Presto Mashlet tool) visualizing the output of such mashups on a dashboard. Each mashlet is independent from the others, thus, and the synchronization at presentation level is limited. This enterprise solution focuses on integrating enterprise internal or external data and on visualizing them in the form of widgets. The portal-like approach, in general, provides a satisfying level of usability for end-users. However, universal integration is not actually achieved.

\[^{5}\text{http://www.jackbe.com}^\]
Chapter 3

End-User Oriented Mashup Platform Development Methodology

3.1 Overview

In this chapter, we present our proposed methodology for the development of mashup based tools that can lower the barriers for end-users by providing them a tool that speaks their language. For this reason, throughout this chapter we show how we have developed a mashup platform for our reference domain, in order to illustrate how its development can tackle the challenges systematically mentioned in the previous chapters. The development of the platform has allowed us to conceptualize the necessary tasks and ingredients and to structure them into a methodology for the development of domain-specific mashup platforms. The methodology encodes a top-down approach, which starts from the analysis of the target domain and ends with the implementation of the specifically tailored mashup platform. In the next section, we first start from the essential concepts and definitions which are required to be defined before we proceed to the domain analysis step.

3.2 Concepts & Definitions

Before going into the details, we introduce the necessary concepts. First of all, leveraging from the interpretation of web mashups [44]:

**Definition** A web mashup (or mashup) is a web application that integrates data, application logic, and/or user interfaces (UIs) sourced from the Web. Typically, a mashup integrates and orchestrates two or more elements.

Simplifying an evaluation task by tailoring a mashup tool to the specific domain of research evaluation first of all requires understanding what a domain is. We define a domain and, then, a domain-specific mashup as follows:

**Definition** A domain is a delimited sphere of concepts and processes; domain concepts consist of data and relationships; domain processes operate on domain concepts and are either atomic (activities) or composite (processes integrating multiple activities), defined according to domain rules.
Definition A domain-specific mashup is a mashup that describes a composite domain process that manipulates domain concepts via domain activities and processes following domain rules. It is specified in a domain-specific, graphical modeling notation.

A domain-specific mashup is therefore a web mashup specified with a domain-specific model. The domain defines the “universe” in the context of which we can define domain-specific mashups. It defines the information that is processed by the mashup, both conceptually and in terms of concrete data types (e.g., XML schemas). It defines the classes of components that can be part of the process and how they can be combined, as well as a notation that carries meaning in the domain (such as specific graphical symbols for components of different classes).

As we will see later in detail, every mashup can only use components that conform to the domain process model and that exchange data which belongs to the conceptual model. This means that each component can send or receive data based on the entities or relationships of the conceptual model. Finally, the domain defines rules that represent invariants to be met by each mashups. It has a static part, which describes the concepts that are proper of the domain, and a dynamic part, which describes the modifications the concepts may be subject to. For instance, in our reference scenario, concepts include publications, researchers, metrics, etc. The process models define classes of components such as data extraction from digital libraries, metric computation, or filtering and aggregation components. A domain rule could, for instance, disallow the use of a specific information source for the computation of a given metric. These domain restrictions and the exposed domain concepts at the mashup modeling level is what enables simplification of the language and its usage.

Generic mashup tools are neither aware of these concepts, nor of these operations. Given Definition 3.2 we can therefore say that our reference scenarios ask for a mashup that is specific to the domain of research evaluation, i.e., it asks for a domain-specific mashup. So following this we can define a domain-specific mashup tool as:

Definition A domain-specific mashup tool (DMT) is a development and execution environment that enables domain experts, i.e., the actors operating in the domain, to develop and execute domain-specific mashups via a syntax that exposes all features of the domain.

A DMT is initially “empty”. It then gets populated with specific components that provide functionality needed to implement mashup behaviors. For example, software developers (not end-users) will define libraries of components for research evaluation, such as components to extract data from Google Scholar, or to compute the h-index, or to group researchers based on their institution, or to visualize results in different ways. Because all components fit in the classes and interact based on a common data model, it becomes easier to combine them and to define mashups, as the DMT knows what can be combined and can guide the user in matching components. The domain model can be arbitrarily extended, though the caveat here is that a domain model that is too rich can become difficult for software developers to follow.

3.3 Challenges and problems

Given these definitions, the problem we solve is that of providing the necessary concepts and a methodology for the development of domain-specific mashup models and DMTs. The problem
is neither simple nor of immediate solution. While domain modeling is a common task in software engineering, its application to the development of mashup platforms is not trivial. For instance, we must precisely understand which domain properties are needed to exhaustively cover all those domain aspects that are necessary to tailor a mashup platform to a specific domain, which property comes into play in which step of the development of the platform, how domain aspects are materialized (e.g., visualized) in the mashup platform, and so on.

The DMT idea is heavily grounded on a rich corpus of research in Human-Computer Interaction (HCI), demonstrating that consideration of user knowledge and prior experience are required to create truly usable and inclusive products, and are key considerations in the performance of usability evaluations [51]. The prior experience of products is important to their usability, and the transfer of previous experience depends upon the nature of prior and subsequent experience of similar tasks [52]. Familiarity of the interface design, its interaction style, or the metaphor it conforms to if it possesses one, are key features for successful and intuitive interaction [53].

More familiar interfaces, or interface features, allow for easier information processing in terms of user capability, and the subsequent human responses can be performed at an automatic and subconscious level. [54] identified that the use of semantics could be an effective tool for enhancing product design and use, particularly for novel users, as they can indicate how the product or interface will behave and how interaction is likely to occur. Similarly, [55] stressed that to be usable and accessible, interfaces need to be easily understood and learned, and in the process, must cause minimal cognitive load. Effective interaction consists of users understanding potential actions, the execution of specific action, and the perception of the effects of that action.

As we cannot exploit the users’ technical expertise, we propose here to exploit their knowledge of the task domain. In other words, we intend to transform mashups from technical tools built around a computing metaphor to true cognitive artifacts [56], capable to operate upon familiar information in order to ”serve a representational function that affect human cognitive performance.”

### 3.4 Methodology

In order to develop a DMT, we have to look into the details of three incremental aspects, i.e., the domain concepts, the domain processes, and the implementation of the DMT. In following we state and define all the ingredients for developing a domain-specific mashup platform. Specifically, developing a domain-specific mashup platform requires:

1. **Definition of a domain concept model (DCM)** to express domain data and relationships. The concepts are the core of each domain. They drive the implementation of the DMT and of its data types and components. It is therefore crucial to precisely delimit the concepts that characterize the domain, in order to instruct the tool how to use them and to develop components that understand them. The specification of domain concepts allows the mashup platform to understand what kind of data objects it must support. This is different from generic mashup platforms, which provide support for generic data formats, not specific objects.
2. Identification of a generic mashup meta-model\(^1\) (MM) that suits the composition needs of the domain and the selected scenarios. A variety of different mashup approaches, i.e., meta-models, have emerged over the last years, e.g., ranging from data mashups, over user interface mashups to process mashups. Before thinking about domain-specific features, it is important to identify a meta-model that is able to accommodate the domain processes to be mashed up.

3. Definition of a domain-specific mashup meta-model. Given a generic MM, the next step is understanding how to inject the domain into it so that all features of the domain can be communicated to the developer. We approach this by specifying and developing:

   (a) A domain process model (PM) that expresses classes of domain activities and, possibly, ready processes. Domain activities and processes represent the dynamic aspect of the domain. They operate on and manipulate the domain concepts. Injecting the domain into the tool means introducing domain-specific extensions into the mashup meta-model, e.g., to take into account the nature of domain activities. The activities that can be composed in order to form new processes indicate which mashup components in terms of data, application logic, and UI components are needed to implement the domain-specific mashups. In the context of mashups, we can map activities and processes to reusable components of the platform.

   (b) A domain rule model that may constrain the use of processes or activities, in order to guarantee the correct use of concepts and components in the tool. We specify domain rules in a domain rule model.

   (c) A domain syntax that provides each concept in the domain-specific mashup meta-model (the union of MM and PM) with its own symbol. The claim here is that just catering for domain-specific activities or processes is not enough, if these are not accompanied with visual metaphors that the domain expert is acquainted with and that visually convey the respective functionalities.

   (d) A set of instances of domain-specific components. This is the step in which the reusable domain-knowledge is encoded, in order to enable domain experts to mash it up into new applications.

4. Implementation of the DMT as a tool whose expressive power is that of the domain-specific mashup meta-model and that is able to host and integrate the domain-specific activities and processes.

   (a) DMT. The DMT must support all features that are specified in both the domain-specific mashup meta-model and the domain concept model. Specifically, the extended mashup meta-model determines the expressive power of the DMT.

   (b) Components. The components instantiate the concepts in the domain-specific meta-model extension and implement the domain activities identified in step 3(d).

\(^1\)We use the term meta-model to describe the constructs (and the relationships among them) that rule the design of mashup models. With the term instance we refer to the actual mashup application that can be operated by the user.
The above steps mostly focus on the *design* of a domain-specific mashup platform. Since domains, however, typically *evolve* over time, in a concrete deployment it might be necessary to periodically update domain models, components, and the platform implementation (that is, iterating over the above design steps), in order to take into account changing requirements or practices. The better the analysis and design of the domain in the first place, the less modifications will be required in the subsequent evolution steps, e.g., limiting evolution to the implementation of new components only.

In the next subsections, we expand each of the above design steps starting from the domain concept model.

### 3.5 The Domain Concept Model

It is important to precisely delimit the concepts that characterize the domain, in order to instruct the tool how to use them and to develop components that understand them. We specify domain knowledge in the form of a *domain concept model*. The domain concept model is constructed by the IT experts via verbal interaction with the domain experts or via behavioral observation of the experts performing their daily activities and performing a suitable task-analysis. The heart of each domain is represented by the information items each expert of that domain knows and understands.

The concept model represents the information experts know, understand, and use in their work. Modeling this kind of information requires understanding the fundamental information items and how they relate to each other, eventually producing a model that represents the knowledge base that is shared among the experts of the domain. In domain-specific mashups, the concept model has three kinds of *stakeholders* (and usages), and understanding this helps us to define how the domain should be represented.

- The first stakeholders are the mashup modelers (domain experts), i.e., the end-users that will develop different mashups from existing components. For them it is important that the concept model is easy to understand, and an entity-relationship diagram (possibly with a description) is a commonly adopted technique to communicate conceptual models.

- The second kind of stakeholders are the developers of components, which are programmers. They need to be aware of the data format in which entities and relationships can be represented, e.g., in terms of XML schemas, in order to implement components that can interoperate with other components of the domain.

- The third stakeholder is the DMT itself, which enforces compliance of data exchanges with the concept model.

Therefore:

**Definition** The *domain concept model (DCM)* describes the *conceptual entities* and the *relationships* among them, which, together, constitute the domain knowledge.

A DCM is an example of data that is used to be as input to or output from a mashup component. Modeling DCM is also an attempt to separate out what doesn’t vary much from
what does in a particular domain. These first-class concept types are constrained by the domain rules.

We express the domain-model as a conventional entity-relationship diagram. It also includes a representation of the entities as XML schemas. For instance, in Figure 3.1 we put only main concepts we could identify in our reference scenarios into a DCM, detailing entities, attributes, and relationships. The core element in the evaluation of scientific production and quality is the publication, which is typically published in the context of a specific venue, e.g., a conference or journal, by a publisher. It is written by one or more researchers belonging to an institution. Increasingly – with the growing importance of the Internet as an information source for research evaluation – also the source (e.g., Scopus, the ACM digital library or Microsoft Academic) from which publications are accessed is gaining importance, as each of them typically provides only a partial view on the scientific production of a researcher and, hence, the choice of the source will affect the evaluation result. The actual evaluation is represented in the model by the metric entity, which can be computed over any of the other entities.

In order to develop a DMT, the ER (Entity-Relationship) model has to be generated through several interactions between the domain expert and the IT expert, who has knowledge of conceptual modeling. The IT expert also generates the XML schemas corresponding to the ER model, which are the actual artifacts processed by the DMT.

In fact, although the ER model is part of the concept model, it is never processed itself by the DMT. It rather serves as a reference for any user of the platform to inform them on the
The Generic Mashup Meta-Model

When discussing the domain concept model we made the implicit choice to start from generic (i.e., domain-independent) models like Entity-Relationship diagrams and XML, as these are well established data modeling and type specification languages amenable to humans and machines. For end-user development of mashups, the choice is less obvious since it is not easy to identify a modeling formalism that is amenable to defining end-user mashups (which is why we endeavor to define a domain-specific mashup approach). If we take existing mashup models and simply inject specific data types in the system, we are not likely to be successful in reducing the complexity level. However, the availability of the DCM makes it possible to derive a different kind of mashup modeling formalism, as discussed next.

To define the type of mashups and, hence, the modeling formalism that is required, it is necessary to model which features (in terms of software capabilities) the mashups should be able to support. Mashups are particular types of web applications. They are component-based, may integrate a variety of services, data sources, and UIs. They may need an own layout for placing components, require control flows or data flows, ask for the synchronization of UIs and the orchestration of services, allow concurrent access or not, and so on. Which exact features a mashup type supports are described by its mashup meta-model.

Besides specifying a type or class of mashups, the mashup meta-model (MM) specifies how to draw the actual mashup (process) models. In the following, we first define a generic mashup meta-model, which may fit a variety of different domains, then we show how to define the domain-specific mashup meta-model, which will allow us to draw domain-specific mashup models.

**Definition** The generic mashup meta-model (MM) specifies a class of mashups and, thereby, the expressive power, i.e., the concepts and composition paradigms, the mashup platform must know in order to support the development of that class of mashups.

The MM therefore implicitly specifies the expressive power of the mashup platform. Identifying the right features of the mashups that fit a given domain is therefore crucial. For instance, our research evaluation scenario asks for the capability to integrate data sources (to access publications and researchers via the Web), web services (to compute metrics and perform transformations), and UIs (to render the output of the assessment). We call this capability universal integration. Next, the scenario asks for data processing capabilities that are similar to what we know from Yahoo! Pipes, i.e., data flows. It requires dedicated software components that implement the basic activities in the scenario, e.g., compute the impact of a researcher (the sum of his/her publications weighted by the venue ranking), compute the percentile of the researcher inside the national sample (producing outputs like ”top 10%”), or plot the department ranking in a bar chart. Figure 3.2 depicts our mashup meta-model that supports the universal integration and also enforce various rules that of a domain-specific type or a of generic nature. In following we describe the details of the proposed mashup meta-model.
3.6.1 The mashup meta-model

We start from a very simple MM, both in terms of notation and execution semantics, which enables end-users to model own mashups.

