POLITECNICO DI MILANO
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA
DOCTORAL PROGRAMME IN COMPUTER SCIENCE AND ENGINEERING

END-USER DEVELOPMENT OF MASHUPS: MODELS, COMPOSITION PARADIGMS AND TOOLS

Doctoral Dissertation of:
Matteo Picozzi

Supervisor:
Prof. Maristella Matera

Tutor:
Prof. Letizia Tanca

The Chair of the Doctoral Program:
Prof. Carlo Fiorini

2013 – Cycle XXVI

Abstract

WITH With the Web 2.0 revolution, new technologies, new standards and new application models have been introduced in the Web scenario. The Web has become more mature and full of potentialities as a platform for the development of interactive rich applications. The use of client-side scripting languages, the diffusion of Web Services and public APIs, and the always increasing basic skills of laypeople in the development of Web applications shaped up a scenario in which a new class of web applications, the *mashups*, was born. Mashups integrate, at different levels of the application stack, data, functionality and user interfaces from different resources such as Web Services, public APIs or enterprise databases. Mashups emerged in response to the need of users, not necessarily experts of technology, to quickly assemble Web resources to create new Web applications solving their situational needs. One characterizing feature of these applications is that they are very often developed by the *end users*, i.e., people who actually need the final application. To accommodate this practice, which can be fruitful in several situations where the possibility of constructing applications satisfying specific needs is required, in the last years different tools, conceived to offer intuitive composition languages, have been proposed. Unfortunately, most of such tools, after a couple of years of activity, were dismissed.

My PhD thesis aims at investigating and defining a framework that includes models, composition paradigms and tools for the *End-User Development (EUD)* of mashups. Two main reasons for the failure of the mashup tools so far proposed are indeed their incompleteness with respect to the users needs and their difficulty of use. The framework defined in this thesis aims therefore at covering the most salient activities in a mashup life cycle, and proposes a new composition paradigm based on abstractions that try as much as possible to hide technical details. The end users are enabled to integrate data of diverse resources, to create components that could be used in a mashup composition, to generate mashups that can be deployed on different kinds of devices (e.g., mobile devices or multitouch screens). We have also investigated collaboration mechanisms to allow groups of users to share resources and co-create applications. This last feature is particularly fruitful to promote the potential of mashup composition as a paradigm for knowledge sharing and creation.

Sommario

LA rivoluzione del Web 2.0 ha introdotto, tra le altre cose, nuove tecnologie, nuovi standard e nuovi modelli che consentono di sviluppare applicazioni più ricche ed interattive. L'utilizzo massiccio di linguaggi di scripting client-side, la diffusione dei web services e di API pubbliche e la sempre maggiore diffusione di conoscenze basilari di tecnologie per lo sviluppo di applicazioni Web, hanno delineato lo scenario in cui sono nati i mashup.

I mashup sono applicazioni che integrano, a diversi livelli dello stack applicativo, dati, funzionalità e interfacce fornite da risorse differenti, come web service, API pubbliche o database aziendali. I mashup sono emersi per rispondere alle reali esigenze degli utenti, non necessariamente esperti di tecnologia, che avevano bisogno di integrare velocemente risorse Web e creare nuove applicazioni. Una peculiarità dei mashup è che spesso sono sviluppati dalle stesse persone che ne hanno bisogno e che ne faranno uso. Queste caratteristiche possono essere molto vantaggiose laddove vi sia la necessità di sviluppare in modo flessibile applicazioni specifiche. A tal scopo, negli scorsi anni sono stati proposti numerosi tool di composizione che fanno uso di linguaggi intuitivi. Purtroppo, molti di questi tool, dopo qualche anno di attività sono stati dismessi.

Il lavoro illustrato in questa tesi di dottorato si è concentrato sulla definizione di un ecosistema di modelli, paradigmi, tecniche e strumenti che consentano lo sviluppo di mashup da parte degli utenti finali (l'end-user development di mashup). L'inadeguatezza rispetto ai veri bisogni degli utenti e la loro difficoltà di utilizzo sono state alcune delle cause del fallimento

di molti mashup tool. Il nostro ecosistema si pone l'obiettivo di coprire tutte le fasi del ciclo di vita di un mashup e comprende un paradigma di composizione nuovo e semplificato, basato su astrazioni che nascondano il più possibile i dettagli tecnici agli utenti. Le astrazioni che abbiamo definito consentono di integrare dati forniti da diverse sorgenti presenti sul web, di creare componenti che possano essere usati per comporre mashup e di generare mashup che possano essere utilizzati su dispositivi di diverso tipo come dispositivi mobili o schermi multi-touch. Ci siamo anche concentrati su meccanismi di collaborazione che consentano a gruppi di utenti di condividere risorse e co-creare applicazioni. Ciò è particolarmente utile per valorizzare il potenziale della composizione di mashup come paradigma per la creazione e condivisione di conoscenza.

Acknowledgements

The Ph.D. program has been a rich experience. During these three years I had the opportunity to do a lot of new experiences, to put myself on test and to grow professionally and humanly. I met a lot of people belonging to academy and industry coming from all over the world, I had the opportunity to better know professors and researchers I first met when I was a bachelor and master student, I wrote and presented articles, and prepared and corrected exams.
At the end of this journey, I would like to sincerely thank all the people who made this expetience possible. In particular, I would like to thank Maristella for all the offered opportunities, for having believed in me, for the suggestions, the numerous exchange of views and the precious and constant help. I would like to thank also Chiara Francalanci, the Lombardy Region and Beta80 for the fundings of the first two years, and Stefano Ceri, Paolo Cremonesi and Maria Francesca Costabile (and all the group of the University of Bari) for the third year. An acknowledge goes also to Nikolay Mehandjiev (and all his colleagues) who have allowed me to spend an awesome period at the Manchester Business School.
And, last but not least, i would like to thank all my family, Chiara and my friends (included my Ph. D. Student colleagues met in these year) for their closeness and patience.

Ringraziamenti

Il dottorato di ricerca è stato un percorso molto ricco. Durante questi tre anni ho avuto modo di fare numerose esperienze nuove, di essere messo alla prova sotto molteplici punti di vista e di crescere dal punto di vista professionale ed umano. Ho incontrato e conosciuto persone nuove, provenienti da università e da aziende di tutto il mondo, ho avuto modo di approfondire la conoscenza di professori e ricercatori “dall’altra parte” della cattedra, ho scritto e presentato articoli e ho preparato e corretto esami.

A conclusione di questo percorso, vorrei ringraziare sinceramente tutte le persone che hanno reso possibile questa esperienza. In primis, vorrei ringraziare Maristella per tutte le opportunità offertemi, per aver creduto in me, per i consigli, i numerosi confronti ed il prezioso e costante aiuto. Vorrei anche ringraziare Chiara Francalanci, la Regione Lombardia e Beta80 per aver creato le condizioni necessarie per svolgere i primi due anni e Stefano Ceri, Paolo Cremonesi e Maria Francesca Costabile (e tutto il suo gruppo di ricerca dell’Università di Bari) per il terzo anno. Un ringraziamento va anche a Nikolay Mehandjiev (e a tutti i suoi colleghi) per avermi concesso di passare uno splendido periodo presso la Manchester Business School.

Da ultimo ma non per ultimi, i miei ringraziamenti vanno alla mia famiglia, a Chiara e ai miei amici (inclusi i miei colleghi dottorandi conosciuti in questi anni) per la vicinanza e la pazienza.

Contents

1 Introduction **1**

1.1 Rationale 2
1.2 Mashups and User Driven Innovation 3
1.3 UI-centric Composition 5
1.4 Problem Statement 6
1.5 Contributions 6
1.6 Research Methods 9
1.7 Validation 9
1.8 Thesis Outline 10

2 Background and related works **13**

2.1 Mashups 13
2.1.1 Mashup components 15
2.1.2 Mashup types 16
2.1.3 Integration logic 18
2.1.4 Advanced mashups 19
2.2 End-user development and mashup development 20
2.2.1 User-driven innovation 21
2.2.2 Long tail applications 22
2.2.3 End-users involvement in the mashup development scenario 22
2.2.4 Collaboration in mashup development 25
2.3 Mashup Tools 28
2.4 W3C standards 31

3 UI-centric composition paradigm **35**
3.1 Adopting EUD principles 36
3.2 Development process 40
3.2.1 Selection of data services and visual templates 40
3.2.2 Visual mapping 43
3.2.3 Schema generation and execution on multiple devices 45
3.2.4 Component synchronization within UI mashups . . . 45
3.2.5 Composition assistance mechanisms 46
3.2.6 Collaboration 48

4 Models **51**
4.1 Models for UI Component synchronization 51
4.1.1 UI Component synchronization 54
4.1.2 Domain Specific Languages 55
4.2 Visual Integration Model 58
4.2.1 UI-based integration of data sources 62
4.2.2 Domain Specific Language 65
4.2.3 Union and fusion 66
4.3 Models for quality-aware mashup composition 68
4.3.1 Dimensions for quality-aware assisted composition . 70
4.3.2 Community perceived quality 74
4.4 Collaboration model 74

5 PEUDOM **79**
5.1 Overall functionalities and architecture 79
5.2 Composition environments 82
5.2.1 UI Mashup Dashboard 82
5.2.2 Component Editor 88
5.3 Execution environments 91
5.3.1 UI Mashup Engine 91
5.3.2 Multi-device execution engines 92
5.4 Composition Assistance Module 94
5.5 Collaboration modules 97
5.5.1 Sharing resources 97
5.5.2 Architecture 101

6 Validation of the proposed approaches **105**
6.1 Performance evaluation 105
6.1.1 Mashup composition and execution 106
6.1.2 Data-fusion 109
6.1.3 Performances on mobile devices 112

6.2 User studies 113
6.2.1 Overall methodology 114
6.2.2 Study on the mashup composition paradigm 115
6.2.3 Study on the component creation 117
6.2.4 Preliminary study on collaboration mechanisms . . . 119
6.2.5 Preliminary study on domain-specific customization . 120
6.3 Composition assistance mechanisms 125

7 Conclusions and future work 129
7.1 Impact of the developed framework 131
7.2 Future work 132
7.3 Achievements 133

Bibliography 135

CHAPTER 1

Introduction

The Web 2.0 revolution has introduced a plethora of novelties in the Web scenario. Besides the introduction of new technologies enabling the development of more interactive rich applications, a fully distributed and democratic communication platform has emerged, that equally involves developers, information providers, and consumers. The availability of client-side technologies, the diffusion of Web services and public APIs, and the always increasing diffusion of basic skills in the development of Web applications have shaped up a scenario in which the end users evolve from passive consumers of applications to producers of contents and applications. Thus new applications, the *Web mashups*, has emerged, as Web artifacts that are created by integrating, at different levels of the application stack, data, functionality and user interfaces made available by different online resources.

Mashups have emerged in response to the need of users, not necessarily experts of technology, to quickly assemble Web resources to create new Web applications solving their situational needs. To help users "help themselves" and create their own mashups, in the last years a number of tools have been proposed offering intuitive composition languages. The dream was to provide a gate to a "programmable Web", where end users are allowed to construct composite applications that merge content and functions

in a way that satisfies the long tail of their specific needs. However, the approaches proposed so far do not fully accommodate this vision. Unfortunately, most of the proposed tools were dismissed after a short time of activity. The main causes of such failure, as also observed in some user-centric studies [73], are the incompleteness with respect to the users needs and especially the difficulty of use of the adopted composition paradigms and design notations.
This thesis aims at proposing an alternative solution for the mashup development that, by covering the most salient activities in the mashup life cycle and by proposing a composition paradigm that abstracts as much as possible from technical details, is more adequate with respect to the skills of average, non-expert users. This thesis therefore introduces a mashup development framework that is oriented towards the End User Development. It contributes to the definition of adequate abstractions that hide the technology and implementation complexity that characterizes the integration of heterogeneous resources, and that can be adopted by the end users in a kind of "democratic" paradigm for mashup development. Given the fundamental role of user interfaces, as a medium easily understandable by the end users, the proposed approach is characterized by a composition process guided by UI-centric models able to support the WYSIWYG specification of data integration and service orchestration at the user interface level. Model-to-code generative techniques translate models into application schemata that in turn guide at runtime the dynamic instantiation of pervasive, composite applications by means of different lightweight execution environments that can be run on the Web and on mobile devices and that also support collaboration practices for mashup sharing and co-creation.

1.1 Rationale

Past years have seen an evolution in the way information seeking applications are constructed and deliver responses to user's information needs. The emergence of *Web mashups* [37] mirrors indeed the trend, introduced by the Web 2.0 scenario, of allowing the users, not necessarily skilled programmers, to get rapid access to diverse resources offering functionality and data on the Web, and create new value by integrating them into simple, but situated, applications. The proliferation of mobile devices with the capability of running software applications, and the abundant quantity of content and services that can be accessed through such new devices have also favored this evolution and are constantly increasing the desire of the end users to participate in the development of their own artifacts satisfy-

ing their information needs. The ease of accessing and composing services thus offers to the end users the opportunity to accommodate the "long tail" of their specific needs not always addressed by the commonly available applications. Nevertheless, for service providers the opportunity is to capitalize on the variety of their service's usages, as a source of preferences to fine-tune their services and also identify the best innovative uses [37, 51].
The emerging scenario is therefore characterized by a strong potential for user-centric innovation. However, the research on Web mashups has not produced substantial improvements for the end users in the mashup development practices, with the consequence that still only few experts are able to create their mashup applications by programming by hand the service integration [73].
We directly experienced some limits of mashup development practices in the context of our research, when we tried to define composition paradigms and tools through which specific communities of end-users, exploiting the potential of mashup technologies, could construct Web applications responding to the flexibility arising more and more in working environments and personal life situations [4, 21, 23]. Also in the context of user studies, we observed that user-centric development of mashups can be made concrete only through methods and tools adopting adequate abstractions, able to hide technicalities so that technologies become accessible to the end users. This need is even more accentuated in the mobile context, where even expert users need to get acquainted with very specific technologies and languages that vary depending on the target mobile device. There is therefore a need for composition paradigms and enabling tools abstracting from implementation details, but at the same time supporting the creation of full-fledged applications.

1.2 Mashups and User Driven Innovation

Because of the mashup intrinsic value as development practice to let end users to produce new value, mashup composition is in line with the so-called "culture of participation" [39], in which users evolve from passive consumers of applications to active co-creators of new ideas, knowledge, and products. There is indeed a specific driver at the heart of the user participation to the mashup phenomenon: *user innovation*, i.e., the desire and capability of users to develop their own things, to realize their own ideas, and to express their own creativity. According to very recent works published in the literature there is indeed an emerging need to make software systems flexible, i.e., able to support a large variety of tasks to replace fixed applications with elastic, situational environments that can accommodate different needs. New design principles are therefore emerging [60], to promote paradigms in which end users can access contents and functionality, but also flexibly use and manipulate such resources in several situations and across several applications. This new trend promotes a user-driven innovation approach: service providers offer *innovation toolkits* through which users can build their own products [92] starting from reusable services and by means of adequate composition tools. The advantage resulting from such approach is that the iterative experimentation generally needed to develop and test a new product can be entirely carried out by the user, who are enabled to create solutions that closely meet their needs. At the same time, if an experiment turns out to add significant value, the service provider can integrate the user innovation back into its core product [51].

The mashup innovation potential is however not adequately supported by the approaches for mashup composition proposed so far. The research on mashups has been focusing on enabling technologies and standards, with little attention on easing the mashup development process - in many cases mashup creation still involves the manual programming of the service integration. Research teams and industrial players tried to define simplified composition approaches, mostly based on visual notations and lightweight design and execution platforms running on the Web. However, most of such projects failed, due to their inadequacy with respect to some key principles defined and experimentally validated in the End-User Development (EUD) domain [29, 73]. According to the EUD vision, enabling a larger class of users to create their own applications requires the availability of intuitive abstractions, interactive development tools, and a high level of assistance [15,64]. In particular, for a mashup development process to reflect the user-driven innovation potential, the challenge is to let users concentrate on the conception of new ideas, rather than on the technicalities beyond service composition; in other words, users should be enabled to easily access resources responding to personal needs, integrate them to compose new applications, and simply run such applications without worrying about what happens behind the scenes. The prototype-centric and iterative approach that in the last years has characterized the development of modern Web applications ought to be even more accentuated [37]: the composers, who correspond to the mashup end user, just mash-up some services and run the result to check whether it works and responds to their needs. In case of unsatisfactory results, they fixe the problems and are immediately able to run the mashup again.

1.3 UI-centric Composition

Our claim is that EUD of mashups can be promoted by UI-centric approaches, where composers are allowed to focus on user-oriented artifacts, i.e., user interfaces that they are able to use, not programmatic interfaces or data formats. Nevertheless, operating at the UI level must give users the possibility, by means of visual actions on the UI of the composite application, to define integration also at the data and the logic layer. Some projects (e.g., Dynvoker [89], SOA4All [59]) already had a similar intuition, and focused on easing the creation of effective presentations on top of services, to provide a direct channel between the user and the service. However, such approaches focused on the interaction with single services, while they did not investigate the integration of multiple services. The importance of UI as language for development emerges also from some works on model-driven development. For example, in [84] is presented MockAPI, an approach supporting API-first Web application development based on the annotation of user interface mockups to derive running API prototypes that are the starting point for agile development.

There has been a considerable body of research on mashup tools, the so-called *mashup makers*, which provide visual notations for combining services, without requiring users to write code. Among the most prominent platforms, Yahoo!Pipes (http://pipes.yahoo.com) focuses on data integration via RSS or Atom feeds, and offers a data-flow composition language. JackBe Presto (http://jackbe.com) also adopts a pipes-like approach for data mashups, and allows a portal-like aggregation of UI widgets (mashlets). IBM DAMIA [86] offers support to quickly assemble data feeds from the Internet and a variety of enterprise data sources. Mashart [34] focuses on the integration of heterogeneous components, offering a design paradigm through which users create graph-based models representing the way mashup components are integrate at different level of the application stack by coupling events and output parameters of some components with input parameters of other components.

With respect to manual programming, the previous platforms certainly alleviate the mashup composition tasks. However, to some extent they still require an understanding of the integration logic (e.g., data flow, parameter coupling, and composition operator programming). In other words, such tools tried to supply abstractions for non-programmers, but on top of the same model for APIs and Web services conceived for programmers. Even worse, in some cases building a complete Web application equipped with a user interface requires the adoption of additional tools or technologies. A study about users' expectations and usability problems of mashup composition environments [73] showed the evidence of a fundamental issue concerning conceptual understanding of service composition (i.e., end users do not think about connecting services). In our previous work focusing on a platform for the composition of Web mashups we partially solved these problems thanks to the definition of a visual composition paradigm that the users judged easy to use and intuitive [23]. The work presented in this thesis capitalize on these initial results and goes one step forward toward the definition by the end users of pervasive components to access data sources and create UIs for data visualization. We believe that this is a valuable result, considering that one of the pitfalls of current mashup platforms is that even the only registration of new components, without considering their creation from scratch, is a task for expert administrators only.

1.4 Problem Statement

The general research question that guides our work is the following:

> *How can we enable end users to exploit the plethora of resources available on-line and to develop mashups?*

This implies conceiving adequate composition paradigms, based on interactive metaphors, to enable end users, with technological skills or not, to develop applications that meet their situational needs by reusing already available resources with a very low effort.

1.5 Contributions

In light of the previous considerations, we believe that there are numerous opportunities for research and development in the area of *end-user development of mashups*. In a preliminary investigation, we assessed how the integration of *UI components* at the presentation layer [96], if supported by composition paradigms abstracting from technical details [23], is adequate for the end-users to compose Web mashups by synchronizing the behavior of pre-packaged applications. In this thesis we clarify our perspective on the end user development of mashups though UI synchronization paradigms, and also introduce a new composition approach that enables the creation of multi-device mashups. The ease of use, the intuitiveness and the effectiveness of the proposed paradigm are demonstrated by some user studies we conducted to analyze the behavior of real users using our tools. We also show how such composition paradigms can be sustained by UI-centric modeling abstractions enabling the composition of heterogeneous resources and by a reference architecture enabling the fruition of the composed applications on different client devices. More specifically, we introduce the following contributions:

- *User-driven development process.* We argue that the user-driven creation of mashups is more challenging than the provider-driven development of services. Firstly, for non-expert users it is desirable to have a mashup composition paradigm that hides the back-end complexity and simplifies the data, service and UI aggregation process through mechanisms as close as possible to the front-end organization, i.e., the "appearance"of the final application that the user would really experience. Therefore, our approach is strongly characterized by a visual composition paradigm allowing users to integrate contents into unified visualizations and synchronize the behavior of the resulting UI components based on an event-driven logic. The users manipulate exactly the data and the visual elements they will see and interact with in their final applications. A WYSIWIG style indeed allows the users to immediately get realistic examples of the data that will populate the visual elements and the way contents will be fused into unified visualization and synchronized with other resources. In other words, the composition method is based on an "integration-by-example" paradigm, where the users can immediately observe the effect of their composition actions.