1. As shown in the Figure 3.2, a **mashup** $m = \langle C, P, R, VP, L \rangle$, defined according to the meta-model MM, consists of a set of **components** $C$, a set of **connectors** (i.e., data pipes) $P$, a set of rules $R$, a set of view ports $VP$ that can host and render components with own UI, and a layout $L$ that specifies the graphical arrangement of components.

A mashup compiles with a set of rules. These rules can be of various types, for example, **data type inclusion**: validates when a new data type introduced to the mashup; **data type dependency** confirms inheritance of those data types that are already exist; **component compatibility** validates components compatibility upon connecting two components. Sometimes two components seem compatible to each other even then their ordering (i.e., the position of a component in a mashup) could make problems. So **component order** checks for right ordering. Finally, there could be many domain-specific rules that a mashup must consider. The detail of domain-specific rules are given later in this chapter.

2. A **component** $c = \langle IPT, OPT, CPT, type, desc \rangle$, where $c \in C$, is like a task that performs some data, application, or UI action.

Components have port through which pipes are connected. Ports can be divided in input (IPT) and output ports (OPT), where input ports carry data into the component, while output ports carry data generated (or handed over) by the component. Each component must have at least either an input or an output port. Both IPTs and OPTs can have parameters of specific data types. The data types include both primitive and domain-specific types once the MM gets extended for a domain.
**Configuration** ports (CPT) are used to configure the components. They are typically used to configure filters (defining the filter conditions) or to define the nature of a query on a data source. The configuration data can be a constant (e.g., a parameter defined by the end-user) or can arrive in a pipe from another component. Conceptually, constant configurations are as if they come from a component feeding a constant value.

A component can be of type *information source*, *information processor*, or *information sink*. Components with no input ports are called *information sources* and work as data source by supplying data to other components. Components with no output ports are called *information sinks*. All UI components are always of information sink type. They do not perform business logic on the consumed data, but to visualize it to users. Components with both input and output ports are called *information processors*. These components take data, process it and produce results.

The type (type) of the components denotes whether they are *UI components*, which display data and can be rendered in the mashup’s layout, or *application components*, which either fetch or process information or a data source components. Mainly the type information is used by the internal’s logic but it could also be used to arrange components for better presentation for end-users.

Components can also have a description desc at an arbitrary level of formalization, whose purpose is to inform the user about the data the components handle and produce.

3. A pipe (i.e., connector) $p \in P$ carries data (e.g., XML/JSON documents) between the ports of two components, implementing a data flow logic. So, $p \in IPT \times (OPT \cup CPT)$.

4. A view port $vp \in VP$ identifies a place holder, e.g., a DIV element or an IFRAME, inside the HTML template that gives the component its graphical identity. Typically, a template has multiple placeholders.

5. Finally, the layout $L$ defines which component with own UI is to be rendered in which view port of the template. Therefore $l \in C \times VP$.

Each mashup following this MM must have at least a source and a sink, and all ports of all components must be attached to a pipe or manually filled with data (the configuration port).

This is all we need to define a mashup and as we will see, this is an executable specification. There is nothing else besides this picture. This is not that far from the complexity of specifying a flowchart, for example. It is very distant from what can be an (executable) BPMN specification or a BPEL process in terms of complexity.

In the model above there are *no variables* and *no data mappings*. This is at the heart of enabling end-user development as this is where much of the complexity resides. It is unrealistic to ask end-users to perform data mapping operations. Because there is a DCM, each component is required to be able to process any document that conforms to the model. This does not mean that a component must process every single XML element. For example, a component that computes the h-index will likely do so for researchers, not for publications, and probably not for publishers (though it is conceivable to have an h-index computed for publishers as well). So the component will "attach" a metric only to the researcher information that flows in. Anything else that flows in is just passed through without alterations. The component description will help
users to understand what the component operates on or generates, and this is why an informal description suffices. What this means is that each component in a domain-specific mashup must be able to implement this pass-through semantics and it must operate on or generate one or more (but not all) elements as specified in the DCM. Therefore, our MM assumes that all components comply to understand the DCM.

Furthermore, in the model there are also no gateways as in BPMN, although it is possible to have dedicated components that, for example, implement an if-then semantics and have two output ports for this purpose. In this case, one of the output ports will be populated with an empty feed. Complex routing semantics are virtually impossible for non-experts to understand (and in many cases for experts as well) and for this reason if they are needed we delegate them to the components which are done by programmers and are understood by end-users in the context of a domain.

3.6.2 Operational semantics

The behavior of the components and the semantics of the MM are as follows:

1. Executions of the mashups are initiated by the user. A user have to explicitly start the execution using some user interface means (e.g., a button click).

2. Components that are ready for execution are identified. A component is ready when all the input and configuration ports are filled with data, that is, they have all necessary data to start processing.

3. All ready components are then executed. They process the data in input ports, consuming the respective data items from the input feed, and generate output on their output ports. The generated output fills the inputs of other components, turning them executable.

4. The execution proceeds by identifying ready components and executing them (i.e., reiterating steps 2 and 3), until there are no components to be executed left. However, during the execution if in case some component requires user interaction (e.g., an input) before it proceed, then the execution stops and starts again after user acts as needed. This means it is possible to interact with the mashup execution during runtime. At this point, all components have been executed, and all the sinks have received and rendered information.

3.6.3 Generic mashup syntax

Developing mashups based on this meta-model, i.e., graphically composing a mashup in a mashup tool, requires defining a syntax for the concepts in the MM. In Figure 3.3 we map the above MM to a basic set of generic graphical symbols and composition rules. In the next section, we show where to configure domain-specific symbols.

3.7 The Domain-Specific Mashup Meta-Model

The mashup meta-model (MM) described in the previous section allows the definition of a class of mashups that can fit in different domains. Thus, it is not yet tailored to a specific domain,
The Domain-Specific Mashup Meta-Model

Despite the relative simplicity, providing users with a DCM-restricted mashup meta-model is still not likely to be sufficient in terms of ease of use. The user will still be faced with a large number of possible components to be placed on a canvas.

The next step is therefore understanding the dynamics of the concepts in the model, that is, the typical classes of processes and activities that are performed by domain experts in the domain, in order to transform or evolve concrete instances of the concepts in the DCM and to arrive at a structuring of components as well as to an intuitive graphical notation. What we obtain from this is a domain-specific mashup meta-model. Each domain-specific meta-model is a specialization of the mashup meta-model along four dimensions: (1) Domain-specific activities and processes, (2) Domain-specific rules, (3) Domain-specific syntax, (4) Domain instances.

The domain-specific meta-model extension extends the MM with domain-specific sub-types of the component entity in the MM. Sub-types allow the injection of classes of domain processes or activities into the MM and, hence, the introduction of domain-specific terminology and syntax.

3.7.1 Domain process model

Definition The domain process model (PM) describes the classes of processes or activities that the domain expert may want to mash up to implement composite, domain-specific processes.

Operatively, the process model is again derived by specializing the generic meta-model based on interactions with domain experts, just like for the domain concept model. This time the topic of the interaction is aimed at defining classes of components, their interactions and notations. In the case of research evaluation, this led to the identification of the following classes of activities, i.e., classes of components:
For simplicity, we discuss only the processes that are necessary to implement the reference scenarios.

1. **Source extraction** activities. They are like queries over digital libraries such as DBLP or Google Scholar. They may have no input port, and have one output port (the extracted data). These components may have one or more configuration ports that specify in essence the "query". For example a source component may take in input a set of researchers and extract publications and citations for every researcher from Google Scholar.

2. **Metric computation** activities, which can take in input institutions, venues, researchers, or publications and attach a metric to them. The corresponding components have at least one input and one output port. For example, a component determines the h-index for researchers, or determines the percentile of a metric based on a distribution.

3. **Aggregation** activities, which define groups of items based on some parameter (e.g., affiliation).

4. **Filtering** activities, which receive an input pipe and return in output a filtering of the input, based on a criterion that arrives in a configuration port. For example we can filter researchers based on the nationality or affiliation or based on the value of a metric.

5. **UI widgets**, corresponding to information sink components that plot or map information on researchers, venues, publications, and related metrics.

### 3.7.2 Domain rules

As a domain comprises of domain concepts, activities/processes and rules (i.e., constraints, restrictions on concepts and activities) to prescribe and/or restrict the way in which domain experts use domain activities and processes to achieve their goals. These domain rules can be defined in a way like integrity constraints e.g., from the cardinalities between concepts in a domain concept model. However domain rules not only cover integrity constraints but usually also allow or restrict domain behaviors (i.e., domain activities/processes). For example, a rule could be that DBLP cannot be used for computing H-index metrics and thus can be instantiated as a *Component compatibility (mashup)* rule disallowing the usage of these two components in the same mashup composition). The rule enforcement in the DMT provides assistance and guidance to the domain-experts improving usability, composition correctness and development errors reduction.

A well know way to define these rules is through ECA structure (event, condition, activity) which means: if the *event* occurs and *conditions* are met, then execute the *activity* [57]. Figure 3.4 depicts the extension of MM along with rule model. Domain rules can be classified in many different ways. When analyzing rules in the context of domain processes, following are the two rule types we identify: *Activity rules*: domain rules related to the a particular activity or subset of an activity. *Integrity rules*: are related to the domain objects and their relationships, for example, the value of the H-index metric cannot be negative.
3.7.3 Domain syntax

A possible domain-specific syntax for the classes in the PM (derived from the generic syntax presented in Figure 3.3) is shown in Figure 3.5. Its semantics is the one described by the MM in Section 3.6. In practice, defining a PM that fully represents a domain requires considering multiple scenarios for a given domain, aiming at covering all possible classes of processes in the domain.

3.7.4 Domain instances

Domain instances are fully functional domain-specific components that are ready to be used in mashup compositions. These domain-specific components with domain syntax (i.e., domain symbols) implement domain activities and processes, consuming and producing domain-specific concepts at input and output ports. Figure 3.6 actually exemplifies the use of instances of domain-specific components. For example, the Microsoft Academic Publications component is an instance of source extraction activity with a configuration port (SetResearchers) that allows the setup of the researchers for which publications are to be loaded from Microsoft Aca-
The component’s symbol is an instantiation of the parametric source component type in Figure 3.5 without static query. Similarly, the Italian Researchers (source extraction activity), the Venue Ranking (source extraction activity), the Impact (metric computation activity), the Impact Percentiles (metric computation activity), and the Bar Chart (UI widget) components.

In summary, what we do is limiting the flexibility of a generic mashup tool to a specific class of mashups, gaining however in intuitiveness, due to the strong focus on the specific needs and issues of the target domain. Given the models introduced so far, we can therefore refine our definition of DMT given earlier as follows:

**Definition** A *domain-specific mashup tool (DMT)* is a development and execution environment that (i) implements a domain-specific mashup meta-model, (ii) exposes a domain-specific modeling syntax, and (iii) includes an extensible set of domain-specific component instances.

Once the domain models are ready, the IT expert can then customize a mashup platform that meets the requirements that emerge from the domain model. The DMT will therefore expose not only a concept model, but also a process model that specializes MM and that presents to the user a set of components grouped in a domain-meaningful way and with a graphical appearance that makes sense for the domain. Doing so implies, first, understanding which type of mashups the platform should support and, then, tailoring the mashup platform to the specific domain. To this end, the next chapter presents the implementation related details that implements a generic mashup platform following the mashup-meta model.

Figure 3.6: An example of the use of instances of domain-specific components
Chapter 4

Domain-Specific Mashup Platform Development

4.1 Overview

In the previous chapter we have presented the methodology for the development of a domain-specific mashup platform. The methodology clearly separate domain-independent concerns (i.e., in the form of mashup meta-model) from what of domain-specific ones (i.e., domain-specific mashup meta-model and its extensions). A mashup platform whose development follows the defined methodology initially stays empty in terms of domain-specific knowledge (i.e., terminologies, concepts, rules, activities etc.) that is then injected tailoring it to a domain-specific mashup tool. That is how we enable generic platform to be tailored for a specific domain. In this chapter we present how we developed the domain-specific platform that will be then tailored for our reference domain (that is presented in the next chapter). However, in this chapter we specifically focus on the technical concerns and technological design decisions that we have taken.

The mashup meta-model proposed in the previous chapter explains well the capabilities a mashup platform can offer whose implementation follows the specified model. For example, the provision of universal integration is achieved through the support of components those can be of type service, UI or data, that is, the platform from its architectural design supports this capability. The easy-of-use feature, that is to effectively enable non-technical users in development, is achieved through via no-complex mapping concept. Intuitiveness is achieved via introducing domain-specific syntax for composition constructs. These are all the fundamental characteristics that help our platform to provide an effective end-user development environment. In this chapter we not only present steps in the development of a mashup platform that should on one side reduce the complexity of creating mashups for non-technical users but on the other side support developers in the process of developing new components. Primarily focusing on domain-specific mashups greatly help us achieve these objectives.

To this end, we first present our baseline mashup engine that is comprised of various modules, which are explained later in this chapter. We aim for a very lightweight yet powerful mashup engine that can easily run in web browsers, that is, at the client-side with no need to download any extra software. For this purpose, the engine is implemented using JavaScript language.
and runs at client-side in a web browser. The choice of using JavaScript language over other languages is highly motivated by the fact that most Web 2.0 Ajax based web applications whose major goals are to offer fast interactive yet attractive designs and user interfaces, use client-side languages like JavaScript.

### 4.2 Components & Compositions Execution Insights

Before describing the technical details of the mashup platform, we first present a few design aspects that must be considered in order to get maximum benefit of our mashup meta-model. Just to clarify a few terminologies, we refer a "composition" to a set of components connected together to make a mashup. Components are fundamental units contain presentation or application logic and perform certain operations. They usually take some input and generate some output (i.e., the result of their operation). Several components can be connected, so that the output of one component serves as input for another component, which forms a mashup composition (i.e., also referred as a mashup or simply a composition). In the next sections, both "mashup" and "composition" terms interchangeably used, though representing the same meaning.