- *UI-centric models for resource integration.* We propose a new visual-oriented model for mashup composition, based on associating *data and function renderers*, exposed by online resources, to visual renderers available in some pre-defined *UI templates*. Associating a UI to an integrated data set is one of the most challenging tasks in the construction of mashups. Our UI templates and the consequent *visual integration model* try to alleviate this task, also providing the users with an environment where they program data integration and synchronization of different services by expressing at the interface level examples of what they would like to experience during the execution of the final application. We therefore propose Domain Specific Languages (DSLs) that encapsulates the fundamental constructs of the visual mapping paradigm, yet abstracting from specific visualization styles and execution devices.

- *Multi-device deployment of composite resources.* Our modeling abstractions enable a model-driven process that transforms "examples" of data integration and UI component synchronization, visually defined by users, into applications that can be run on multiple devices. A design environment allows users to visually express data integration queries and automatically generates a composition description, based on our DSLs, that is then exploited by client-side execution engines running on different devices to dynamically integrate data and visually render the final mashup. Independently of our proof of concept implementation, we believe that the modeling abstractions under these mechanisms captures salient, generally valid aspects, that can be reused throughout different UI templates and execution platforms.

- *Lightweight integration of resources.* Our visual integration model supports union and intersection joins to create integrated views of data coming from multiple services and a publish-subscribe, event-driven synchronization of data sources and services spanning multiple domains, both in form of remote APIs (e.g., the most common map-based services or other APIs commonly adopted in Web 2.0 applications), or of context- and content-based services local to mobile devices (e.g., the access to the GPS component or to the info stored in the personal address book). The main challenge is not the definition of a new data or service integration technique; rather our method introduces some abstractions that sustain a paradigm where the local and global integration schemata and publish-subscribe coupling of UI components are dynamically derived from the association operated by users through a visual interactive paradigm. One of our goal is indeed to promote a lightweight integration of data and services that does not require any dedicated integration platform and is based on queries and integration mechanisms at the UI level that the users can intuitively define and easily execute on different client devices, even the mobile ones that are characterized by limited computing capabilities.

- *Collaboration.* We introduce asynchronous and synchronous collaboration techniques in mashup development and execution, enabling users to share resources with other people (e.g., friends, teammates, clients, co-workers) and to co-create artifacts, in line with the Web 2.0 style. In particular, we enable the production of annotations on the available resources and introduce techniques for synchronous co-editing that are based on the propagation of users' actions to different active instances of a shared information space.

- *Proof of concepts.* We illustrate a reference architecture for the execution of Web and mobile mashups that has guided the development of our mashup platform, PEUDOM (Platform for End User Development of Mashups). Through a performance test we show that a lightweight integration of resources is possible also under the limitations posed by the computing environments of client-side applications and mobile devices. Through user studies we also validate the adequateness of the composition paradigm with respect to the abilities of the average end users.

1.6 Research Methods

The methodology applied in this research is composed of the following steps:

1. Analysis of the state of the art and outline of lacks or not considered aspects;
2. Proposal of solutions to fill the lacks, i.e., new models, composition paradigms and technology solutions;
3. Implementation of the proposed solutions;
4. Validation through performance test and user studies;
5. Refinements trough iterations of the previous points.

The research has been organized according to an incremental method. The first part of the research focused on investigating technological solutions for the composition of mashups through the synchronization at the UI level of components, and the provision of abstractions that could support the EUD of mashups [24]. The second part focused on a paradigm and a tool for UI component creation, with particular attention to the execution of the created components within standalone execution environments running on different devices. The last part was related to the collaborative creation of mashups and components. At the end of each part of the research, we conducted experiments in order to evaluate and validate the achieved results and proceeded by solving the emerged issues. The next chapters in this thesis will be devoted to illustrating the main ideas and results emerged from the research, and the experiments conducted to validate them.

1.7 Validation

In order to validate and evaluate our approach, we have conducted performance tests and three user studies focusing on the main platform module, following the incremental flow described in Section 1.6. In particular,

the performance tests focused on the UI synchronization mechanisms, the lightweight integration techniques and on the performances of our approach on mobile devices.
Each part of our research also ended with a user study session to evaluate the effectiveness and efficiency of the achieved solutions with respect to skills and expectations of average users. The first study was about the composition of mashups on the first version of the Mashup Dashboard. The second study was about the component creation and the exporting and use of the created components also on mobile devices. The last study focused on the collaboration mechanisms. All the user-centric studies followed the same four-phase methodology: (i) pre-experience and placement questionnaire; (ii) brief explanation of the experience and demo of the tool; (iii) user experience and observation; (iv) post-experience evaluation questionnaire.
With the user study we want to answer to three main questions in order to understand if the tool is intuitive and easy to use and if end users would use again the tool in the future.

1. Do end users feel comfortable with the tool?

2. Are they able to accomplish some tasks that let them use the most important features and functionalities of the tool?

3. Are they satisfied with the use of the tool and with the work done through it?

During the experiments we observed two different categories of users: expert and non-expert users with respect to mashups and, in general, to Web application development. We measured their performance in terms of task execution time and number and type of errors. Through the post-experiment questionnaire we were able to measure easy of use and user satisfaction.

1.8 Thesis Outline

This thesis is organized as follows:

- *Chapter 2 - Background*: this chapter illustrates the research background that underlies the contributions introduced in this thesis. In particular, it defines mashups and describes their main types, the mashup components and the main integration logics. Thus, it defines EUD and highlights its relationship with mashup development to ends with an overview on mashup tools and technologies for mashups.

- *Chapter 3 - UI-centric composition paradigm*: this chapter presents one of the main contributions of our research. Based on EUD principles, we developed a lightweight UI-centric composition paradigm that supports users in all the phases of the mashup life-cycle. It includes the creation of components, their synchronization within UI mashups, composition assistance mechanisms and collaboration techniques.

- *Chapter 4 - Models*: this chapters illustrates the models that enable the composition paradigm described in the previous chapter, and that constitute the basis for the developed platform. After the definition of the basic abstractions, we describe the models enabling: (i) UI component synchronization; (ii) UI-based integration of data sources; (iii) Quality-aware mashup composition; and (iv) Collaboration through mashups composition.

- *Chapter 5 - PEUDOM*: this chapter describes PEUDOM Platform for the End User Development of Mashup), which we implemented on the basis of the composition paradigm and the models described in the previous chapters. In particular, the functionalities and the architecture of the platform modules are described.

- *Chapter 6 - Validation of the proposed approaches*: this chapter presents the results of performance tests, three user studies and a field study on a specific domain, cultural heritage, our platform was customized to. These studies allowed us to validate the technology feasibility of our contributions and to evaluate our approaches collecting feedbacks by end users.

- *Chapter 7 - Conclusions*: this chapter draws our conclusions, summarizes the main results that have been achieved so far, and outlines the future work of our research.

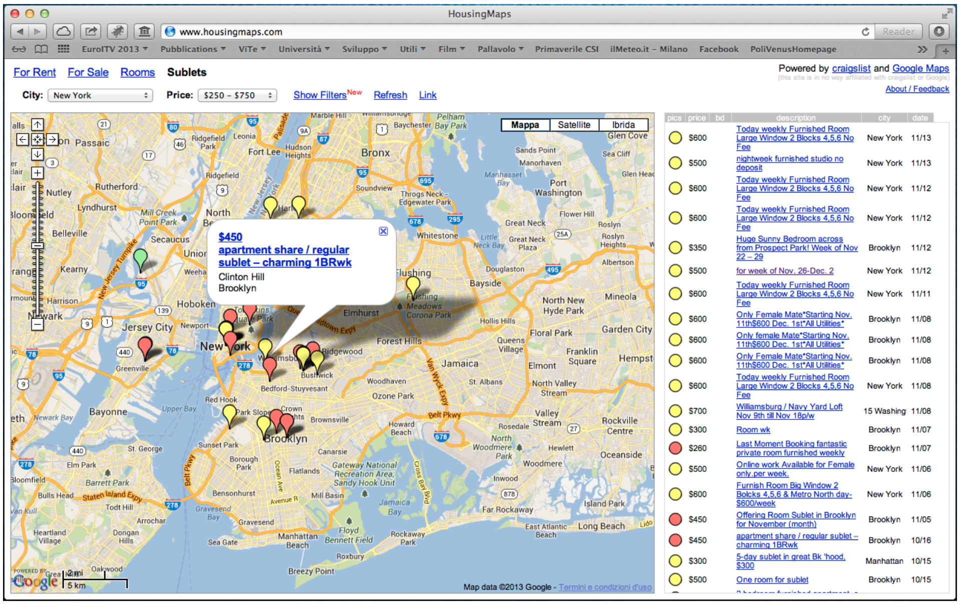
CHAPTER 2

Background and related works

In this chapter we present the research background that underlies the contribution introduced by this thesis. In Section 2.1 mashups are introduced. In particular, they are categorized on the basis of the layers where integration is possible and the most diffused components and composition mechanisms are described. Then, Section 2.2 identifies the elements that need to be considered for investigating a user-driven innovation scenario. Section 2.3 gives an overview of the tools so far proposed in research and industrial contexts. Finally, Section 2.4 describes the most used technologies for mashup development.

2.1 Mashups

Web mashups are “composite” Web applications constructed by integrating ready-to-use functions and contents exposed by public or private services and Web APIs. The mashup composition paradigm was initially exploited in the context of the consumer Web, as a means for rapidly creating applications starting from public programmable APIs. A huge number of new applications were for example built by integrating data sources with Google Maps. Figure 2.1 shows *HousingMaps* (`http://www.housingmaps.`


HousingMaps
www.housingmaps.com
For Rent
For Sale
Rooms
Sublets
City: New York
Price: $250 – $750
Show Filters New
Refresh
Link
Powered by craigslist and Google Maps
(this site is in no way affiliated with craigslist or Google)
About / Feedback
Mappa
Satellite
Ibrida
$450
apartment share / regular sublet – charming 1BRwk
Clinton Hill
Brooklyn
Map data ©2013 Google - Termini e condizioni d'uso

pics	price	bd	description	city	date
●	$600		Today weekly Furnished Room Large Window 2 Blocks 4,5,6 No Fee	New York	11/13
●	$500		nightweek furnished studio no deposit	New York	11/13
●	$600		Today weekly Furnished Room Large Window 2 Blocks 4,5,6 No Fee	New York	11/12
●	$600		Today weekly Furnished Room Large Window 2 Blocks 4,5,6 No Fee	New York	11/12
●	$350		Huge Sunny Bedroom across from Prospect Park! Week of Nov 22 – 29	Brooklyn	11/12
●	$500		for week of Nov. 26-Dec. 2	New York	11/12
●	$600		Today weekly Furnished Room Large Window 2 Blocks 4,5,6 No Fee	New York	11/11
●	$600		Only Female Mate*Starting Nov. 11th$600 Dec. 1st*All Utilities*	Brooklyn	11/08
●	$600		Only female Mate*Starting Nov. 11th$600 Dec. 1st*All Utilities*	Brooklyn	11/08
●	$600		Only Female Mate*Starting Nov. 11th$600 Dec. 1st*All Utilities*	Brooklyn	11/08
●	$600		Today weekly Furnished Room Large Window 2 Blocks 4,5,6 No Fee	New York	11/08
●	$700		Williamsburg / Navy Yard Loft Nov 9th till Nov 18p/w	15 Washing	11/08
●	$300		Room wk	Brooklyn	11/07
🔴	$260		Last Moment Booking fantastic private room furnished weekly	Brooklyn	11/07
●	$500		Online work Available for Female only.per week.	New York	11/06
●	$600		Furnish Room Big Window 2 Bolcks 4,5,6 & Metro North day-$600/week	New York	11/06
🔴	$450		Offering Room Sublet in Brooklyn for November (month)	Brooklyn	11/05
🔴	$450		apartment share / regular sublet – charming 1BRwk	Brooklyn	10/16
●	$300		5-day sublet in great Bk 'hood, $300	Manhattan	10/15
●	$500		One room for sublet	Brooklyn	10/15