#### 4.2.1 Orchestration style

Of the many important aspects, orchestration is a key aspect to be considered prior to the development of a mashup platform. Generally, the orchestration, where multiple complex computing units involved, manages their coordination while their execution. Similarly, in our case, it specifies how to synchronize the execution of components in a composition making better coordination among them. Mostly, there are three prominent approaches that have been adopted [58]:

**Flow-based** approach maintains orchestration as sequencing of components that is also a kind of flowchart based approach, where multiple units (e.g., components) connect together whose execution happens according to a defined sequence.

**Event-based** approach offers a publish-subscribe way, where pub/sub models maintain synchronous behavior among components. When a component behaves like publisher sends messages to a queue which then consumed by all those components (subscribers) who are interested.

**Layout-based** approach place components in a composition into a common layout that then each component’s behavior is specified individually by accounting for the other components’ reactions to user interactions.

We use a combination of flow-based and event-based approaches. That is, generally components execution takes place following flow-based style, however, components’ operations can subscribe to the various data buses that then received required data when an event triggered.

#### 4.2.2 Data-passing style

The choice of data-passing approach is another pivotal aspect, which describes the behavior through which data flows among various components. This important property alone can be used to effectively distinguish among various mashup tools, especially when the target end-user
Components Definitions 31

belong to a non-technical user class. Mainly two approaches have been followed in the past, which are data-flow and blackboard-based [58].

According to the data-flow approach, actual data flow from a component to another component. A component starts its execution upon receiving data its waiting for, and once the execution completes it sends the data to the next component in the flow. The data-flow approach considered more intuitive for non-technical users as it follows the philosophy of a natural workflow in daily life work. On the other hand, according to the blackboard based approach data is written to variables, which serve as the source and target of an operation invocation on components, much like in programming languages.

In our case, we follow the data-flow based approach. For example, a data source component produces data (after fetching from a database/web service etc.) and hand over it to the next connected component that then consumes it for further processing. To convey the execution status of a component that would also reflect a composition execution status, we aim to present to end-users the execution status of individual components. Another aspect related to the data-flow approach, which we describe later in this chapter, is that sometimes on the background instead of passing the actual data we pass control data. This scenario gets activated for components whose implementation is of a web-service type. So far we have presented the different types of components, while we will present how these components can be implemented (i.e., as a web service, or as a client side implementation) later.

4.2.3 Compositions execution

A mashup composition, which comprises of several connected components, executes to achieve its goals. The execution of a composition means, running its components in an order that is defined by the end-user. The general execution semantics of a composition/mashup is described in section 3.6.2. However, in this section we look at whether the execution follows an instance-based or a continuous approach [58]. An instance-based model is the traditional service composition model, in which a certain kind of message’s arrival activates a new instance of the composition, and the system executes the instance within the same main thread and context (much like a program run). On the other hand, the continuous model has one instance per component in a composition model. Each component works as a thread, processing the input data feed and transforming or filtering it to generate the output. The strategy we follow is continuous model based, that is, to allow various components to execute using their own threads and also communicate between them when required.

4.3 Components Definitions

Given the above insights, now we first detail on Component Definition Language (CDL), which is build based on components capabilities described in the mashup meta-model. CDL represents just the technical version of what a component is defined by the mashup meta-model. As components are the main building blocks of a mashup, they consume data, perform certain actions/manipulation and produce results. From a software development point of view, a component can be seen as a function or method. It can take one or more input and produce one or more outputs. The input might come from another component or from an external
service or from component’s own UI. Furthermore, a component might require direct user interaction and provide a corresponding UI (i.e., configuration UI). Considering this idea, we show in Figure 4.1 the component definition language model and in Figure 4.2 depicts the component communication mechanism. In following we elaborate the details of both aspects.

### 4.3.1 Component Definition Language (CDL)

From a technical point of view CDL is comprised of the following elements to build a component:

- **Operations**: A component exposes a set of actions that it can perform by means of operations. Operation can be seen as the input configuration ports (i.e., IPT’s) as defined in the mashup meta-model. Operations are invoked through events (described below) and can accept one or more parameters as input and can produce one or more output parameters. Each parameter can have a certain data type. The generic model does not constrain the types, although the mashup tool can restrict possible types that is from a domain concept model (DCM) in the form of an XSD. To complete its computation, a component might need to have operations to be called in a certain order. Therefore, operations can be dependent on each other. Ideally, an operation should expect only one input, to make its purpose more intelligible to the composition designer. This does not necessarily restrain the capabilities of components: A multi-input operation can be split up into several dependent operations.
Components Definitions

- **Events**: Events are the way to propagate results of a component’s action to other components. Events implement the output ports (i.e., OPT) defined in the mashup meta-model. They are either generated programmatically, for example after an operation is completed, or through the user interacting with a component’s UI. Like the input for operations, the output contained in the event data should conform to one or more data types. Creating a composition mainly consists of connecting events with operations which accept the same data types (i.e., domain concepts).

These two concepts, operations and events, are general enough to cover any kind of interaction between components as they are essentially a mixture of the Observer pattern [59] and the more general Publish/Subscribe pattern [60]. These are common patterns used in Model View Controller (MVC) architectures, providing a way to decouple different parts of an application and make them easily exchangeable. From this point of view, applying such a concept seems to be a logical step: the components are the different parts of a composition (the application) and they need to be highly exchangeable due to the dynamic nature of mashups.

- **Requests**: In a composition components send data to each other through operations events connections, but this is not the only type of communication that happens in a composition, instead, often times components communicate with external services or API to fetch or to process data. In the previous chapter we described that components can be of an information source type that act like data sources or can be of type information processor that implement business logic. For these kinds of components it is common that they call external services to accomplish their task. External interactions can be triggered from an operation or from a component’s UI (i.e., against a user UI interaction). Although these interactions can be seen as internal calls without the relevance to the environment, we think that a formal specification of this interaction can be useful for the platform provider and lead to a more comprehensive specification of the component interface. We call this characteristic a **request**. The model is deliberately kept universal in this regard and does not require a detailed specification of the possible type of a service.
or the request and response formats, though the current implementation expects requests to be executable by means of Ajax. Interpreting the response is in the responsibility of the component implementation.

- **Configurations**: Each component can have a set of configuration parameters (Configuration ports (i.e., CPTs)) according to the mashup meta-model. Users use these configuration parameters to configure components through the component’s UI. These parameters must have a data type that can be a primitive or a platform specified type (e.g., domain type) and each parameter value (i.e., user supplied value) of a component belongs to that specific instance of the component and the composition, hence not shared among other compositions.

- **Meta-data**: Finally, a component can have an arbitrary set of meta-data associated with it, typically in the form of key-value pairs. This can be leveraged by the mashup platform to store necessary, platform specific information. For example, the description and type attribute of a component can be defined using the meta-data feature.

### 4.4 Mashup Compositions Definitions

Given the component definition language, we now present *Mashup Definition Language* (MDL), a technical version of what a mashup is defined by the mashup meta-model. The mashup definition language provides a way in which a mashup composition can be defined in terms of components, connections among them through pipes, their states (i.e., parametric values they hold), their instances information and the layout information. We pursue the same goal for the mashup model as for the component model: A minimal set of characteristics that is necessary to represent a functional composition.

In essence, a mashup composition, which is formed using multiple components, is defined connecting events (i.e., output ports) with the operation (i.e., input ports). An event emits data which passes through a pipe and finally consumed by an operation. Components in a composition may hold a user-defined configuration parameters. A composition contains its layout information, which is then used to render components to their proper layout and position. So basically an MDL that is capable to accommodate the above mentioned snippets of information is suitable for our purpose from a technical point of view.

#### 4.4.1 Mashup Definition Language (MDL)

Technically, based on the MDL a mashup is comprised of three basic things, as described below:

- **Components** Many components form compositions hence play major role in building mashups. A composition can have multiple components connected together. The MDL defines connections among components in terms of source and target components. A source component is the one whose event is connected to another component’s operation that is called a target component in this case. Each component assigned an instance id in a composition along with its full qualified name. Other information like a component’s configuration details also preserved in the MDL.
• **Connections** When two components connect, resultantly form a connection. A connection information in the form of source and target components is maintained by the MDL. That is how the association between events and operations of two components is maintained. That is then this information used by the mashup engine to work as a publish-subscriber approach to hand over data to the target operation emitted by the source event.

• **Meta-Data** Mashup definition language also permits to define arbitrary parameters in the form of meta-data (i.e., specifically key-value pair). These parameters can be used to define some special cases such as mashup composition state, permissions. This also provides a way of extending the definition language with details which are not anticipated yet.

### 4.5 The Mashup Engine

Given the CDL that defines components and the MDL that defines mashup compositions (or mashups), we now pursue for a mashup engine that allows the development of components following CDL, composing mashup compositions and finally running those compositions following MDL. The mashup engine must be able to incorporate the above described aspects like orchestration and data-passing style. We aim for data-flow paradigm as a general approach, which must also be conveyed and understandable by the end-users, and sometimes we use control-signals to decrease the data passing overhead hence to increase the overall performance. However, mashup engine’s decision on when specifically data or control-signal flows, is described in the next chapter there we first introduce necessary concepts for its understanding.

The **Mashup Engine** is a core part of the platform, which manages various modules and all communications that take place among these modules. One of the main objectives, which drive along the development of the mashup engine, was to keep separate platform or environment specific requirements to that of a mashup tool’s specific ones (i.e., domain-specific) and in parallel to provide a consolidated platform that can easily be tailored to a specific domain. That’s the reason, throughout the elaboration of various steps of the mashup engine, which we described in the next sub-sections, we mainly focus on those set of generic aspects whose design and implementation is not dependent but of course inspired of a domain. This allows us to use the engine for other domains with similar characteristics as of our reference domain (i.e., research evaluation).

#### 4.5.1 Mashup Engine Architecture

The mashup engine is designed and developed for the client side technologies (e.g., web browser). JavaScript and the Goolge Closure Library were the languages used for the development. In following, we describe the main modules, their roles and relations to each other. Figure 4.3 depicts an overall architecture of the mashup engine. The modules presented in the architecture are the main building blocks of the engine. Later, we will present how UI of a mashup tool can interact with them to give information and control to users. Some modules only describe an

---

1 [https://code.google.com/closure/library/](https://code.google.com/closure/library/)
A few platform provided input elements, more can be added.

**4.5.2 The Repository Module**

The repository module is the one responsible for performing typical CRUD operations (i.e., create, read, update and delete) for managing components and compositions and other external calls that the engine needs to perform. For example, this is the place which is used to access web services that a developer defines inside a component definition. Moreover, the repository module does not fix the way components and compositions are stored to some persistent storage. Instead, the repository interface is designed for the usage of synchronous and asynchronous storage facilities. For example one can use the HTML 5 local storage API, or a server side storage accessed with Ajax or any other means. As an overall this gives provision to the platform provider to decide on how and where the storage will be done based on his requirements and possibilities.

In addition to that, the repository module also independent of what representation is used to describe components or composition (e.g., as in our case either XML or JSON). Hence, this task
is delegated to component and composition mappers (described below) to transfer whatever representation of components and composition to component Descriptor or Composition which are then understandable by the components and composition classes.

4.5.3 Component- and Composition Mapper

The mashup engine does not restrict on a specific component, and compositions representational format. This characteristic introduced because there are many other formats that can be used like to name a few famous ones include W3C Widgets\(^2\) and OpenAjax Widgets\(^3\) and there might be others to be developed in the future. For this reason we introduced the concept of mapper in the platform that actually maps a particular format to the system’s internal one. In the previous sections we showed the components and compositions representations that the mashup engine followed in which the default mappers work.

So, to not restrict the engine to a specific representation, the mapper’s perform the conversion to a specific representation into our internal model and vice versa. It might not be possible to map any component description to our component model, but the idea leads to a certain degree of independence and leaves room for extensions. This again leaves the choice which representations to use and support on platform provider.

4.5.4 Component Descriptor and Component

As described earlier that a component, which is a basic building block, can be used in several compositions and can also occur more than once in a composition. Therefore, it is necessary to distinguish between the properties shared by all the instances of the component, like the operations and events and those of instance-specific properties, like a composition-specific name, it’s id, configuration settings etc.

The component descriptor and component class are used to keep track of these aspects of a component. Basically, the component descriptor is a software artifact that represents a component’s model definition, which is stored in its CDL document. The information that a component descriptor gets populated after parsing component’s CDL include operations, events, requests and configuration parameter details of a component. An instance of the component class is generated to represent this and component’s instance-specific information like the values of configuration parameters, instance id, and it also includes the basic execution logic needed to run a component’s action.

The user interface of the mashup tool directly interacts with component descriptor and component instances to provide information about them and let the user manipulate them through the configuration interface and the UI of a component.

4.5.5 Composition

Compositions represent user defined workflows that accomplish some tasks. As compositions comprised of multiple components connected together in an order to interact with each other by means of connections made between events and operations. The composition module task is

\(^2\)http://www.w3.org/TR/widgets/  
\(^3\)http://www.openajax.org/member/wiki/OpenAjax Metadata 1.0 Specification Widget Overview
to register those connections and performs the communication between components. Basically, the composition listens to each component’s event and notifies the connected components. It’s also responsible for passing the event’s emitted data to the operation. Finally, in order to inform the users about a possible state of a composition while running, the composition module is used by the UI of a mashup tool to convey such indicators.

4.5.6 Data Mapper

During a composition’s run-time, all components that belong to a composition and are connected with each other communicate by means of sending and receiving data among them. Basically, the actual communication takes place between an event and an operation. Event emits data and operation consumes it. The data mappers are responsible for the conversion of the data received from the event so that the data is understandable by the operation. Data mappers would not worthwhile for components those understand and built based on and for a specific domain, for example, in our case many components understand our reference domain DCM (e.g., XSD), but components such as bar charts, pie charts, and other visualization components understand a predefined data-format, which depends on what visualization API is used (as we use Google Charts API). Without the conversion functionality a new instance of a same visualization component would be needed that specifically implements a component’s specific data visualization requirements that ends in developing too many new UI components.

This is the reason, we use data mappers to encapsulate the data conversion logic so that the data is converted in a format that is understandable by the target component. The current implementation of the data mappers follows our current domain model and hence the conversion is automatically performed for such components. Simply, a new data-mapper will be needed in case of a different domain.