Figure 2.1: *An HousingMaps screenshot.*

com) that is considered the first map-based mashup. It is a mashup of Craiglist and GoogleMaps. Craiglist provides advertisement information. Initially, it displayed data just in a list view. Hence, HousingMaps developer integrated the data provided by Craiglist with GoogleMaps, thus achieving a more effective visualization for that kind of data. HousingMaps was published in 2005 by Paul Rademacher who hacked Google Maps in order to use it in his Web page. This happened before Google released the GoogleMap API, when mashup developers were hackers.

Soon the potential of the mashup integration technology emerged also for the development of new applications by end users, even in more critical domains. For example the so-called *enterprise mashups* [54] were proposed as tools for the enterprise users – not necessarily technology-skilled – to compose flexibly their dashboards for process and data analysis by reusing and composing corporate services enabling the access to enterprise information sources, Web resources and open services.

Service composition has been traditionally covered by powerful standards and technologies (such as BPEL and WSCDL), which however can be mastered by IT experts only [85]. Mashups have instead emerged as an alternative solution to service composition that can help realize the dream of a *programmable Web* [71] by non-programmer users. Different from traditional, more mature Web service integration practices, mashup composition emphasizes novel issues, such as the composition of heterogeneous resources based on different service technologies and at different layers of the application stack, i.e., data, business logics and presentation. The *integration at the presentation layer* is the very innovative aspect that mashups brings into the Web technology scenario, which enables the creation, with limited effort and time, of full-fledged Web applications where also the user interface (UI) of the final application can be easily achieved by synchronizing the UIs of different ready-to-use components.

2.1.1 Mashup components

Components are the atomic part of a mashup. We can define *mashup components* as any reusable software module that allow composition – abstracting from their internals and technology specifics. Components could be based on very different technologies, from SOAP Web services to RESTful services to Web APIs, from RSS Feeds to UI widgets. What makes difficult to develop a mashup is indeed this heterogeneity of technologies that are not developed for interoperability.
Components are usually created by wrapping resources. For example, if we consider a Web API like Google Maps, which is one of the most used also because it requires a little effort in order to be included inside a mashup, the API by itself is not enough: a developer needs at least to call the API operations. Some wrappers are very simple because the resource provides by itself contents and visualization capabilities. However, wrappers can also be very complex when the base resource provides only data, and the component has to provide also data visualizations.
As for mashups, also components can be classified in three categories that cover the three layers of the application stack:

- **Data components** provide access to data. Data sources can be either static like RSS of Atom feeds, or dynamic like Web services, services or Web APIs. A resource is considered dynamic if it is possible to formulate queries and, according to the parameters that are dynamically provided, it returns proper results. The use of a data component in a mashup especially aims at integrating the data provided with other data already available.

- **Logic components** provide access to business logic or functionalities. A logic component can provide, for instance, payment (e.g., PayPal) or booking functionality, or can simply have computing logic in the form of reusable algorithms. In case of logic components, one have to set up the interaction with the component: how to provide input, how to use or visualize outputs.

- **User interface components** come with their own UI. Often, they also have their business logic and their data, but sometimes they just provide a visualization for data coming from data or logic components. Google Maps is representative of this type of components. The use of a UI component typically requires a viewport or place-holder area, e.g., an HTML empty `<div>` tag, in which to render the component UI. Thus a UI component brings with itself the presentation layer.

As illustrated in the next section, the type of components in a sense determines also the type of the mashup where the components are integrated.
In order to use a component, a developer has to know how it works, which are its properties, and which operation it is possible to invoke. Hence, before using it, s/he has to "read" the component. What can be read depends on what the provider of the component makes available. It is possible to deserve three main situations:

- **No description**: the component is not provided with any description. This requires the developer to read the component source code (if available), to code examples where the component is used or to follow a reverse engineering approach to understand the component behavior. This is of course cumbersome and error-prone.
- **Human-readable descriptor**: often, a component is accompanied by a human-readable text description. For instance, API documentations fall in this class.
- **Machine-readable documentation**: finally we can also have machine-readable descriptions (typically XML-based documents), which cannot be only read by developers but also by development tools or runtime environments, in order to ease the development and automate execution respectively.

2.1.2 Mashup types

Several kinds of mashups can be identified in the Web scenario, which are characterized by the use of different types of composition and by different integration solutions. Mashups could consist of just a resource inclusion in a Web page, at one of the three tiers of a Web application (i.e., data, logic and presentation), or could be composite applications whose modules, the *components*, are orchestrated and synchronized in a sophisticated manner.

Developing a mashup means especially integrating data, logic or UI. We can identify four different types of mashups that refer to the layer on which the integration is done:

- **Data mashups** integrate data coming from diverse resources (e.g., services, RSS feeds and Web pages). They fetch and manipulate data in order to return them as an unique result set, which is the data mashup output. They may require data mediation, cleaning, splitting, joining or transforming the structure and semantic of the rough data into an unified view. Data mashups are typically published as data resources.

- **Logic mashups** integrates components at the application logic layer. This type of mashup is very similar to service composition but with a big difference: service composition composes homogeneous services while a logic mashup composes heterogeneous components. An example of logic integration could be a mashup in which a component provides a location string that will be converted to a latitude/longitude couple using *Google Geocoder*, and this code will be used to provide coordinates to feed another component. Pure logic mashups are not very diffused. Such integration practices are instead used in hybrid mashups.

- **User Interface (UI) mashups** integrate resources at the presentation layer. They integrate components, which have an independent UI, in a slingle composition and possibly synchronize them. Synchronizing UI components means to allow each component to react to user actions and propagate such actions on another component: a source component triggers an event as consequence of user action, and also other components (at least one) react to this event changing their status. UI mashups are typically published as interactive Web applications.

- **Hybrid mashups** span multiple layers of the application stack. They bring together more than one of the integrations practices described above and the integration can also involve more than one layer. The core challenge of hybrid mashups is the mediation between the three layers, i.e., cross-level integration, since each layer typically handles concerns that are very different from the others. Among the mashup types, hybrid mashups are, of course, the most rich and complete type because they include all the potentialities of the other types. Depending on the prevailing layer where integration takes place, they can result as interactive Web applications or as Web services.

Looking at the mashup ecosystem, e.g., the mashups published on the biggest mashup and API repository on the Web, `programmableweb.com`, it is clear that the majority of mashups have a sort of, although minimal, user interface. Hence, we can observe that UI and hybrid mashups are the most adopted solutions.

2.1.3 Integration logic

One of the most important mashups feature is the integration logic. It refers to the way components are used to form a composite application and how they are enabled to communicate with each other. Integration logic means that a component can influence the state of another component. We can identify four different kinds of logics:

- **UI-based integration** is an aggregation of different UI components merely from a user interface point of view. The components appear in the same page but they are not synchronized together. This kind of logic is anyway an improvement with respect to a navigation through multiple pages, each one devoted to the interaction of one single component, and allows one to have information from different sources in the same one. We can consider Portlets as mashup with a UI-based integration. Consider iGoogle or the most recent Netvibes (`www.netvibes.com`): they enable the presentation in a unique page of contents from different sources, each in its own widget context; however those widgets are not synchronized together.
- **Tightly coupled integration** is an integration logic that can be applied to all kinds of components. Components synchronize among them by invoking operations on other components. This kind of integration logic is commonly used by the developers who are not interested in modifying the mashup or reusing the wrapped components, and thus hard code the integration functionalities in the source code of the component wrappers. In this way, each component is enabled to invoke operations of other components, and to pass parameters to implement data/control flows.
- **Orchestrated integration** applies to all kind of components and consists in a centralized composition logic that orchestrates the execution of the individual components by directly interact with them, e.g., via messages, function calls or events. The other way around, e.g., to communicate with other components, components can interact with

the composition logic; in this case the composition logic mediates communications.

- **Choreographed integration** applies to all types of components that may present active behavior and requires components to comply with a given convention (or contract) that specifies how to communicate with each other. Choreographed integration requires the mashup to provide a communication infrastructure and the setup logic of components, then components know themselves how to communicate. The communication infrastructure is typically a message or an event bus. An event bus knows how to propagate a raised event to the right subscribed operation while a message bus distributes tagged messages to all the message subscribers. This integration logic is mainly applied in mashup platforms where components can be added, reused and easily included in new compositions with a small effort.

These types of integrations are ordered by the flexibility of the related complexity of mashups architecture. The more integration mechanisms are decoupled from the components, the more it is easy to modify the integration and reuse components. In the following chapters we indeed show how it is possible to decouple synchronization of UI components and enable end users to select their synchronization bindings through an end user development tool. The work presented in this thesis greatly aims at flexible mechanisms and architectures.

2.1.4 Advanced mashups

We emphasized so far the internal architecture and integration aspects of a mashup, distinguishing between data, logic, UI, and hybrid mashups in function of where integration occurs in the application stack. Mashups have been originally conceived as Web applications that collect data from different sources and construct integrated views on service data and operations facilitating content fruition, hence they are typically executed on desktop machines, e.g., laptops or desktop PCs. It is also possible to classify mashups with respect to an application-oriented point of view:

- **Multi-user mashups**: the mashups described so far were related to conventional Web applications, i.e., instantiated independently for each individual user of the mashup. Multi-user mashups, instead, are mashups that bring together multiple users in a same instance of a mashup and provide different levels of collaboration or cooperation [68] and can implement business logics as processes [36].