4.5.7 Data Processor

To perform tasks a component is responsible for, component may require to call external services to fetch data or to perform computations. Also, at run-time mashup engine needs to inspect the data that is being transferred between components for various reasons, like, to read/write meta-data, calling external services, conversion of data etc. For this reason, data processors provide a way to the platform to intercept any communication that a component initiates to either interact with other components or to call services. The data processors allow for pre and post processing of the data that components use. Figure 4.4 depicts how a data processor can be seen as a wrapper around a component. Another advantage of the data processors is for the component developers. A component developer can develop and focus on components related implementation concerns without having to worry about platform-specific requirements or conditions. That means, it also increases the reusability of components those belong to the same domain and share similar characteristics.

To further understand the potential of data processors, in following we describe the details of how components communicate with each other having the data processor in place to intercept their calls. As explained already that a composition keeps the information about connected components in terms of events and operations references. These events and operations exchange messages. A message, which is passed from an event to an operation, consists of two parts,
i.e., a **body** and a **header**, exactly as HTTP header and body means to HTTP protocol. The header contains meta-data information, whereas the body contains the actual payload. In our case, the header contains arbitrary meta-data that is not of concern to a component’s implementation. Whereas, the data in a message’s body populated by the data an event emits. A component implementation can only access to the body of a message not to the header as it’s only understandable by the platform.

It is then a data processor’s responsibility that it intercept the communication and process header section accordingly. Data processors make different components to synchronize themselves if they understand the header information. Figure 4.4 depicts the communication between two components along with different stages a message passes through. The figure shows, a white box inside written PL (i.e., payload, the actual data), a black box represents header. This is also the point where the mashup engine decides about switching data-flow approach (the default approach) to control-flow approach. We explain the switching mechanism in the next chapter in section 5.4 after conveying the necessary concepts that are required for its understanding.

### 4.5.8 Configuration Interface

A mashup platform whose target users are non-technical, must offer a user-interface that is intuitive, easy to use and consistent. Less-skilled users tend to prefer high intuitiveness, with relatively less but consistent UI elements to play with configuration settings. We noticed that some mashup tools come with too complex user interface options (e.g., Yahoo Pipes) and others with too limited options (e.g., mashArt). However, a right balance is preferable between the two extremes.

To this end, one possibility is to let components developers decide entirely on what type of configuration interface a component should provide. This approach could lead a component’s configuration interface to arbitrarily complex level, and it might result in major inconsistencies with other components. In order to solve all these issues, the choice we made is to provide a basic set of UI configuration elements along with the parameters the input needs.
Chapter 5

ResEval Mash: A Domain-Specific Mashup Tool

5.1 Overview

The domain-specific mashup platform described in the previous chapter provides a consolidated ground for the development of a domain-specific mashup tool. In this chapter, we present how we have developed such a mashup tool for our reference domain. ResEval Mash\(^1\)\(^{[46]}\)\(^{[61]}\) is a mashup tool tailored to the research evaluation field, i.e., for the assessment of the productivity or quality of researchers, teams, institutions, journals, and the like. The tool is specifically tailored to the need of sourcing data about scientific publications and researchers from the Web, aggregating them, computing metrics (also complex and adhoc ones), and visualizing them. ResEval Mash is a hosted mashup platform\(^{[62]}\) with a client-side editor and runtime engine, both running in a common web browser. It supports the processing of also large amounts of data, a feature that is achieved via the sensible distribution of the respective computation steps over client and server.

In the following, we first present the important design principles which have been learned from our past works, and also in result of those interactions that we did with domain experts. Moreover, we show how ResEval Mash has been implemented, starting from the domain models introduced throughout the previous sections.

5.2 Design Principles

Starting from the considerations that we presented in the section\(^3\)\(^{[3.2]}\), the implementation of ResEval Mash is based on a set of design principles (described below), which we think are crucial for the success of a mashup platform like ResEval Mash. These design principles stem both from our experience with the selected domain and, the earlier work on this direction\(^{[11]}\)^{[63]}\(^{[63]}\). Moreover, these are also based on our past experience with the similar problems in the context of the LiquidPub European project\(^2\).

\(^1\)\[http://open.reseval.org\]
\(^2\)\[http://liquidpub.org/\]
5.2.1 Intuitive graphical user interface

The 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. To this end, we built as very simple yet powerful interface of the tool, that implements domain-syntax model to its various visual parts [64]. That is, the tool visualizes intuitive graphical symbols those of domain-specific nature and easily understandable by domain-experts.

5.2.2 Hidden data mappings

In order to prevent the users from defining data mappings, the mashup component used in the platform are all able to understand and manipulate the domain concepts expressed in the DCM, which defines the domain entities and their relations. That is, they accept as input and produce as output only domain entities (e.g., researchers, publications, metric values). Since all the components, hence, speak the same language, composition can do without explicit data mappings and it is enough to model which component feeds input to which other component.

5.2.3 Data-intensive processes

Although apparently simple, the chosen domain is peculiar in that it may require the processing of large amounts of data. For instance, we may need to extract and process all the publications of the Italian researchers, i.e., on average several dozens of publications by about sixty-one thousand researchers Loading these large amounts of data from remote services and processing them in the browser on the client side is unfeasible due to bandwidth, resource, and time restrictions. Data processing should therefore be kept, especially for this kind of scenarios, on the server side (we achieve this via dedicated RESTful web services running on the server).

5.2.4 Platform-specific services

As opposed to common web services, which are typically designed to be independent of the external world, the previous two principles instead demand for services that are specifically designed and implemented to efficiently run in our domain-specific architecture. That is, they must be aware of the platform they run on. As we will see, this allows the services to access shared resources (e.g., the data passed between components) in a protected and speedy fashion.

5.2.5 Runtime transparency

Finally, research evaluation processes like our reference scenarios focus on the processing of data, which – from a mashup paradigm point of view – demands for a data flow mashup paradigm. Although data flows are relatively intuitive at design time, they typically are not very intuitive at runtime, especially when processing a data flow logic takes several seconds (as
could happen in our case). In order to convey to the user what is going on during execution, we therefore want to provide transparency in the state of a running mashup.

We identify two key points where transparency is important in the mashup model: component state and processing state. At each instant of time during the execution of a mashup, the runtime environment should allow the user to inspect the data processed and produced by each component, and the environment should graphically communicate the processing progress by animating a graphical representation of the mashup model with suitable indications (i.e., in our case we use different colors to represent different states).

These principles require ResEval Mash to specifically take into account the characteristics of the research evaluation domain. Doing so produces a platform that is fundamentally different from generic mashup platforms, such as Yahoo! Pipes.

5.3 ResEval Mash Architecture

5.3.1 Overview

Figure 5.1 illustrates the internal architecture that takes into account the above principles and the domain-specific requirements introduced throughout the previous sections: **Hidden data mappings** are achieved by implementing mashup components that all comply with the **domain conceptual model** described in Figure 3.1. If all instances of domain activities understand this domain concept model and produce and consume data according to it, we can omit data mappings from the composition environment in that the respective components simply know how to interpret inputs. The **processing of large amounts of data** is achieved at the server side by implementing platform-specific services that all operate on a **shared memory**, which allows the components to read and write back data and prevents them from having to pass data directly from one service to another. To provide users with a mashup environment that has an **intuitive graphical UI**, we design first a domain syntax as explained in section 3.7.3, which provides each object in the composition environment with a visual metaphor that the domain expert is acquainted with and that visually convey the respective functionalities. For instance, ResEval Mash uses a gauge for metrics and the icons that resemble the chart types of graphical output components.

The core of the platform is the functionalities exposed to the domain expert in the form of modeling constructs. These must address the specific domain needs and cover as many as possible mashup scenarios inside the chosen domain. To design these constructs, a thorough analysis of the domain is needed, so as to produce a domain process model as described in section 3.7.1, which specifies the classes of domain activities and, possibly, ready processes that are needed (e.g., data sources and metrics). The components and services implement the **domain process model** i.e., all the typical domain activities that characterize the research evaluation domain. **Runtime transparency** is achieved by controlling data processing from the client and animating accordingly the mashup model in the Composition Editor. Doing so requires that each design-time modeling construct has an equivalent runtime component that is able to render its runtime state to the user. The modeling constructs are the ones of the **domain-specific syntax** illustrated in Figure 3.5, which can be used to compose mashups like

---

3  http://pipes.yahoo.com/pipes/
the one in our reference scenario (see Figure 3.6). Given such a model, the Mashup Engine is able to run the mashup according to the meta-model introduced in Section 3.6. The role of the individual module in Figure 5.1 is described as follows:

5.3.2 Mashup Engine

The most important part of the platform is the Mashup Engine, which is developed for the client-side processing, that is we control data processing on the server from the client. We have already presented and described the details of the mashup engine’s internal behavior in the previous chapter. However, here we just try to highlight its interaction with those modules that we introduced in the ResEval Mash’s architecture. The engine is primarily responsible for running a mashup composition, triggering the component’s actions and managing the communication between client and server. As a component either binds with one or more services or with a JavaScript implementation, the engine is responsible for checking the respective binding and for executing the corresponding action. The engine is also responsible for the management of complex interactions among components. A detailed view of these possible interaction scenarios is given later in this chapter.

5.3.3 Composition editor

Figure 5.2 shows ResEval Mash’s composition editor. The composition editor provides the mashup canvas to the users. It shows a components list from which users can drag and drop components onto the canvas and connect them. The composition editor implements the domain-
specific mashup meta-model and exposes it through the domain syntax. From the editor it is also possible to launch the execution of a composition through a run button and hand the mashup over to the mashup engine for execution.

Each container with a symbol on the canvas represents one component with its name, a symbol, configuration interface, and in case of a UI component it’s output interface. On the left side of a component shows its input ports (i.e., where it accepts connections) and on the right side it shows output ports (i.e., where it emits data to the next connected components). Components can be expanded to view their configuration interface and various ports. To connect two components, a user has to click and drag from the output port of a source component and drop and release click on input port of a target component. Figure 5.3 depicts how connections can be performed. Composition editor highlights all compatible ports (i.e., checking domain concepts compatibility) of all components present on the canvas upon a mouse click on a component’s event (i.e., output port), that is how editor knows that user intends to make a connection.

To provide run-time transparency, which convey to users the state of each component’s execution, composition editor shows various visual states of the components. A component can have and change among three visual states. These visual states correspond to a component’s execution status, like a component which is not in the running state shows its label and boarder in black color. Whilst, a component which is in the running state shows an extra label in yellow color right below the component, which shows the operation name which is being executed. The third state represents the successful execution of a component and shows both component name
ResEval Mash’s composition editor highlighting compatible ports upon making connections among components and its boarder in green color. The fourth and the final state, which shows component’s boarder and a notice in a red color that represents the component execution failed due to some reason.

5.3.4 Component Registration Interface

The tool also comes with a component registration interface for developers, which aids them in the setup and the addition of new components to the platform. The interface allows the developer to define components starting from ready templates. In order to develop a component, the developer has to provide two artifacts: (i) a component definition (Figure 5.4) and (ii) a component implementation. The implementation consists either of JavaScript code for client-side components or it can be linked to a web service, which is achieved providing a binding to a web service for server-side components.

To provide ease to component developers, especially with dynamic, untyped languages such as JavaScript, testing and debugging can take much time since errors are often only discovered when the code is executed, the editor provides a supportive interface which allows easy adjustments to the code directly in the browser. Developing inside the browser is not very popular yet, but is possible, especially for languages native to the browser environment, such as JavaScript. Developers do not need to upload code changes of a component repeatedly after some changes are made as the editor automatically identifies that new changes are available and hence it deploys new version.

The component editor consists of two separate editors, one for the component model definition and one for the implementation. Both editors provide syntax highlighting and rudimentary code completion support. For example, the component model editor offers code completion for the elements of our XML representation. This is realized with the help of the CodeMirror\textsuperscript{4} library.

\textsuperscript{4}http://codemirror.net/
5.3.5 Server-Side Services

On the server side, we have a set of RESTful web services, i.e., the repository services, authentication services, and components services. Repository services enable CRUD operations for components and compositions, that is, the mashup engine interacts with these services to perform CRUD operations. Authentication services are used for user authentication and authorization. Components services manage and allow the invocation of those components whose business logic is implemented as a server-side web service. These web services, together with the client-side components, implement the domain process model. The idea behind these services was to move the computation from the client side to the server side to improve performance by utilizing server’s computational power and big memory hence reducing client side burden. The interaction details between client side components and their services is explained in section 5.5. And, a detail explanation of how to develop a service for a component is given in section 5.6.

5.3.6 CDM Memory Manager, CDM Module & Shared Memory

The common data model (CDM) memory manager enforces and supports the checking of data types in the system. To use the domain-specific data-types (i.e., DCM concepts) for various modules on the server side, it is necessary to have an interface that can read a domain-specific model (e.g., an XML schema definition (XSD) in our case), and can parse it, generate implementation classes. These classes then expose their definitions to the other modules. In order to configure the CDM, the CDM memory manager generates corresponding Java classes (e.g.,
in our case these classes are POJO, annotated with JAXB annotations) from an XSD that encodes the domain concept model. Having the required data-types context in place, the CDM memory manager is then responsible for the insertion and retrieval of data to and from the shared memory.

On the server side, a shared memory is maintained that CDM memory manager uses it to read and write data. The shared memory can store multiple states of data, and also multiple instances of the same data. However, all the data must comply with the data-types provided by the common data model module. The insertion and retrieval to and from the shared memory follows key-value pair mechanism, where the value represents actual data and key works as an identifier. Client side mashup engine initially generates a key, which passes along the data that is then used by the CDM memory manager to store in the shared memory. Hence, all data processing services read and write to this shared memory through the CDM memory manager. That means, the CDM interacts with the shared memory to provide a space for each mashup execution instance if required. In our first prototype we use the server’s working memory (RAM) as shared memory, which allows for high performance. Clearly, this solution fits the purpose of our prototype but it may not scale to in-production installations, which may need to deal with large numbers of users and large amounts of data that only hardly can be kept in RAM if it offers small memory. However, in our future work, we aim to develop a persistent database-based shared memory.

5.3.7 Local Database and the Web

Both the database and the Web represent the data which is required and used by the component services. We as a platform provider provide a database and a basic set of services on top of it. A third-party service can be deployed and thus it can use an external database anywhere on the Web. However, the development of a third-party service must comply with the specification presented later in the section 5.6.

5.4 Intelligent Switching between Data-flow and Control-flow

As explained earlier that the use of server-side web services is one way to implement business logic of a component, which can also be implemented through JavaScript that is we call these components as client-side components. For components whose business logic is implemented by server side services require to send data from client side to the server side for various processing. Moreover, if a composition comprised of more than one such component then for each such component data must be sent to the server side from the client and vice-versa. This scenario will be even worse, especially if the data (i.e., data which is being used in communications) are bigger; as in our case too, then following this strategy poses serious challenges in terms of speed that decreases an overall performance of the platform. To deal with data-intensive compositions, which deals with huge amounts of data, the traditional mechanism (i.e., data-flow back and forth between client and server) is not an appropriate choice. For this reason, the

---

5 The database holds data that we have crawled, downloaded from various sources for performance purposes.
The platform achieves the functionality of intelligently switching between data to control flow and back to data-flow with the help of data processors presented in 4.5.7. The data processors are designed to intercept any operation call, request or event. To call a corresponding web service of a component, it is required to intercept whenever an operation of the component is called, and sending a request to the service. This is achieved by configuring a ServiceCall data process, a specific type of the data processor. The operation "get researchers" (line 5) is configured, having a service URL (line 6), with passthrough parameter as false and overwrite parameter as true and also mentioning what type of service it is (line 9). The passthrough parameters represents if the data (i.e., the service response) will be needed by the component implementation (i.e., client-side implementation) for performing some actions before it will be handed over to the next component. The overwrite parameter represents if the original data
will be overwritten or not. These lines of configuration make sure that whenever an operation of a component is called, the data processor checks for its ServiceCall configuration.

The overall procedure, which is to detect and to decide when and where to send data, is represented in the flow chart in Figure 5.5 and explained as follows. In case, if a servicecall is configured, the data processor inspects the header of the message for a key (i.e., a numeric identifier comprises of composition id and time stamp information) which should be dispatched to the service to be called. The key is used on the server side and used as an identifier for the data in the shared memory. A new key is generated by the data processor if one is not found in the message, which also means the particular data does not exist on the server side.

The data processor further checks if some data already exist in the message body that will be dispatched to the server along with the Data Request parameter. The data request can be set either as "yes" or "no", which describes whether the response of the service has to be sent back to the client side or not. The decision about setting data request as "yes" or "no" depends on the next component in the composition connected to the component that is being inspected. A component whose business logic is implemented as a JavaScript file requires the data on the client side for processing, on the other hand a component whose business logic is implemented by a service on the server side does not require the data to be present on the client side. That is how the mashup engine using data processors detects and makes decisions accordingly. The details of each particular interaction scenario between client and server side components as described in the next section 5.5.
Figure 5.7 depicts how the data processor intercept call upon an event trigger. First it checks whether the triggered event is the result of an operation or a request. In both cases, the control data presence is checked. If control data is available, the key is set in the header, the response data is set in the body of the message. Figure 5.7 depicts the flow chart of the mechanism used to detect whether the target component is a client-side or a server-side component. The result is then used to configure dataRequest parameter values that decides whether to fetch data or not on the client-side.

5.5 Components Models and Data Passing Logic

There are two component models in ResEval Mash, depending on whether the respective business logic resides either on the client or the server side: server components (SC) are implemented as RESTful web services that run at the server side; client components (CC) are implemented as JavaScript file and run on the client side. Independently of the component model, each component has a client-side component front-end, which allows (i) the Mashup Engine to enact component operations and (ii) the user to inspect the state of the mashup during runtime. All communications between components are mediated by the Mashup Engine, internally implementing a dedicated event bus for shipping data via events. Server components require interactions with their server-side business logic and the shared memory; this interaction needs to be mediated by the Mashup Engine. Client components directly interact with their client-side business logic; this interaction does not require the intervention of the Mashup Engine.

The components consume or produce different types of data: actual data (D), configuration parameters (CP), and control data like request status (RS), a flag that conveys whether actual data is required in output (DR), and a key (K) identifying data items in the shared memory. All components can consume and produce actual data, yet, as we will see, not always producing actual data in output is necessary. The configuration parameters enable the setup of the components. The request status enables rendering the processing status at runtime. The key is crucial to identify data items produced by one component and to be ”passed” as input to
another component. As explained earlier, instead of directly passing data from one service to another, for performance reasons we use a shared memory that all services can access and only pass a key, i.e., a reference to the actual data from component to component.

Based on the flow of components in the mashup model, we can have different data passing patterns. Given the two different types of components, we can recognize four possible interaction patterns. The four patterns are illustrated in Figure 5.8. All of these interactions are mediated by the mashup engine, hence neither a composition composer nor a component developer needs to think about these complexities while component or composition development. In particular, we may have two types of interaction, that is, (i) the interaction among components that are connected in the designed composition and (ii) the interaction among a component and its server-side implementation (only in the case of components of type SC). Both these types of interaction are managed by the Mashup Engine. In the first type, the Mashup Engine manages the event bus used to publish the components’ events (carrying associated data) and trigger the subscribed components’ operations. In the second type, the Mashup Engine acts as a proxy for the web service operation invocation with the help of data processors. In both cases, the Mashup Engine has the role of managing and including the correct control data in all the events and service invocations, that is crucial for letting the platform work properly. In following we elaborate individual interaction pattern separately.

1. **SC-SC interaction:** As shown in the Figure 5.8 both the components (SC A & SC B) are of type SC. Component A is connected with component B. Since component A is the first component in the composition and it does not require any input, it can start the execution immediately. It is the responsibility of the Mashup Engine to trigger the operation of the component A (step 1). At this point, component A calls its back-end web service through the Mashup Engine, passing only the configuration parameters (CP) to it (2). The Mashup Engine, analyzing the composition model, knows that the next component in the flow is also a server component (component B), so it extends component A’s request adding a key control information to the original request, which can be used by component A’s service to mark the data it produces in the shared memory. Hence, the Mashup Engine invokes service A (3). Service A receives the control data, executes its own logic, and stores its output into the Shared Memory (4). Once the execution ends, Service A sends back the control data (i.e., key and request status) to the Mashup Engine (5), which forwards the request status to component A (6); the engine keeps track of the key. With this, component A has completed and the engine can enable the next component (7). In the SC-SC interaction, we do not need to ship any data from the server to the client.

2. **SC-CC interaction:** Once activated, component B enacts its server-side logic (8, 9, 10). The Mashup Engine detects that the next component in the flow is a client component, so it adds the DR control data parameter in addition to the key and the configuration parameters, in order to instruct the web service B to send actual output data back to the client side after it has been stored in the Shared Memory. In this way, when service B finishes its execution, it returns the control data and the actual output data of the service (i.e., key, request status and output data) to the Mashup Engine (11), which then passes the request status to component B (12) and the actual data to the next component in the
mashup, i.e., component C (13).

3. **CC-CC interaction:** Client component to client component interactions do not require to interact with the server-side services. Once the component C’s operation is triggered in response to the termination of component B, it is ready to start its execution and to pass component B’s output data to the JavaScript function implementing its business logic. Once component C finishes its execution, it sends its output data back to the engine (14), which is then able to start component D (15) by passing C’s output data.

4. **CC-SC interaction:** After the completion of component D (16), the Mashup Engine passes the respective data to component E as input (17). At this point, component E calls its corresponding service E, passing to it the actual data and possible configuration parameters (18), along with the key appended by the Mashup Engine (19). Possibly, also the Output Data Request flag could be included in the control data but, as explained, this depends on the next component in the flow, which for presentation purposes is not further defined in Figure 5.8. Eventually, service E returns its response (i.e., key and request status – plus possible output data if the DR flag is present) to the Mashup Engine (21), which is then delivered to component E (22).
While ResEval Mash fully supports these four data passing patterns and is able to understand whether data are to be processed at the client or the server side, it has to be noted that the actual decision of where data are to be processed is up to the developer of the respective mashup component. Client components by definition require data at the client side; server components on the server side. Therefore, if large amounts of data are to be processed, a sensible design of the respective components is paramount. As a rule of thumb, we can say that data should be processed on the server side whenever possible, and component developers should use client components only when really necessary. For instance, visualization components of course require client-side data processing. Yet, if they are used as sinks in the mashup model (which is usually the case), they will have to process only the final output of the actual data processing logic, which is typically of smaller size compared to the actual data sourced from the initial data sources (e.g., a table of h-indexes vs the lists of publications by the set of the respective researchers).

5.6 The Domain-Specific Service Ecosystem

An innovative aspect of our mashup platform is its approach based on the concept of domain-specific components. In Section 5.3 we described the role of the Components services in the architecture of the system. These are not simply generic web services, but web services that constitute a domain-specific service ecosystem, i.e., a set of services respecting shared models and conventions and that are designed to work collaboratively where each of them provides a brick to solve more complex problems proper of the specific domain. Having such an ecosystem of compatible and compliant services, introduces several advantages that make our tool actually usable and able to respond to the specific requirements of the domain we are dealing with.

Given the important role domain-specific components and services play in our platform, next we describe how they are designed and illustrate some details of their implementation and their interactions with the other parts of the system.

A ResEval Mash component requires the definition of two main artifacts: the component descriptor (i.e., following the component definition language specifications) and the component implementation.

The *component descriptor* describes, to briefly mention, the main properties of a component, which are:

1. *Operations*. Functions that are triggered as a consequence of an external event that take some input data and perform a given business logic.

2. *Events*. Messages produced by the component to inform the external world of its state changes, e.g., due to interactions with the user or an operation completion. Events may carry output data.

3. *Implementation binding*. A binding defining how to reach the component implementation.

4. *Configuration parameters*. Parameters that, as opposed to input data, are set up at composition design time by the designer to configure the component’s behavior.
5. **Meta-data.** The component’s information, such as name and natural language description of the component itself.

In our platform the component descriptors are implemented as XML file, which must comply with an XML Schema Definition (XSD). The XSD defines both the schema for the component descriptors and the admitted data types. Validating the descriptor against the data types definition we can actually enforce the adoption of the common domain concept model (DCM), which enable smooth composability and no need for data mapping in the Composition Editor, as discussed in Section 5.2.

For example, an excerpt of the Italian Researchers component descriptor along with its representation in the Composition Editor is shown in Figure 5.9. The component is implemented through a server-side web service. Its descriptor does not present any operation and it has an event called **Researchers Loaded**, which is used to emit the list of researchers that are retrieved by the associated back-end service. The binding between the service and its client-side counterpart is set up in the descriptor through the `<request>` tag. As shown, this tag includes the information needed to invoke the service, i.e., its end-point URL and the configuration parameters that must be sent along with the request. In addition, the attribute `triggers` specifies the event to be raised upon service completion. The attribute `runOn`, instead, specifies the component’s operation that must be invoked to start the service call. In this particular case, since the component has no operations and no inputs to wait for, when the mashup is started the Mashup Engine automatically invokes the back-end service associated with the component, causing the process execution to start. If we were dealing with a component implemented via client-side JavaScript, we would not need the `<request>` tag, and the implementation binding
would be represented by the ref attribute of the component operation or event, whose value would be the name of the JavaScript function implementing the related business logic.

The component in Figure 5.9 has different configuration parameters, which are used to define the search criteria to be applied to retrieve the researchers. We can see the uniId parameter. Beside the name of the related label, we must specify the renderer to be used, that is, the way in which the parameter will be represented in the Composition Editor. In this case, we are using a text input field with auto-completion features. The auto-completion feature is provided by a dedicated service operation that can be reached at the address specified in the url option. Finally, we can see the presence of the configTemplate tag, which is just used to set the order in which the parameters must be presented in the component representation in the Composition Editor.

The other main artifact that constitutes a ResEval Mash component is its implementation. As already discussed above, a component can be implemented in two different ways: through client-side JavaScript code (client component) or through a server-side web service (server component). The choice of having a client-side or a server-side implementation depends mainly on the type of component to be created, which may be a UI component (i.e., a component the user can interact with at runtime through a graphical interface) or a service component (i.e., a component that runs a specific business logic but does not have any UI). UI components (e.g., the Bar Chart of our scenario) are always implemented through client-side JavaScript files since they must directly interact with the browser to create and manage the graphical user interface. Service components (e.g., the Microsoft Academic Publications of our scenario), instead, can be implemented in both ways, depending on their characteristics. In the research evaluation domain, since they typically deal with large amounts of data, service components are commonly implemented through server-side web services. In such a way, they do not have the computational power constraints present at the client-side and, moreover, they can exploit the platform features offered at the server-side, like the Shared Memory mechanism, which, e.g., permit to efficiently deal with data-intensive processes. In other cases, where we do not have particular computational requirements, a service component can be implemented via client-side JavaScript, which runs directly in the browser. The JavaScript implementation, both in case of UI and service components, must include the functions implementing the component’s business logic.

For example, our Italian Researchers service component is implemented at server-side since it has to deal with large amounts of data (i.e., thousands of researchers), so it belongs to the server components category (introduced in Section 5.5). This type of components, to correctly work within our domain-specific platform, must be implemented as Java RESTful web service following specific implementation guidelines. In particular, the service must be able to properly communicate with the other parts of the system and, thus, it must be aware of the data passing patterns discussed before and the shared memory. Figure 5.10 shows the interaction protocol with the other components of the platform the service must comply with.

The service is invoked through an HTTP POST request by the client-side Mashup Engine, performed through an asynchronous Ajax invocation (the half arrowheads in the figure represent asynchronous calls). The need to expose all the operations through HTTP POST comes from the fact that in many cases it must be possible to send complex objects as parameters to the service, which would not be possible in general using a GET re-
The Domain-Specific Service Ecosystem

Figure 5.10: Platform-specific interaction protocol each service must comply with

quest. For instance, in our example, the operation is invoked through a POST request at the URL http://.../resevalmash-api/resources/italianSource/researchers and the component’s configuration parameters (e.g., selected university or department) are posted in the request body. Besides the parameters, the body also includes control data, that is the key and the OutputDataRequired flag.

Once the request coming from the Mashup Engine is received by the service, the service code must process it following the sequence diagram shown in Figure 5.10. If the service is designed to accept input data, first it will get the data from the Shared Memory through the API provided by the Server-Side Engine, using the received key as parameter.

Then, the service may need to have access to other data for executing its core business logic. The services developed and deployed by us (as platform owners) can use the system database to persistently store their data, as shown in the second optional box. This is, for instance, the case of our Italian Researchers component that retrieves the researchers from the system database, where the whole Italian researchers data source has been pre-loaded for efficiency reasons. Third-party services, instead, do not have access to the system database but they can use external data sources as external databases or online services available on the Web. Clearly, the usage of the system database guarantees higher performances and avoids possible network
bottlenecks.