- **Mobile mashups**: In line with the general trend in software/Web engineering and the growing demand coming from a user basis that is increasingly accessing the Web via mobile devices, mobile mashups aim at bringing mashups to mobile devices, such as mobile phones or tablets. For example, the visualization of geo-referenced data on a map keeps to be one of the most adopted classes of mobile apps [13]. The mashup flexibility, laying in the composition of varying services, combined with the increased diffusion and capabilities of mobile devices, is now unveiling the *mobile mashup* paradigm [31, 70, 95]. In some cases, the mashup is available as final application that is executed on the mobile device [21]; in other cases, also the mashup development environment may run on the mobile device, further complicating the development of the mashup environment.
- **Telco mashups**: Bringing together the power of both multi-user and mobile mashups, telco mashups aim at providing people with novel, integrated communication capabilities and features [42]. Rather than aiming at the communication among software components, telco mashups aim at the communication among people, e.g., to further enhance collaboration. The peculiar characteristic of telco mashups is that they may integrate services that make use of telco networks (e.g., to send text messages or establish phone calls), creating a so-called converged network for communications.

In our approach

With our platform, described in Chapter 5, we enable end users to develop hybrid mashups that includes data and UI components, choreographed by a centralized event-bus. Components have a machine readable XML-based descriptor that is used by the platform to manage the couplings defined among them. In our approach we support the development of multi-device mashups through the creation of component schemata that allow the execution of such components on multiple devices, also mobile ones with respect to telco mashups and generated mobile mashups include heterogeneous resources, not only telco-networks services. Finally, we support multi-user mashups through the implementation of sharing and collaboration mechanisms.

2.2 End-user development and mashup development

The End-User Development (EUD) [27, 1, 47] has emerged as a paradigm that promotes the definition of

> “*methods, techniques, and tools to allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact*” [62].

Capitalizing on traditional HCI principles, EUD especially focuses on the capability of systems to offer support during run time to empower users to create their applications very often, blurring the distinction between design time and run-time. A “culture of participation” [39], in which users evolve from passive consumers of applications to active co-creators of new ideas, knowledge, and products, is indeed more and more gaining momentum [92].

Mashups are therefore emerging as a technology for EUD, enabling the creation of innovative solutions in any context where flexibility and task variability become a dominant requirement. In this section we will discuss such a potencial, which is the rationale at the basis of our research.

2.2.1 User-driven innovation

There is a specific driver at the heart of the mashup phenomenon and user participation: *user innovation*, i.e., the desire and capability of users to develop their own things, to realize their own ideas, and to express their own creativity. In a traditional design-build-evaluate cycle, feedback from the user is only collected once a product prototype has been developed. Thus feedback is collected late, and changes to the product, that reflect an improved understanding of customer requirements, are costly. In a user-driven innovation approach, a service provider offers users an innovation toolkit through which users can build their own products [86]. This toolkit provides a constrained interface on the capabilities of the company’s product platform, but this ensures that the new products are properly constructed, adhering to a sort of conservative invention [50].

In general, the idea behind an innovation toolkit is that the iterative experimentation needed to develop a new product can now be entirely carried out by the user. Many users can work in parallel on the solution to a problem, by focusing on their own version of the problem. They can create a solution that closely meets their needs and can more quickly obtain feedback from their development experiments. At the same time, the toolkit provider does not carry the cost of failed experiments. Nonetheless, if an experiment turns out to add significant value, the company can integrate the user innovation back into its core product.

2.2.2 Long tail applications

Another important driver for the mashup growing is the difficulty to elicit requirements from users. It is impossible to know requirements from all the possible users. Sometimes, requirements from different users may be in contrast with each other. These aspects relate to what is generally called the "long tail" of requirements that are not satisfied by common applications.

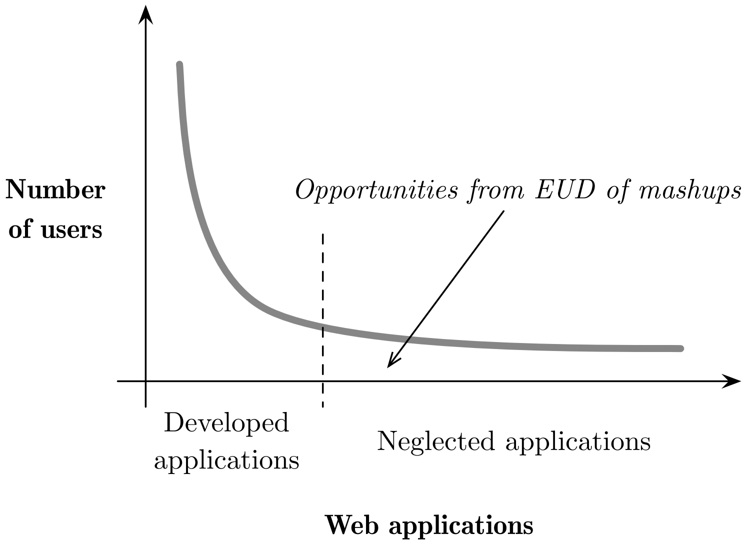


Figure 2.2: *Relationship between the long tail of applications and end-user developed mashups [26].*

As represented in Figure 2.2, the situational applications that meet very specific requirements fall into the long tail of applications usually not implemented in a development scenario "centralized" in an IT department [53]. EUD could be a solution to the specific users' needs, though modest they are, that generally are not being met. A *"distributed" development scenario*, where users are directly involved based on EUD practices, can therefore support a change in the business application paradigm, by accommodating the agility needed to produce quickly ad-hoc applications that address new problems or support new business opportunities [26].

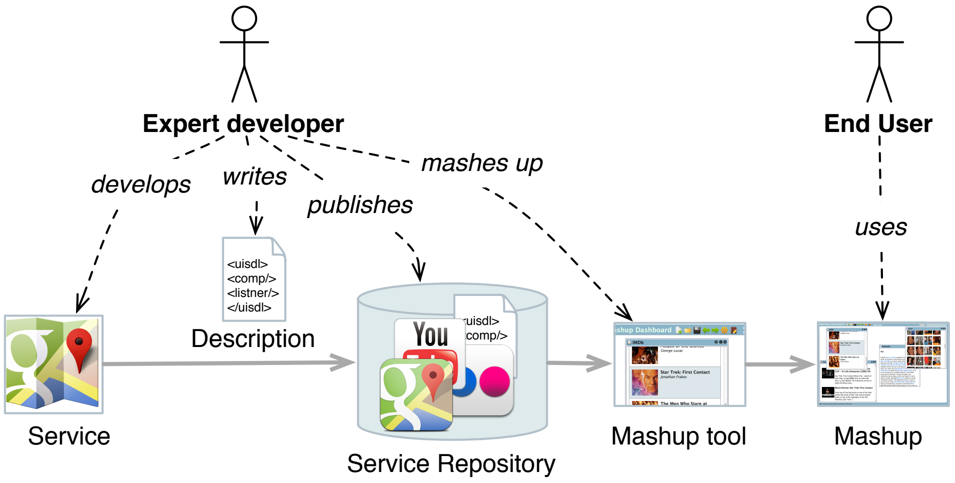
2.2.3 End-users involvement in the mashup development scenario

The way mashups are developed depends on the nature of a mashup. While current *consumer mashups* (for example, all the numerous mashups based on Google Maps) are mainly the results of some hacking activities by expert developers, *enterprise mashups* highlight different development scenarios. According to [33], is it possible to outline in two scenarios the different contributions of users at different skill levels:

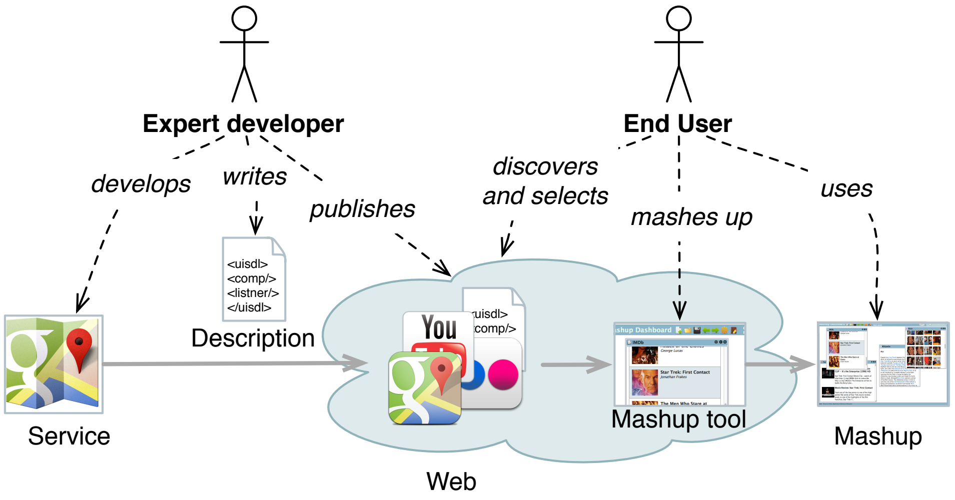
(a) Mashups can be created by expert developers (e.g., implementers of an IT department, service providers or Web developers in general) to deliver applications quickly. End users are not directly involved in the development process; however they benefit from the shorter turn-around time for developing new applications.

(b) Expert developers deploy a tool that lets anyone create their own mashups. This is analogous to how spreadsheets are used in organizations today: end users (e.g., business analysts) can create spreadsheets without involvement from an IT department. In this scenario end users are directly involved in the creation of their own mashups.

Figure 2.3 illustrates these two scenarios. The two (extreme) corresponding solutions differ in terms of the heterogeneity of the services that can be combined, the diversity of user needs that can be met, and the level of sophistication of either the user or the tools that support their work. A tool for the creation of mashups (scenario (b), Figure 2.3b) will, initially, be the most challenging scenario to implement. However, it also provides the biggest pay-off. Using the tool, users can combine services and data to create their own mashups. The tool constrains what users can do and, hence, ensures the composability of mashup components. In the sense of the earlier discussion on user innovation [87, 93], such a tool provides a toolkit that enables users to create their own applications. However, users are not limited in terms of the types of applications they can build: this scenario, therefore, supports the greatest diversity of user needs.
Another distinction between the two scenarios is the degree of control over the quality of mashups being created. In scenario (a)) (Figure 2.3a), the IT department (or the developer in general) fully controls what kind of mashup is being developed. Thus, the IT department ensures the quality of those mashups. However, not all mashups have strict requirements in terms of security, performance, or reliability; they may only be used for a specific purpose, and a complex solution developed by the IT department would also be too costly. In scenario (b)), the IT department selects which components can be mashed up and provides an environment for safely executing the mashups. Users can create mashups from those components to meet needs unanticipated or not served by the IT department. Such mashups may subsequently serve as prototypes for hardened applications developed



(a) *Mashups are developed by IT experts and end users can only use them.*



(b) *Mashups are developed directly by end uesers.*

Figure 2.3: *Mashup development scenarios.*

by the IT department, should there be a need for the mashup to be exposed to many users within the enterprise, or if the mashup has to be offered to outside users.