Once the service has retrieved all the necessary data, it starts executing its core business logic (for our example component, it consists in the filtering of the researchers of interest based on the configuration parameters). The business logic execution results are then stored in the Shared Memory. Typically, all the services will produce some output data, although, possibly, there could be exceptions like, for instance, a service that is only designed to send emails. Finally, the service must send a response back to the Mashup Engine. The response content depends on the **OutputDataRequest** flag value. If it is set to false, as shown in the upper part of the alternative box in the figure, the response will contain the **Key** and the **RequestStatus** of the service (success or error). If the flag is set to true, in addition to those control data, the response will also contain the actual **OutputData** produced by the service logic.

So far, all components and services for ResEval Mash have been implemented by ourselves, yet the idea is to open the platform also to external developers for the development of *custom components*. In order to ease component development, e.g., the setup of the connection with the Shared Memory and the processing of the individual control data items, we will provide a dedicated Java interface that can be extended with the custom logic. The description, registration, and deployment of custom components is then possible via the dedicated Component Registration Interface briefly described in Section 5.3.
Chapter 6

User Studies and Evaluation

6.1 Overview

To evaluate different aspects associated with the work presented in this thesis, like whether being domain-specific is preferable or not for end-users; and if yes, how much expressive a tool should be, that is, what level of flexibility a mashup tool should offer so that the corresponding complexity stays within the boundaries of non-technical users. For this purpose, we conducted a few user studies. The first study, which was mainly focused on the usability evaluation of our mashup tool also partially used for the comparative analysis between ours and other mashup based tool. To this end, to understand the users’ preference over domain-specific versus generic mashup tools, we used Yahoo! Pipes as a generic tool example. Moreover, to determine what level of complexity a non-technical domain-expert can deal with in case of a domain-specific mashup tool preference, we built four different prototypes. Each prototype encompasses different level complexity which surly depends on the flexibility and customization features that these tools provide.

After the first user study, we addressed and incorporated the suggestions, feedbacks and new requirements gathered during the first user study. The improvements and changes made our mashup tool more usable and useful. Again, to validate the usability of the ResEval Mash tool, we conducted the second user study to test relatively advance features specifically related to the usability of the ResEval Mash tool. The next section elaborate on the first user study and the section 6.3 presents the details of our second user study. Finally we conclude this chapter with analysis and discussion.

6.2 Comparative & Usability Evaluation: User Study-1

Being generic versus specific is fundamentally different from their roots, that is, the former covers a broader level in comparison to the latter, which stays specific and can be a specialized form of what former offers. For instance, in this context, simply asking someone to make a computer application and to its opposite asking to make an accounting application is different. Likewise, for building generic or specific application depends what constructive constructs are provided to a developer that is we believe especially for non-technical users those constructs would be more valuable if they are aligned to their level of domain-expertise. Moreover, what
a development environment conveys or understand from an end-user point of view is different, like whether the language that an application speaks understandable by the end-users or not. In order to assess all these aspects, we designed our first user evaluation experiment as described in the next section.

6.2.1 Task Design

To evaluate the different aspects related to the usability of our domain-specific tool as well as to determine users’ preference, we performed contextual interviews of 28 users. Among them, 7 were professors, 5 administrative people, 1 post-doc, 3 PhD students and 12 master’s level students. These participants were having different levels of technical skills. The technical skills of the participants were determined asking the following questions:

- What is the user skill level with tools such as MS Excel, MS Word etc?
- Is the user aware of the meaning of web service?
- Is the user able to draw/understand a (simple) process following a given graphical notation (e.g., flowchart)?

In addition to the above mentioned questions, the participants were also asked to write (in a text box) if they program computer applications, or involved in programming tasks and other details related to their technical as well as domain skills. From those recorded answers, we can say that all of the users were domain-experts (i.e., they know research evaluation and were involved in some kind of evaluation tasks), excepts the master’s level students, which were having low domain-expertise than the former group. Among all the user, 5 administrative were highly domain-expert and most of them were directly involved in the research evaluation task that we used during the study. Table 6.1 presents the details of all the participants with their technical and domain skills.

<table>
<thead>
<tr>
<th>Users</th>
<th>Position</th>
<th>Technical Skills</th>
<th>Domain Skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>professor</td>
<td>4 (good skilled), 3 (moderate)</td>
<td>very skilled</td>
</tr>
<tr>
<td>5</td>
<td>administrator</td>
<td>4 (moderate), 1 (very skilled)</td>
<td>very skilled</td>
</tr>
<tr>
<td>1</td>
<td>post-doc</td>
<td>good skilled</td>
<td>very skilled</td>
</tr>
<tr>
<td>3</td>
<td>PhD</td>
<td>good skilled</td>
<td>good skilled</td>
</tr>
<tr>
<td>12</td>
<td>master's level student</td>
<td>moderate</td>
<td>moderate</td>
</tr>
</tbody>
</table>

Table 6.1: User Details

We use one of the famous mashup tool Yahoo! Pipes as a generic tool example and our ResEval mashup tool as a domain-specific one. To better understand the appropriate level of expressive power, flexibility, and difficulty that our tool (i.e., ResEval Mash) offers, we developed four separate prototypes of the tool based on our selected scenario, each prototype offered a different level of flexibility and, consequently, complexity.

6.2.2 Evaluation Procedure

First, the participants were asked about their technical and domain -skills (to assign them to the appropriate user category as mentioned in the previous section), then we instructed them to perform the following tasks in steps, providing help only upon explicit request:
In this scenario we take University of Trieste departments evaluation procedure, which is being used to distribute funds among departments. The procedure is itself complex further more data requirements from different scholarly data sources (e.g., Scopus, DBLP etc.) its processing and visualization of results with different algorithms makes it more tedious job for a human to accomplish. We try to solve it using domain specific mashup solution. Following are the summarized steps after reading almost 10 pages of actual procedure.

1. A list of all researchers working in Italian universities is retrieved from a national registry, and a reference sample of faculty members with similar statistical features (e.g. belonging to the same disciplinary sector) of the evaluated department is compiled for the selected UniTN department.

2. Bibliometric data (publications) for each researcher of the selected department and for all Italian researchers in the selected sample are extracted from an agreed data source (e.g., Microsoft Academic, Scopus, DBLP, etc.).

3. The publications obtained in the previous step are then weighted using a venue classification: i.e., the publication venues of the scientific production are classified by an UniTN committee in three quality categories mainly based on ISI Journal Impact Factor: A/1.0 (top), B/0.6 (average), C/0.3 (low) - by an UniTN committee. For each researcher a single weighted publication count is obtained with a simple weighted sum of the his/her publications.

4. The statistical distribution - more specifically a negative binomial distribution - of the weighted publication count metric is then computed for the researchers’ reference sample.

5. Each department researcher is then positioned, based on his/her individual weighted publication count, over the computed referenced statistical distribution, thereby estimating his/her percentile, i.e. the percentage of the researchers in the same disciplinary sector that have the same or lower values for the specific metric.

6. The percentile for each researcher in the selected department is used as the parameter that estimates the publishing profile of that researcher and is used for the comparison with other faculty members in same department.

7. Moreover, the sum of the individual percentiles for each department researcher is used as the parameter that estimates the publishing profile of that department and is used for the comparison between other departments and, in the end, to determine the distribution of the research budget within UniTN.

Figure 6.1: Prototype-1: fixed components with fixed configuration options
1. In the first step, we introduced prototype 1 as depicted in Figure 6.1. This prototype consists of an explanation of the scenario and a pictorial form of it in which different components are connected making a composition. On the same page we provide a button to start the execution of the process. That is, the mashup is pre-built (by us; acted as a developer) and can only be executed by the participants. The participants were only allowed to click the start button in order to execute the process and the final results shown to them on the same page. The provided composition did not allow the participants to make any type of interaction with the components thus restricting them only with the control over process execution (i.e., start or stop).

2. In the next step, prototype 2 was presented, as depicted in Figure 6.2. This time the participants were presented the same scenario with configurable components, as it can be seen in the figure that the components’ configuration panels are open. Thus allowing participants to configure components through components’ configuration panels, that is various parameters (e.g., filtering options, date ranges etc.) values can be of user-defined. Once the configured components are ready, the participants can start the process execution using a start button.

3. In this step, prototype 3 was presented to the participants as depicted in Figure 6.3. The participants are now allowed to change components in the mashup model by choosing among different implementations of the same component class, again through configuration panels. For example, a Microsoft Academic Search (MAS) data source can be replaced with a DBLP data source. The possibility to change the configuration parameters and to substitute the components, provides more flexibility to the participants in order to tune the scenario according to their needs.

4. During this step, a fully functional mashup composition environment (i.e., ResEval Mash) was presented to the participants. This tool provides the possibility to drag-and-drop components onto a composition canvas, to fill their configuration parameters, to connect them together and to execute composition, hence giving maximum flexibility to the participants so they can be as expressive as they want. The participants during this step were allowed to use all the features of the ResEval Mash tool, as during this step they were asked to compose a mashup composition based on the scenario they have been experiencing during the previous steps.

5. Finally, we presented to the participants the Yahoo! Pipes tool, which is a popular generic mashup tool. An example pipe, i.e., a composition, is shown to the participants as a short tutorial to introduce the tool. Then, participants were asked to imagine how they would implement our specific scenario in Pipes and were asked to implement it to whatever level they can reach.

### 6.2.3 Questionnaires

After each step of the procedure, which are mentioned in the previous section, the participants were presented with a set of questionnaires to answer. The questions related to various aspects like, what difficulties they encountered, their understandability level, their suggestions for the
Figure 6.2: Prototype-2: Showing a more customizable approach, where user allowed to configure the components.

improvements etc. were asked. In following the detail of these questions is presented, and the metaphors used for recording their answers are mentioned in (parentheses):

1. What is your opinion about the difficulty level of this task? (a set of six radio-option buttons ranging from extremely difficult to extremely easy were presented)

2. What are the main difficulties you encountered? and why? (a multi-line text box)

3. What are the advantages of this step as compared to the manual/previous approach? (a multi-line text box)

4. What do you think are the disadvantages of this step as compared to the manual/previous step approach? (a multi-line text box)

5. Do you think the increased flexibility of the tool with respect to the previous step would
be useful for you to adapt the process to your specific needs? (ten radio-option buttons ranging from extremely efficient to extremely inefficient)

6. Do you understand how the process executed behind the scene? (ten radio-option buttons ranging from easily understandable to not understandable at all)

7. Do you feel comfortable having such control over the process execution or you would like to have a clearer idea of what is going on? (ten radio-option buttons ranging from extremely comfortable to extremely uncomfortable)

8. Based on your experience during this step, would you prefer to do this task by yourself using a similar tool or you would prefer explaining and asking a technician to implement it for you? (two options were presented: (i) I’d like to do by myself (ii) I’d like to ask technician)

9. Do you have specific suggestions/requirements to improve usability/usefulness of this tool? (a multi-line text box)

10. Are you happy about the flexibility given by the current tool or you would like to be able to change something to adapt the tool to be used for solving other similar problem? (a multi-line text box)

11. In your opinion what is the difference between previous and this step? (a multi-line text box)

The participants completed the above mentioned questionnaires by themselves after the completion of each step, help was provided if asked by any participants. Finally, after the 5th step that is described in the section 6.2.2 in which we also presented the Yahoo! Pipes tool, we asked participants a set of general questions in order to constitute an overall consensus among all the tools with respect to the flexibility, usefulness and complexity they offer. These questions are presented below, answering options are in parentheses. Throughout the phase of answering questions the participants were allowed to ask assistance if they have difficulty in understand a question. Final questions were as follows:

1. What do you think what is the complexity of computing a research evaluation metric manually? (ten radio-option buttons ranging from extremely complex to extremely easy)

2. Now, you have seen all the different tools (steps), how would you judge them? (for all the five steps, we presented three radio-option buttons with flexible, useful and complex as options.)

3. Which step/tool would you consider closer to your needs considering both simplicity and flexibility among 5 different tools and why? (a multi-line text box)

4. Would you use this tool in your real life? (yes/no)
6.2.4 Results

As the objective of the study was to collect feedback about two main questions. First, if the participants are indeed more comfortable with domain-specific mashup tools compared to general purpose tools like Yahoo Pipes. Second, in the case domain-specific tools are preferred, which is the right tradeoff among flexibility and complexity, i.e., which of the tools 1-4 is most effective.

Results of the first step

As for the first step in which we presented the prototype that allows the participants only to the execute process, leaving configurations and component modification options fixed (i.e., hard coded). The results of the quantifiable questions (question number 1, 5, 6 and 7) whose answers
What is your opinion about the difficulty level of this task?

Do you think this tool would save time as compared to perform the procedure manually or with the help of other tools like MS office tools?

Do you understand how the process executed behind the scene?

Do you feel comfortable having such control over the process execution or you would like to have a clearer idea of what is going on?

Figure 6.4: Results of user study-1, prototype-1

The figure shows four charts in which it can be noticed that in chart-1, most of the participants found the first step extremely easy to perform, as the prototype used in this step had lowest complexity that is to only execute the process with a button click. As chart-2 depicts, most of the participants consider the approach as an efficient upon asking whether they feel the approach is time saving as compared to the other tools or manual effort. However, the chart-3 shows, a half-half division of the opinions when the participants were asked whether they understand how the process executed behind the scene or not. As no execution status was conveyed, nor a progress bar was shown, and also they were having no idea how data is flowing, so many participants remained in dark showing they did not understand it. The chart-4 shows most of the participants feel uncomfortable with the level of control that the tool offers, which implies that they certainly need more control over the execution of the process.

Regarding the remaining questions (i.e., 2, 3, 4, 8, 9, 10 and 11; as presented in the section 6.2.4) whose answers were collected in textual format, we show here some important responses. For the second question, main responses were "not flexible, but simple", "not difficult", "I can't use for my daily job, as I need to deal with loads of variations". For the third question typical answers were, "easier, fast, efficient", "not difficult at all", "much faster than the manual approach" etc. Regarding the fourth question, user respond like, "you can not check what's going on behind the scene", "unskilling people", "all data are registered with more accuracy and
Comparative & Usability Evaluation: User Study-1

Results of the second step

In response to the second step, which is mentioned in the section 6.2.2, Figure 6.5 depicts the charts of the four questions that are 1, 5, 6 and 7. The chart-1 shows that again the participants found step-2 task (i.e., prototype-2) easy to use. As shown in the chart-2, most of the participants liked the increased flexibility of the tool. The third chart shows that almost an equal division of the perception of the understanding of the execution of the process. However, most of the participants still not comfortable using this tool and thus demanded for more control over the process, as depicted in the chart-4. Regarding questions (i.e., 2, 3, 4, 8, 9, 10 and 11; as presented in the section 6.2.4) whose answers are in text format, mainly the users' responses to the second question were "more flexible than the previous but not flexible to change components", "understanding configuration parameters" etc. For the third question responses were "parameteric approach is better than the previous one", "its flexible", "you can choose different parameters", "auto-completion is perfect for me" etc. The fourth question received responses like "still no idea what's behind the process", "still not able to change the execution flow", "not always reliable" etc.