In von Hippel's terms [93], mashups "democratize" innovation, allowing end-users to meet their own needs, which a central IT department or, more in general, a service provider cannot always address. Their use also shortens the time by which users obtain the desired functionality. One use of mashups is indeed to prototype a solution to a problem faced by a specific user, and later generalize it to a larger user community. Mashup development is therefore similar to open source development (which was the source for von Hippel's metaphor) in two ways: the contributors to an open source project are also users of the software it produces; and open source projects provide a mechanism whereby contributors can progress from passive users to providers of feedback and feature requests and to code contributors. Similarly, mashup developers are often also users of thir mashups (in scenario (b)); and not all the users of mashups need to be developers, but they can contribute to mashup development by providing feedback and feature requests.

2.2.4 Collaboration in mashup development

Another important aspect in the EUD of mashups is the collaboration among users. Collaboration in mashup-based development can be beneficial in collective intelligence scenarios [43], where teams of people co-create knowledge by sharing integrated information spaces with professional peers, in meta-design environments [40], where end users shape up their tools in collaboration with expert developers, or in scenarios where people, not able to develop by themselves their own applications, ask for help and advice from experts within reference communities in a kind of crowdsourced Web Engineering [77].

Consider the case of enterprise mashups: if a user develops a mashup that is useful for her/his work and, by developing this application, s/he is filling an IT department lack, it would be beneficial for the enterprise if this application is shared to all the colleagues that can be interested in it. It is also possible that the user who had the idea to develop such an application does not have all the domain knowledge to develop it. If s/he can develop the mashup with the help of her/his colleagues, this would improve the value of the application itself.

This is the concept behind collective intelligence, which is recognized as an important resource especially in the enterprises [9]. However, Web 2.0

collaboration paradigms are also emerging; a notable example of collaborative Web 2.0 platform is Google Drive, which enables users to share and collaborate in document editing. It is possible to find other examples of the diffusion of collaborative mechanisms also among the mashup tools, e.g., Yahoo! Pipes provides a community for mashups developers where it is possible to share data mashups. Collective Intelligence has its foundations in the Computer-Supported Cooperative Work (CSCW) discipline.
For several years CSCW has been "addressing how user coordination to perform collaborative activities can be supported by means of computer systems" [27, 88]. One adopted way of conceptualizing CSCW systems is to consider the context of a system's use. With this respect, one useful conceptualization tool is the *Groupware Matrix* [55], which is reported in Table 2.1. The matrix has two dimensions: a *spatial dimension*, to distinguish between co-located or remote collaboration, and a *temporal dimension*, to distinguish between synchronous or asynchronous communication. Each matrix element identifies a different collaboration work context. For example, according to this matrix, mashup-based collaboration, for the nature of the medium on which mashups are executed (Web browser or mobile execution modules), can be characterized as a remote collaboration. Also, based on the time distribution of the work, different approaches for communication can be used. As we will show later in this thesis, version control and annotations can be used to enable asynchronous communication, while instant communication, like a live chat and live editing, can be used to support the synchronous co-creation of integrated workspaces by different users.

	same time (synchronous)	**different time** (asynchronous)
same place (co-located)	**Face to face interactions** *decision rooms, single display group-ware, shared table, wall display, room-ware, ...*	**Continuous task** *team rooms, large public display, shift work group-ware, project management, ...*
different place (remote)	**Remote interactions** *video conferencing, instance messaging, chats/NUDs/visrtual worlds, shared screen, multi-user editors, ...*	**Communication + coordination** *email, bulletin boards, blogs, asynchronous conferencing group calendars, work-flow, version control, annotations, ...*

Table 2.1: *The CSCW Matrix (a.k.a. Groupware Matrix)*

Two *coordination mechanisms* in groupware are also highlighted in [7]:

sharing of resources and *communication* among the collaborating actors. The main issues to be considered for sharing are the *ownership* of the shared resources, the *trust* in the other people involved, hence the *privacy levels* and the *users' roles*. Moreover, in the case of a remote synchronous collaboration, e.g., in co-editing scenarios, each actor must be aware of who are the other users that are online in that moment and what they are doing.

While collaboration is a mature research field in the CSCW community, collaboration in the creation of Web artifacts, and especially Web mashups, is still scarcely explored. The tools so far proposed ease mashup development to unskilled users [3, 74] by offering intuitive visual notations substituting the programming of the component integration logic. Such tools reached the goal of enabling the end user development of mashups; however most of them offer paradigms for the creation of single-user applications, while they do not support the co-creation of shared information workspaces.

The need for mechanisms to let user collaborate for the creation of Web artifacts is however also evident from some recent works proposed in literature. In the context of Web-based collaborative learning, the notion of *Web Space Configuration* is introduced in [67] as a basic container for instantiating W3C Widgets. The proposed approach is characterized by the independence of the created Widgets composition from the runtime environment. Such independence is thus exploited to support portability of the created applications, but also its sharing through broadcasting and co-editing. More specifically, broadcasting and co-editing are achieved by establishing a long-lasting connection by the owner of a Web space, who invites other users to join and see the Web space (broadcasting) and to apply changes (co-editing).

In [78] the authors propose a crowdsourcing paradigm where user participation is adopted as a solution to responsive design in Web application development, trying to collectively solve problems related to the adaptation of the Web apps to different screen devices. According to this approach, system developers can provide an interface where adaptive features can evolve at runtime with the help of users who can refine the adaptations to better match their peculiar usage context. This paradigm opens Web development towards the social dimension. However, the aim is limited to let the users adjust the presentation of their Web apps to best fit their current device, while it neglects collaboration and coordination of different stakeholders. Additionally, if focuses exclusively on presentation adaptation, while it does not cover at all modifications of the application content.

In [45] the authors propose a *generic awareness infrastructure* that aims at providing basic awareness services that are reusable throughout different platforms. Awareness support is anchored at a standardized layer thus providing an application-agnostic solution. Different collaborating clients therefore include a component, the *generic awareness adapter*, that embeds awareness widgets and is devoted to managing awareness mechanisms through the propagation (from the client to the server and vice-versa) of awareness information. The approach is very interesting, especially because of its portability across different platforms, and its intrinsic extensibility, being it based on the integration of widgets managing the different collaboration aspects.

In our approach

Users can collaborate along all the dimensions highlighted by the CSCW matrix. In particular, we adopted synchronous and asynchronous mechanisms (described in Section 5.5). For asynchronous collaboration, we adopted annotations and activity logs. With annotations users can "augment" content or components, in order to share their thoughts with the collaborators. For synchronous cooperation instant messaging tool and presence and action awareness techniques are adopted. We use a chat tool in order to provide an instant communication to users while presence and awareness techniques help users to understand who the other on-line users are and what they are doing, e.g., the highlighting of a component on which another user is working on and the list of on-line users.

2.3 Mashup Tools

In the literature, a lot of tools have been proposed to allow the development of diverse kinds of mashups. However, most of these tools were dismissed after few years of activity or were research projects that have never been really adopted.
In Table 2.2 the tools described below and our platform PEUDOM are classified with respect to different classification dimensions discussed in this chapter.
Some projects have focused on easing the creation of effective presentations on top of Web services, to provide a direct channel between the user and the service. *SOA4All* [58] is a tool that facilitates the creation of service infrastructures and increases the interoperability between large numbers of distributed and heterogeneous functionalities on the Web. *SOA4All* concentrates on the establishment of an instance of a service delivery platform that is optimized and tailored to the needs of Web services. *Serv-Face* [79] adopts a model-driven development approach that applies service

Dimensions		**PEUDOM**	**SOA4ALL**	**ServFace**	**Yahoo! Pipes**	**MashArt**	**Dapper**	**JackBe Presto**	**Damia**	**Netvibes**	**EzWeb**
Mashup types	Data mashups	✓	✓	✓	✓	✓		✓	✓		
	Logic mashups		✓	✓	✓	✓		✓			
	UI mashups	✓			✓	✓	✓			✓	✓
Component types	Data components	✓	✓	✓	✓	✓	✓	✓			
	Logic components		✓	✓	✓	✓		✓			
	UI components	✓			✓	✓	✓			✓	✓
Component description	No description						✓				
	Human-readable										
	Machine-readable	✓	✓	✓	✓	✓		✓		✓	✓
Integration logic	UI-based									✓	
	Tightly coupled										
	Orchestrated		✓	✓	✓			✓			
	Choreographed	✓				✓			✓		✓
Advanced techniques	Multi-user mashup	✓				✓			✓		✓
	Mobile mashup	✓						✓			
	Telco mashup										

Table 2.2: *Placement of out tool with respect to all the other main mashup tools*

annotations to a visual authoring of service-based interactive applications. *Dynvoker* [90] is a tool that allows users to generate forms dynamically to invoke services in many scenarios, including rapid service testing and dynamic inclusion of services as plugins into applications. Such approaches do not allow the composition of multiple services into an integrated application.

There is also a considerable body of research on mashup tools, the so-called *mashup makers*, which provide graphical user interfaces for combining mashup services. *Yahoo! Pipes* [1] is one of the most popular mashup tools. Pipes is a wired environment in which mashup feeds are accessed and "piped" through user-selected functionality (combine, filter, sort, split, count, truncate and so on) to process the feed. It provides also a community for building, sharing, rating and modifying mashups. *MashArt* [35] is a Web tool that provide universal composition as a service in form of an easy-to-use graphical development tool equipped with an environment for fast deployment and execution of composite Web applications though the pipe metaphor.

With respect to manual programming, such platforms certainly alleviate the mashup composition tasks. However, to some extent they still require an understanding of the integration logic (e.g., data flow, parameter coupling, and composition operator programming). In some cases, building a complete Web application equipped with a user interface requires the adoption of additional tools or technologies. Even when they offer an easy composition paradigm, they do not guide at all the composition process. A study about users' expectations and usability problems of a composition environment for the the *ServFace* tool provides evidence of a fundamental issue concerning conceptual understanding of service composition (i.e., end users do not think about connecting services) [75].

Dapper [38] is a tool powered by *Yahoo!* that allows one to collect information from Web pages or feeds, and create a *Dapp*. A *Dapp* is a container of content that could be of different formats, e.g., XML, CSV, JSON, HTML but also Google Gadget, Google Maps and iCal. *Dapper* therefore produces customized contents using Web pages data. Our tools also allow users to combine such data with other resources. We can thus consider *Dapper* as useful addition to the *Component Editor* (see Chapter 5); we are indeed planning to include Dapps into our *Mashup Dashboard* and allow users to use them as mashup components.

Netvibes is a customizable Web portal. Users can select widgets or portlets from a list that are used to compose the personal information spaces. Netvibes is similar to the dismissed iGoogle portal. In these kind of applications it is possible to create UI mashups by including widgets that however cannot communicate. It is in fact not possible to synchronize or to integrate data belonging to different widgets.

JackBe Presto and *IBM Damia* are enterprise-oriented tools which offer support to integration, reporting and visualization of enterprise data. *JabkBe Presto* [52] is a real-time operational intelligence software with real-time Business Intelligence (BI) analytics. It allows to combine data from data warehouses, feeds, social medias, existing BI systems and Excel spreadsheets, and to create data visualization in real time. Its tools foster real-time collaboration through an interactive dashboard that can be executed also on mobile devices. Our approach also allows to execute the created mashups on mobile devices but, through our platform-independent domain specific languages, we generate visual integration descriptors that can be interpreted by native web applications. *IBM Damia* [87] is a data integration platform that allows business users to quickly and easily create data mashups that combine data from desktop, Web, and traditional IT sources into feeds that can be consumed by AJAX and other types of Web applications. In the enterprise context, such tools are very useful to integrate data sources, although it is different to create a full-fledged application also equipped with

a presentation layer.
EzWeb [66] is a mashup platform that empowers users to create and share instant composite applications. It enable users to create a UI mashup including and synchronizing UI components. This is possible thanks to an event-driven paradigm that allows one to couple events and operations. With respect to EzWeb we provide support to an extended mashups lifecycle, in particular for component creation, execution on multiple devices and real-time collaboration.

2.4 W3C standards

> *The W3C Web Applications (WebApps) Working Group is chartered to develop specifications for webapps, including standard APIs for client-side development, and a packaging format for installable webapps*[1].

In the last years they have published several documents and working drafts proposing new specifications. In particular, their publications have been clustered in *API specifications*, *Web Component specifications* and *Widget specifications*. While Widget specifications were officially proposed for the first time in 2006 and have become W3C recommendations in November 2012, Web Components and API specifications are still in an embryonic state.

- *Widgets* are defined as:

  > *full-fledged client-side applications that are authored using Web standards such as HTML and packaged for distribution.* [49]

  Widget packages contain all the needed files, including a configuration file, icons and the default start file. They are typically installed client-side on different devices where they run as stand-alone applications, but they can also be embedded into Web pages and run in a Web browser. In order to deploy Widgets, application servers like Apache Wookie[2] have been developed. Such servers allow one to upload and deploy Widgets in a Web application.

- *Web Components* [48] consist essentially of five pieces: *(i) templates*, which are markup descriptions of the component UI; *(ii) decorators*,

[1]`http://www.w3.org/2008/webapps/`
[2]`http://wookie.apache.org`

which apply templates based on CSS selectors to affect rich, visual and behavioral changes to documents; *(iii) custom elements*, which can encapsulate component state and provide script interfaces; *(iv) Shadow DOM*, which is a DOM subtree that can be associated to an element, but does not appear as child node of the element; and *(v) imports*, which allow loading external files. All those pieces, if used together, allow one to create rich and interactive self-contained components.

- *API Specifications* [47] consist in a set of documents – mainly working drafts and few recommendations – which include specifications on, aspects such as Cross-Origin Resource Sharing to control the security policies of browsers, Server-Sent Events to manage server-side events, Web Messaging and Web Sockets API to manage the communication among applications, Web Storage to manage local variable, and Web Workers to manage computational intensive tasks without interrupting the user interface. All these APIs enrich the features of the webapps by introducing new interfaces that are or will be soon implemented by the future versions of the main browsers or by client-side libraries.

Although Web Component and API Specifications are not mature yet, in the last months they have been more and more diffused. JavaScript libraries, such as, Google's *Polymer*[3] and Mozilla's *Bricks*[4], have been developed in order to exploit the new Web Components and APIs features. Since these specifications are new, they are not natively supported by the modern browsers. For this reason, on the Web many *polyfills* were published, which are client-side libraries that fills the lack of both modern and obsolete browsers in implementing the new HTML5 features (Web Components and the new APIs included) and enables to the adoption of these new technologies also in not up-to-date browsers.
The webapps of the future will indeed take advantage of Widgets, APIs and Web Components. In particular, also mashups will be involved in this evolution. With respect to these emerging new standards it is important to understand how mashups are related to them. First, as better explained in the following chapters, in our approach we use UI components that are packed with a structure similar to the W3C Widgets packages described in [49]. Our component packages contain an HTML template, a set of scripts, the component icon, the component stylesheets and the component libraries. Further, the logical structure of our components is similar to the W3C Web

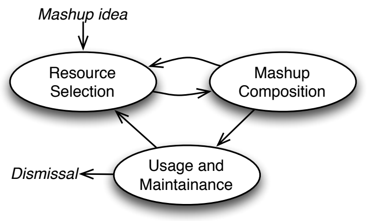
[3]http://www.polymer-project.org
[4]http://mozilla.github.io/brick/

Component structure because we have a base component template that is modified by scripts according to the user interaction like W3C Web Components templates. In our scripts we implement an object for each component which encapsulate all the component functions that are invokable also from other components or objects in general. Hence we implemented the concept of the Web Components custom elements and shadow DOM. Another aspect is the UI synchronization: Web Components are more similar to our approach (see Chapter 4) because they easily support the event-based synchronization, although they do not have an explicit exposition of events and operations. Instead, Widgets have an important limitation: they do not allow the *inter-widget communication*. This means that it is impossible to natively support Widget synchronization. In [32] the authors propose a method for the inter-widget communication in UI mashups. They infer the synchronization among Widgets by monitoring the user interaction on the UI. For example, if a user searches for the same keyword with two different Widgets, the system detects this behavior and infers that, when s/he performs a search with the first one, s/he wants to search for the same keyword also with the other Widget. Thus the system creates a synchronization rule that will be applied whenever this action occurs again.
In the future it will be interesting to see how the World Wide Web will evolve thanks to these standards and, hopefully, thanks also to mashup applications.

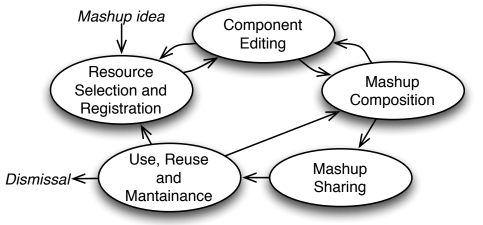
CHAPTER 3

UI-centric composition paradigm

Our mashup development paradigm allows users to compose different resources at different levels of granularity, always operating on the UI of the final interactive artifact to be created. Different levels of granularity lead to an iterative cycle of activities that extend the life cycle of mashups (Figure 3.1), as typically supported by the majority of mash-maker tool [37], especially giving to the users the possibility of creating and editing the components through which successively they can compose more suitable mashups and enabling users to share resources (i.e., registered services, components or mashup compositions) that can be reused by others.
As represented in Figure 3.1b, the user can integrate content retrieved through remote data services and build visualizations where multiple data sets are integrated. The result of this composition is what we call a *UI component*, i.e., widgets that: *i)* provide a user interface (UI) on top of the integrated result set, *ii)* can be executed as a self-contained application on multiple devices, and *iii)* expose an event-driven logic that make them available to be reused and synchronized with other components in a larger mashup. The creation of UI components, especially through a visual paradigm adequate for unskilled users, is an activity not supported by other tools, which generally appear as "closed" environments, where the addition of new compo-



(a) *Mashup life cycle as supported by the majority of mash-maker tools [37].*



(b) *Mashup life cycle as supported by PEUDOM.*

Figure 3.1: *Mashup life cycles.*

nents is a task for more expert developers. Moreover, our approach consider also collaboration among users that is one of the most important aspects in the Web 2.0 scenario.
In the rest of this chapter, we first describe the main EUD principles and the barriers in the adoption of tools by the end users in the mashup development domain. We show how our UI-centric composition paradigm facilitate the mashup development and illustrate our development process, that is based on the models described in Chapter 4, and supported by the interactive composition environments offered by the PEUDOM platform, further described in Chapter 5.

3.1 Adopting EUD principles

In order to identify the elements that are necessary to achieve viable solutions for a user-driven composition scenario, we reviewed the findings of some experimental studies on the definition and validation of EUD principles [44, 57, 65, 94], and re-interpreted such principles taking into account the peculiarity of mashup composition. Also based on the experience that we gained in the last years while building composition tools and observing end users using such tools [4, 21, 23], we identified some complexity factors that also occur in the mashup composition activity, as supported by the tools so far proposed. In particular, with reference to the barriers for EUD identified in [57], we observed that in mashup composition typical difficulties relate to:

- **Design barriers**. They relate to mapping the concept of the desired application to an abstract description of a mashup composition. Not always end users are able to make a distinction between application design and execution; they find it difficult to compose a mashup using abstract notations that are far from the concept of the final application [73]. With this respect, for example, wired notations, adopted by several mashup tools to create composition graphs specifying data and/or event propagation, might not be adequate for the average end user [81].

- **Selection and use barriers**. It is complicated for a non-expert composer to guess how a component can be used, which features of its programmatic interface can be adopted and which effect each service may have on the overall composition. Abstract representations of the service, for example in terms of input/output parameters, that are far from the representation of the data or the UI layout that services are able to offer do not help users understand how the service can be exploited and integrated in a mashup composition.
- **Coordination barriers**. They relate to identifying the way components can be combined and integrated. Assuming that the users is able to identify the services that can allow them to reach their goal, several complexity factors then emerge when they have to define the integration logic to get a coordinated behavior of the identified components. Several tools still require the users to deal with service linking, e.g., through parameter coupling or through the definition of process graphs reflecting the execution flow. However, these aspects, that are strictly related to programming, are not always understood by the ordinary end users [74, 81].
- **Understanding barriers**. They refer to the difficulty of understanding the external behavior of the composition, due to the difficulty in evaluating the behavior of the final application against the composer expectations. Especially when specific design notations are used to compose the application, users cannot easily imagine how the final application will look like.

To overcome the previous problems, we consider the following requirements, already studied and validated in different studies ([15,44,57,63,94], to mention the most relevant), as fundamental ingredients for the successful porting of mashups technology to EUD:

- **Closeness of mapping**. In order to help users understand the features provided by the available services, the effect that each service may have on the overall composition, the way it can be integrated with other services, it is important to come up with representations of services that abstract from technical details about the programmatic interface or communication protocol while increasing role expressiveness [44]. One solution that we adopted is to let users manipulate, e.g., add, remove, or modify, *visual objects*. Also, the addition of one such object into the composition workspace immediately produces the visualization of the UI and of the data provided by the corresponding component. Therefore, users can operate at the level of service visualization properties rather than being required to configure technical details of services invocation and of the integration logic. One of the experiments that we performed with real users [19] allowed us to assess that this visual approach, focusing on the immediate visualization of services, affect positively the user-perceived *usefulness* of mashup composition and the *ease-of-use* of the proposed tool.