In response to the eighth question, 90% said "I'd like to do by myself" and 10% said "I'd like to ask technician". The ninth question where participants gave suggestions as "if you provide me right choice of component then I can do myself", "it should provide the ability to check input parameters", "details about what kind of data are being used", "better than one button" etc. The tenth question received like "more flexibility would be good for me", "I'd like to change the flow of execution", "more options are needed" etc. In response to the eleventh question, which was not the part of the first step, the participants respond as "more customization of the search", "i have more flexibility", "more flexibility" etc.

Results of the third step

In response to the third step, which is mentioned in the section 6.2.2, Figure 6.6 depicts the results of the four questions (i.e., 1, 5, 6 and 7; presented in the section ). As shown in the chart-1, majority of the participants still feel that the difficulty of the presented step is manageable and hence easy to handle. Likewise, most of the participants considered the increased flexibility still an efficient approach, as shown in the chart-2. However, the chart-3, which conveys the understanding level of the execution of the process, still plot that largely the process execution was less understandable for many. The chart-4 shows many of the participants remain uncomfortable, which means they still demand for more control over the process. On the other side, from textual answers, for the second question main responses were "no difficulties", "more details about the sources would be good", "more documentation", "none" etc. For the
third question “more clear, more accurate”, “more flexible than the previous one”, “presentation styles changing is good” etc. were the main responses and for the fourth question “provide more choices”, “its more error prone”, “none” etc. were the main responses. In response to the eighth question, 93% said “I’d like to do by myself” and 7% said “I’d like to ask technician’.

As for the ninth question, the participants suggested as: “having more visual charts would be more useful”, “having more sources would be good”, “check configuration parameter validity” etc. Regarding the tenth question, main responses were “yes I’m happy, but if you provide me more flexibility I will be more happy”, “I’d like to alter the execution flow”, “I’d like to change flow”, “yes” etc. For the eleventh question, the participants differentiated this tool with the previous as “more flexibility”, “this is better”, “you can choose other databases” etc.

Results of the fourth step

As stated in the section 6.2.2 that we presented to the participants the fully functional tool (i.e., ResEval Mash) during the fourth step. The responses regarding the questions (1, 5, 6 and 7) are depicted in the Figure 6.7. In the chart-1, it is clearly shown that this prototype slightly increased the difficulty level but still majority voted it as easy to use. On the other hand, the increased flexibility of the tool was mainly perceived positively and the majority felt it as an efficient approach, as depicted in the chart-2. However, this prototype was extremely understandable by the participants as compared to all the previous ones, as depicted in the
(1) What is your opinion about the difficulty level of this task?

(2) Do you think the increased flexibility of the tool with respect to the previous step would be useful for you to adapt the process to your specific needs?

(3) Do you understand how the process executed behind the scene?

(4) Do you feel comfortable having such control over the process execution or you would like to have a clearer idea of what is going on?

Figure 6.6: Results of user study-1, prototype-3

chart-3. Moreover, as depicted in the chart-4, majority of the participants felt comfortable with the control that the prototype-4 provided.

Regarding the responses against the questions (2, 3, 4, 8, 9, 10 and 11; as presented in the section 6.2.4), for the second question main responses were "no", "the more you are free in changing configuring component, the more you are risky", "need to know components", "documentation of the components required" etc. For the third question mainly the users’ responses were "Its all advantages and not disadvantages", "the more you can personalize the more you feel comfortable", "i can customize", "I can better adopt my needs", "I can reuse compositions, components’ etc. And for the fourth question, “more degree of freedom so there could be more chances of error”, "more knowledge is required”, "takes time to understand” etc. In response to the eighth question, 95% of the participants said "I’d like to do by myself", whereas 5% of the users said "I’d like to ask technician". Main responses for the question number nine were "some training course for the user would be good”, "add more components”, "give suggestion/assistance to the user during composition”, “check intermediate results” etc. The tenth question was received as "I am fine, not more than this”, "I think its enough", "I don’t need more details than this because then we go into the programmers world”, "absolutely happy", "I’m happy with the approach”, "yes, happy” etc. And finally for the eleventh question, the participants’ responses were like "interesting”, “more configurable and more useful”, "need more skills”, “more creative and more options”.

Comparative & Usability Evaluation: User Study-1
Figure 6.7: Results of user study-1, prototype-4

Results of the fifth step

As mentioned in the section 6.2.2 that at the end of the study we asked a set of general questions to the participants. Figure 6.8 depicts the results of the important questions, that is how users judge all different tools in terms of flexibility, usefulness and complexity. As it can be noticed that prototype-1 is little flexible, more useful and little complex, whereas prototype-2 is a bit more flexible, highly useful and little complex too. The prototype-3 is considered more flexible than its usefulness and still considered as little complex. However, it is clearly shown that in case of the prototype 4, which is the full mashup environment (i.e., ResEval Mash), the flexibility is largely increased along with its usefulness. The participants found it more useful and flexible as the tool gives freedom to drive their requirements as they want. As compared to the previous three prototypes, the complexity of prototype 4 increased and that is normal. Because most of the participants stated in their remarks that they would need an introductory training to fully utilize its advantages. In the same figure the results on Yahoo! Pipes tool can also be seen (the last chart). As anticipated it has a high complexity, very low usefulness for the users, however, it is fairly flexible as it offers more options to play with, but mainly suitable for programmers only.
Now, you have seen all the different tools (steps), how would you judge them?

Prototype 1 (fixed configuration, and components)

Prototype 2 (changeable configuration, and fixed components)

Prototype 3 (changeable configuration, and components)

Prototype 4 (fully configurable environment, ResEval Mash)

Figure 6.8: Results of user study-1, general results

6.2.5 Evaluation Analysis & Discussion

In the previous section we have presented the results, which reflect the exact representation of the participants’ responses. However, this section provides an analysis of the overall study in which we analyze inter as well as intra-steps variabilities and patterns particularly focusing on the participants’ skills (e.g., technical, non-technical). Mainly, the technical expertise of the participants mentioned in the table 6.1 which shows 19 out of 28 participants have moderate/low technical and 9 have good technical skills. As mentioned earlier that users with good technical expertise are familiar with the programming languages and they were involved in some sort of programming. On the other hand, users with low technical expertise are not programmers. Based on this, we can divide these users into two groups, that is, technical group (i.e., those who know web services, programmings, etc.) and non-technical group (i.e., those who know MS world, Excel etc. but do not know programming).

Regarding the question number 1, as presented in the section 6.2.4, for all the four steps, as described in the section 6.2.2 the difficulty level slightly increased for the non-technical group as compared to the technical one. Figure 6.9 depicts the distribution of the both groups along various difficulty levels. Non-technical group faced difficulties as the process started providing
more flexibility and customization (see non-tech column of step 3 & 4). However, one can notice that even during the step-4 in which the prototype-4 was presented, the majority of the non-technical participants are still within the boundaries that they consider it easy to use, and a few considered it "slightly difficulty". In response to the very next question (i.e., question-2, where participants were asked to provide textual answers about what difficulties they faced) during the step-4, most of the participants demand for more training and tutorial prior to the use of the tool to effectively deal with difficulty.

In figure 6.10, the distribution regarding both, the technical and non-technical groups in terms of process adaptation with respect to increased flexibility for all the four steps is depicted. Clearly for all the steps the "moderately efficient" pattern is consistent that is to some extent increased for the fourth step. The technical group considered the prototype-4 more efficient than the non-technical. Even then the majority of the non-technical participants voted for the efficient option except one participant who considered it slightly inefficient. Figure 6.11 depicts the level of understandability of the both groups. A very low understandability level can be seen in the step number 1 & 2 and a slight increase is detected in the step-3 but still majority could not easily understand how the process execution is performed. However, for the step number four both non-technical and technical groups shown a good understanding.
of the process execution. Obviously technical users have advantage than non-technical users with their technical skills, that is the reason the process execution during the step-4 were easily understandable for the former group.

The accumulated results of the both groups (i.e., technical and non-technical) on question number 7, which represents participants’ comfortability about the given control over the process, are depicted in Figure 6.12. Clearly the demands for more control over the process have emerged till the step number 3. However, one can notice that during the step-4 majority of the participants of both groups feels comfortable with the given control, that is they now feel they can tailor it as they want up-to the level of their expertise. The demand for more control mainly asked by the technical participants, as a small number of participants still want to go beyond the flexibility that the prototype-4 provides, whereas non-technical participants largely consider prototype-4 as a boundary line for them, or otherwise the complexity will increase, responded many non-technical participants.

As an overall, for non-technical participants the difficulty level increased against each richer prototype, however, most of the non-technical participants still found prototype-4 easy to use, as shown in chart (a) in figure 6.13. The demands for more flexible tool emerged from both technical and non-technical groups and the increased flexibility still within the range of users,
Figure 6.13: Various results of non-technical participants for all four prototypes against questions (1, 5, 6, & 7)

as chart (b) depicts in figure 6.13. Relatively low but some participants, during the first three steps, with high technical background did not even understand how the process executed behind the scene, whereas participants having low technical skills did not understand at all as chart (c) depicts in figure 6.13. Largely all types of participants were uncomfortable with the give control over the process for the first three steps, however, a clear satisfaction on such control can be seen in the fourth prototype which was presented in the step-4, especially for non-technical users as depicted in chart (d) in figure 6.13.

Likewise, during the final step in which the generic mashup tool (i.e., Yahoo! Pipes) was presented, the non-technical participants said the tool is too complex for them and that they are not able to understand most of the component and parameter names. However, the technical participants were not scared by the programming-oriented functionalities and terminology of the tool, but they identified a problem with the level of abstraction of the tool. They think the tool could be potentially able to allow the design of a process similar to our reference scenario. However, a complete implementation of the scenario in Pipes would be too complex and time consuming, since the level of abstraction of the provided functionalities is low and not focused on the domain constructs. All participants in this test declared to prefer the domain-specific tools, like ResEval Mash.

Regarding the tool they consider best among the domain-specific ones – considering both
their needs and abilities – the majority of the participants stated that their preference go to the fully functional mashup composition environment of step-4 (i.e., prototype-4). This was motivated by the fact that the tools up to step 3 are too rigid and cannot adapt to different scenarios. Indeed, in the feedbacks after each step a need for more flexibility is evident until step 3, while in step 4 most of the participants declared that they think the flexibility level is enough to make the tool actually useful. Most of the participants also think they are able to cope with the increased complexity.

Moreover, the majority of the participants was overall satisfied with the prototype-4 (i.e., ResEval Mash), they also pointed out some shortcomings. In particular, some participants would have liked some assistance during the design phase, e.g., in the form of contextual information about the components’ functionality and input and output data they use. Other users were instead more concerned about the availability of components providing all the necessary functionality and allowing for high customization (e.g., specification of custom filters, custom impact functions). Another common concern was related to the trust in the final result. The transparency provided by the tool at runtime was useful to establish trust, but there is still room for the improvements.

In summary, the results of this user study show that there is a clear preference for domain-specific tools and that users can indeed imagine to use such tools to build their own processes (only 1 user said that, instead of using tool 4 by himself, he would prefer to ask a technician). Both classes of the participants judge the complexity of the tool-4 as acceptable and think the benefits in terms of time and effort savings are in balance with the effort it takes to learn its proper and effective use.

6.3 Usability Evaluation: User Study - 2

The suggestions, which were gathered during the first evaluation procedure, were addressed and incorporated in the tool. Mainly we focused on the prototype-4 (i.e., ResEval Mash) to increase its usability and intuitiveness, and domain-syntax was also introduced in components’ list, process transparency enhanced by conveying more intuitive execution status. Moreover, an important and much requested feature, which is to give suggestions while composing compositions, is added. Now the tool provides suggestions on the possible connections that a component can make with other components. This feature would surely enhance composition problems that non-technical users could face. Now the tool also shows helping tips regarding various things that we think would help users to effectively adjust to the user experience that the tool provides.

A summative evaluation was conducted to analyze the user experience with improved ResEval Mash. The results reported in this section concentrate on usability, with an emphasis on the role of prior experience on learning. Lessons learned from the first evaluation study, this time we give an introduction of the mashup tool to participants before they use it. Prior experience was differentiated in two categories which are fundamental in our approach to mashup design: domain knowledge and computing skills. Domain knowledge was controlled by selecting all users with expertise in research evaluation, computing skills varied in the sample from people with no programming knowledge at all, to expert programmers.

The study applied a concurrent talk-aloud protocol, a technique requiring users to verbalise
all their thoughts and opinions while performing a set of tasks. Verbalisation capture techniques have been found to be particularly effective when conducting experimental investigations, which provide an opportunity to study communication between products, designers and users [65][66]. The responses given during task completion are considered more representative of the behavior and problems users have during assessment [67] and concurrent talk-aloud protocols have been shown to encourage participants to go into greater detail, to provide more in-depth evaluation, and help pinpoint usability problems and places where their expectations fail to be met [68].

<table>
<thead>
<tr>
<th>Users</th>
<th>Position</th>
<th>Computing Skills</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Administrative People</td>
<td>No Computing Skills</td>
</tr>
<tr>
<td>1</td>
<td>PhD student</td>
<td>No Computing Skills</td>
</tr>
<tr>
<td>1</td>
<td>Professor</td>
<td>No Computing Skills</td>
</tr>
<tr>
<td>3</td>
<td>PhD Students</td>
<td>High Computing Skills</td>
</tr>
<tr>
<td>1</td>
<td>Post-Doc</td>
<td>High Computing Skills</td>
</tr>
<tr>
<td>1</td>
<td>Professor</td>
<td>High Computing Skills</td>
</tr>
</tbody>
</table>

Table 6.2: User categories

6.3.1 Evaluation Procedure

Ten participants covering a broad range of academic and technical expertise were invited to use ResEval Mash. At the beginning of the study, they signed a consent form presenting ResEval Mash as a tool for allowing non-programmers to develop their own computing applications. Then, they were asked to fill in a questionnaire reporting their computing skills and knowledge about research evaluation alongside some basic demographic information (e.g., age and job position). Specifically, participants were asked to estimate their skills with the use of software similar to the Microsoft Office Suite tools, programming languages, flowcharts and mashup tools, on a 4-point scale, ranging from very skilled to no skilled at all. They were also presented with a list of 21 concepts related to research evaluation and asked to indicate for each of them whether they were aware of these and able to understand their meaning, on a 2 point scale (yes vs. no).

After the questionnaire, participants watched a video tutorial (lasting approximately 10 minutes) that instructed them how to operate ResEval Mash. The video introduced the basic functionalities of the tool, quickly explaining the concept of components, configuration parameters, and data compatibility. It then showed how to create a simple mashup of 4 components to display the H-index of the researchers of the Department of Computer Science and Engineering of the University of Trento on a bar chart according to the Microsoft Academics publication source. Finally, the video presented another mashup example used to summarize and reinforce the concepts shown up to this point, where 4 components were connected to visualize on a bar chart the G-index of a researcher (Figure 6.14(a)).

After training, participants were asked to use the system and to perform a series of tasks as described follow:

1. The first task asked people to start from the first composition presented in the video tutorial and to modify the year parameter of the Microsoft Academic component, to select a different department from the Italian Researchers component and finally to replace the
publication source component currently used in the composition with the Google Scholar component.

2. The second task required them to design a composition to compute the participant’s own publication count and visualize it on a chart. The correct solution required linking together 4 components, as highlighted in Figure 6.14.b.

Whilst completing these two tasks, participants were asked to ”talk aloud” regarding their thoughts and actions. This interaction was filmed, as was the interview that followed task completion. The interview focused on interactional difficulties experienced, the evolution of participants’ conceptual understanding over time, and a detailed usability evaluation stressing a feature based assessment reporting which features were considered to be beneficial to interaction, which were understood, and what participants, as users, would like to see in the system. Finally, participants asked to answer a set of questions as listed below and answering metaphor are in braces:

1. What are the main difficulties you encountered? Why? (a multi-line text box was given to get users’ comments)

2. Did you understand the concept of component? (a multi-line text box)

3. Did you encounter difficulty configuration the components? What? (a multi-line text box)

4. How usable did you find the tool? (choice options as: (1) I feel very comfortable and I can use it. (2) I could use it with some more training. (3) It is very unlikely I would be able to use it, even with much more training. (4) I would not be able to use it at all.)

5. Why do you think the tool is useful? (a multi-line text box)
6.3.2 Participants Description

The sample covered a broad range of job positions and technical skills. Half of it was composed of people who reported not being skilled in programming languages, the other half reported being very skilled or good in relation to programming languages. All the participants possessed moderate to no experience with mashup tools. The breakdown of participants according to Position is reported in Table 6.2 and a detailed breakdown of the participants based on their technical skills is depicted in Figure 6.15.

On average as a group, participants had a good understanding of the domain. They possessed experience of 80% of the 21 domain specific conceptual components listed during the pre-interaction assessment. This value ranged from a minimum of 48% to a maximum of 100%. Table 6.2 shows the details of the participants.

6.3.3 Analysis: Usability Evaluation

The video-capture and talk aloud protocols were used to establish strengths and weaknesses in design and conceptual understanding. A subsequent usability assessment was used to identify the difficulties participants reported experiencing and their understanding of the key features of mashup tool interaction.

Overall, the tool was deemed as usable and something with which participants were comfortable. Independently of their level of computing knowledge, all participants were able to accomplish the tasks with minimal or no help at all. The only visible difference reflected a variable level of confidence in task execution. The IT expert users reflected less before per-
forming their actions and appeared to be more confident during the test. Overall, among the users with lower computing skills there was agreement that more training in the use of the tool would be beneficial, whereas this requirement did not emerge from the more skilled sample. It is worth noticing however that the people reporting this need also indicated a lower level of domain knowledge as compared to the other users.

All participants understood the concept of “component” and had no specific issues in terms of configuring or connecting components. However, the post-doc researcher suggested that it might be beneficial for the system to indicate clearly when a proposed connection was inappropriate or illegal by using color to differentiate the states of legality or appropriateness. Another participant suggested the possibility of disabling the illegal components from the selection panel when a component was selected in the composition canvas. Selection of components was highlighted as a potential problem, as identification of the right component required some time to be performed. During the study, this did not appear to be a major problem, as only a selected number of components (N= 8) were tested. Yet, it is reasonable to assume that this problem will increase as the number of available components grows. One participant suggested a search feature, to complement the current menu selection interaction mode. Regarding the question number 4, which asked “how usable did you find the tool?”, 80% of the participants responded as “I feel very comfortable and I can use it”, and 20% said “I could use it with some more training”.

The task requiring tailoring an existing mashup was generally performed better than the task requiring creating a new mashup. In the latter case, a problem emerged with the selection of the first component (i.e., Researcher Input), as several participants selected the Italian Researchers component expecting to be capable to personalise their query there. Saving of configurations was also a source of uncertainty for some participants. The configuration parameters only needed to be filled in by the users and no other action from them was required. This was not clear to the users that in many cases expected an explicit saving action to be performed (e.g., through a “Save configuration” button) and that also expected a feedback to be returned on configuration completion. Several people used the “Close” button after updating the configuration, leading to deletion of the component.

Furthermore, most participants reported some difficulty interacting with the tool due to the physical interaction of double-clicking on the component image in order to open it and been capable to configure its parameters. This constraint was referenced as taking time to learn.

6.4 General Analysis & Discussion

Our studies indicate real potential for the domain-specific mashup approach to allow people with no computing skills to create their own applications. The comparison between the two groups of users highlighted good performance independently of participants computing skills. The request for higher training emerging from a few less expert users appeared to be rather linked to a weaker domain knowledge than to their computing capabilities. Further research could explore the relative role of these two factors by a full factorial experimental study on a larger sample. However, these studies suggested that ResEval Mash is a successful tool appealing both to expert programmers and end-users with no computing skills.

All participants reported a good level of understanding of the basic concepts implemented
in ResEval Mash, although some suggestions for improvement were collected, mainly related to verbal labels used to denote components. Most usability issues evinced from behavioral observations can be easily solved. For instance, the uncertainty experienced by several users with saving the configuration parameters can be counteracted by adding an explicit saving option in the interface of the components. A more serious issue was highlighted as regards the selection of components, which was found to be an error prone and time demanding task. This problem is likely to increase exponentially with the availability of more components, but it can be partially counteracted by a smart advice system decreasing the number of items available for selection based on a comparison between the current application context and previous successful implementations, as presented in [69]. For instance, illegal components could be automatically disabled and the one used most often made salient.

Overall, the studies provided some interesting results and highlighted the important role of user evaluation in the design of interactive systems. A major finding is related to the ease with which our sample (independently of their technical skills) understood that components had to be linked together so that information could flow between different services. This is a well-acknowledged problem evinced in several user studies of EUD tools (e.g., the ServFace Builder, [10]), which surprisingly did not occur at all in the current study. The mismatch can be due to a different level of complexity of the evaluation tasks, but also to an important design difference. Indeed, ResEval Mash only requires users to connect the components as holistic concepts, whereas other tools, such as the ServFace builder required the user to perform complex connections between individual fields of user interfaces.
Chapter 7

Conclusions and Future work

7.1 Overview

Many crucial decisions such as research funds distribution, faculty recruitment, promotions, PhDs selection, award of grants require measuring quality, productivity, and impact of researchers. The research impact is determined based on the scientific research outputs of a researcher, which is measured either through traditional approaches that are based on published papers, citation records, journal prestige (where papers were published) or through advanced bibliometric approaches such as h-index, g-index, ch-index. Over the last few years scientific production has increased to a large extent, researchers are growing in number and also making collaborations, producing and disseminating scientific results. However, primarily the growth is positive, but it takes research organizations, funding bodies under pressure who want to maximize the wider impact of their investment in research. Evaluating research work, determining research impact is a notoriously challenging problem which so far has no well accepted solution. Bibliometrics approaches, which are typically comprised of citation and content analysis methods, have largely well perceived by many communities. However, a general consensus is that the problem remains unsolved, which is mainly due to its multi-dimensionality nature. Moreover, the proliferation of the scientific data sources (e.g., DBLP, Google Scholar, Microsoft Academic etc.), and highly customized bibliometric indices, and locally developed evaluation procedures are the three highly diverse aspects of this field, which make the overall approach highly subjective. Moreover, people involved in such evaluation processes, most of the time, are not IT experts, and not capable of building appropriate software for crawling data sources, automatically parsing relevant information, merging data and computing the needed personalized metrics. Therefore, in order to empower the interested people, we need to design an appropriate and possibly easy-to-use IT platforms, which could make life easier of those domain-experts who do not expert in IT.

The ever increasing number of computer users, especially those non-technical users (as also in our case), who are not computer programmers, use computer applications to fulfill their daily life situational requirements. A large proportion of such users are teachers, doctors, researchers, administrative persons etc. Those are more expert in their domains than in computing skills. In the past, several enabling approaches have been proposed that aimed at facilitating non-technical end-users. Despite many efforts, it is still a challenging endeavor for users to develop applications that support or fulfill their goals. This is because, generally proposed technolo-
gies require expertise in programming languages and their complicated user experience poses difficulties for the users.

End-user development is a way to solve this problem with an aim to empower non-technical end-users in such a way that they can effectively participate in development processes. Specifically through the end-user development a set of techniques, methods and tools collectively enhance user experiences that then can be easily utilized by the non-technical users. To this end, several approaches have been proposed which we have explained in the chapter 2. Mainly, these approaches whose initial goal was to enable non-technical users to design and develop applications with little or no help from developers, we are still in a situation in which these solutions can only be used by specifically trained developers.

On the other hand, the recent emergence of mashup tools has refueled research on end-user development. We see that mashups are simple applications that rather than being developed from scratch by developers, are composed by integrating and reusing available services, data, functionalities or user interfaces. Likewise, mashup tools provide enabling environments for mashups development, ambitiously aim at enabling non-technical users to develop their own on-demand ad hoc applications.

We believe that doing so is even harder than enabling non-technical users to develop applications because developing full applications is simply complex. The reason is the mashup platforms developed so far come with so many functionalities and too many technicalities that are only suitable for programmers. Yet, being amenable to non-technical users is extremely important as the availability of a wider range of online applications, services and data raised the need for situational, short-lifespan applications that cannot be anticipated and developed through traditional software development processes.

However, we believe that it is impractical to design tools that are generic enough to cover a wide range of application domains, powerful enough to enable the specification of non-trivial logic, and simple enough to be actually suitable for non-programmers. Instead, in our view, we need to give up something and that is a generality since reducing expressive power would mean supporting only the development of simple applications, which is useless, while simplicity is our major aim. That means, giving up generality in practice requires narrowing the focus of a design tool to a domain and tailoring the tool’s development paradigm, models, language, and components to the specific needs of that domain only.

7.2 Contributions Summary of the Thesis

This thesis presented a novel approach for an effective end-user development. The target end-users in this context considered non-programmers, less-technical but domain-experts. The approach, which leverages mashups philosophy, can effectively involve end-users in development tasks. Opposite to the existing mashup based solutions, this thesis presented a novel idea of domain-specific mashups, that is, we specifically focused, during the development of the mashup platform, to a well-defined domain. So that, a tool whose design and development is based on a domain can speak the language of the users, a key aspect that existing approaches lack and consequently failed in aiding end-user development.

To this end, we first introduced the notion of domain-specific mashups and described what they are composed of, how they can be developed and how non-programmers can effectively get
benefits. We described which design artifacts are necessary for a domain-specific mashup tool’s development. In this regard, we presented the *mashup meta-model* and the *domain concept model*, the *domain syntax model* and showed how these can be merged into a *domain-specific mashup meta-model*, which provides a consolidate basis and expressive power to a mashup tool whose development follows it.

Based on the above mentioned design and modeling artifacts, the thesis proposed a systematic *approach* and *methodology* (presented in chapter 3), which consists of a number of steps. The proposed methodology can be used for both, the development of an end-user oriented mashup platform and the development of a domain-specific mashup tool. The developed platform initially stays empty (i.e., in terms of domain-specific concepts) and generic (i.e., ready to be tailored for other similar domains) so then it can be tailored to make a domain-specific mashup tool. In this thesis we followed the same strategy, that is, we first developed a mashup platform (presented in chapter 4) whose capabilities are as defined in the *mashup meta-model*, second, we developed ResEval Mash (presented in chapter 5), which is a domain-specific mashup tool for the research evaluation domain. The development of ResEval Mash was also driven by the requirements that are of end-user specific.

Moreover, this thesis introduced a novel and an efficient approach for data-intensive web applications, specifically applications which follow mashups philosophy. The approach is suitable for those applications, which deal with large amounts of data that travel between client and server. We observed that web-service based mashups applications extensively communicate data not only between client server but also between web services too. In this case, particularly, if these web services are deployed on a web server, then data-intensive communication can be reduced to a large extent with the help of proposed approach (presented in chapter 5, section 5.5).

For the evaluation of our proposed approach and to check the usability of our mashup tool, we performed two separate user studies. In the first user study, which was focused on the usability and comparative evaluation aspects, participants were presented four different mashup based prototypes comprising different level of expressive power and flexibility. Moreover, participants were presented Yahoo! Pipes as a generic mashup tool example. The results presented in the chapter 6 clearly show users’ preference to domain-specific mashup approach and most of the participants preferred ResEval Mash tool among the other prototypes. The participants found ResEval Mash offering right balance in terms of expressive power and complexity that a non-programmer can easily deal with. Many useful comments, feedbacks, and requests for improvements were gathered during this study, which were applied to the tool for further improvements.

After incorporating improvements, the second user study was performed, which was mainly focused on the advance usability evaluation aspects of the improved ResEval Mash tool (presented in chapter 6). The results show that the real potential of our domain-specific mashup approach to allow people with no programming skills to create their own applications. Domain experts found very attractive the no-data mapping approach offered by the ResEval Mash, as asking non-programmers or less-technical users to perform complex data-mapping always creates difficulties for them.
Bibliography