- **Progressive evaluation**. In order to further enhance the users' perception of, and the control on the effects that services and composition actions have on the application under-construction, for example on its look&feel, on the retrieved data set and the way data and component behaviors are synchronized in the overall composition, it is important to provide immediate feedback on "how the user is doing" [44]. We therefore propose an *immediate execution paradigm*, where each single composition action is observable through the immediate execution of the modified application. In other words, we do not distinguish between design and execution: since the very first composition action, the user is able to see a running application, and to observe incrementally the effect of any other subsequent composition action. This choice also avoids the so-called *premature commitment* [44], since the user is not forced to make decisions without being able to observe and evaluate the effect of such decisions. One experiment conducted to validate one tool prototype with real users [19] highlighted that this choice positively affect *self-efficacy*, i.e., the users' perception about their capability to face challenges competently, because the users strongly feel in control of the composition process.

- **Composition support**. In order to further smooth EUD barriers, it is also important to aid those users that do not have sufficient development knowledge. Composition can be assisted or guided in multiple ways, for instance by providing default system-driven service couplings, when possible. Users find helpful any kind of hints that the system is able to provide during the mashup development [74]. However, it is also important to give the users the right level of control to modify the automatically provided solutions, and offer them recommendations for further services and composition patterns that can fit the current composition. In this thesis as well as also discuss some techniques for the generation of quality-aware recommendations [22].

- **Abstraction gradient**. Another fundamental ingredient is to accommodate different users skills and attitudes, and also varying composition contexts by providing the users with different abstraction levels [44], so ensuring a “gentle slope of difficulty” [63]. To meet this requirement, we identified different “composition capabilities”, reflecting different composition granularities. Therefore, our approach lets the users: *i)* embed pre-packaged, ready-to-use components into a workspace, possibly taking advantage of pre-configured rules for component coupling, without the need of defining any additional setting for components execution and integration [23]; *ii)* define additional integration policies to synchronize the UIs of the different components [96]; *iii)* build new components by visually programming the integration of different data sources and the visualization of the integrated data set through pre-defined user interface skeletons.

- **Domain-specific focus**. Domain specificity means to use specific terminology and customizing the interaction on the target users needs, as outlined by Casati in [28]. The composition paradigm illustrated in the sequel of this chapter is not tied to any specific domain. Our approach is indeed general purpose and open in its nature, being it conceived for the integration of heterogeneous services, based on different visual templates and user interfaces supporting mashup composition. However, this openness facilitates the customization of the proposed platform with respect to the characteristics and needs of specific communities of end users. Customization, for example, occurs by selecting and registering into the platform services and data sources (public or private) that can provide contents able to fulfill relevant user information needs. Domain specificity is thus an orthogonal characteristic of our approach that responds to the users’ need of understanding the potentialities of composition tools and being comfortable with the tools they use as a key success factor for the diffusion and the real adoption of composition tools in general. In order to allow users to understand the possibilities offered by the mashup platform and to make sense of the services and components that are available for composition, it is important to restrict the platform to a well-defined domain the user is comfortable with. That is, we need to be able to develop a general platform which can be however easily customized to speak the language of the target users, both in terms of functionalities and terminology known to the user. To ensure an adequate trade off between specialization and offered functionality, we believe that general platforms should be amenable to the customization of the offered components and of the provided composition features. Our UI centric paradigm enables a decoupling of the presentation and interaction layer, which enables the adoption of different UIs on top of the business logic layer of our tool, facilitating the customization of tools with respect to the EUD requirements. This is what we experienced when we specialized our generic platform to support the activities of different communities of end-users [4, 21, 23].

3.2 Development process

Given the ingredients defined above, Figure 3.2 outlines the whole life-cycle for mashup development supported by our approach, from the creation of single components, to the execution of such components on different devices and/or their integration within more complex mashups.

3.2.1 Selection of data services and visual templates

The starting point is the discovery and the registration into the platform of a set of data components. The registration is kept simple and minimum, so that even the end-users can tackle it, and is performed through visual forms where the service configuration data, i.e., the service end point and the request parameters to filter out relevant contents, have to be entered.
In the initial design phase, the composer selects one data component (*component selection*), and one of its pre-defined query (*query selection*). The query is executed and a result set is returned and visualized in a Web-based design environment.
Figure 3.3 shows the environment through which the user can visually program the integration of data extracted from multiple services to create a new UI Component. The left-hand panel displays the result set retrieved by querying the data components selected by the user from the repository of registered components. The panel on the right instead shows a *UI template* that the user chooses among different pre-defined UI templates, also depending on the device where s/he wants to execute the final composition.
As an example, let us suppose the user is interested in retrieving information about the Star Wars movie. S/he visually browses the services already

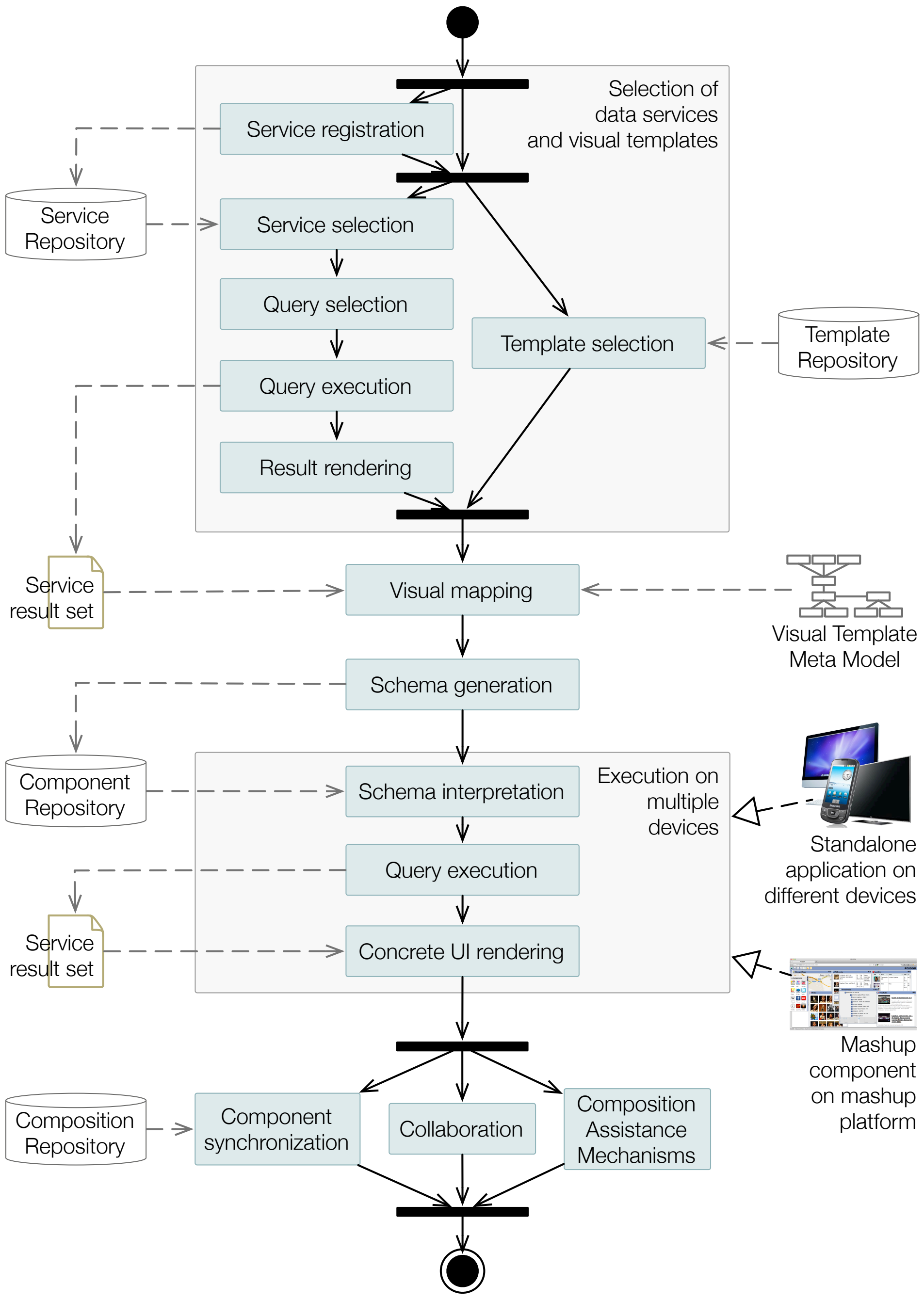
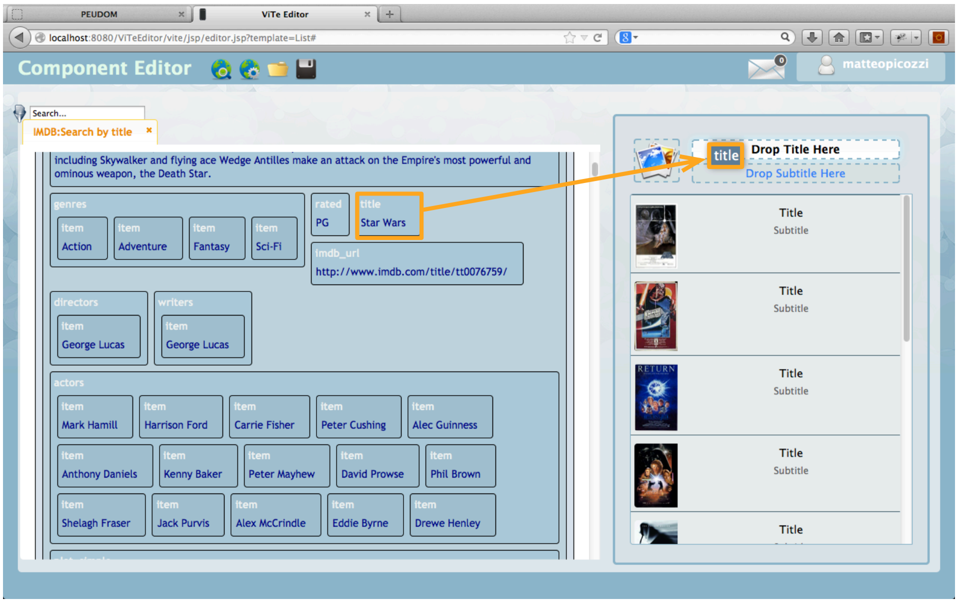


Figure 3.2: *The overall process for component and mashup creation.*



including Skywalker and flying ace Wedge Antilles make an attack on the Empire's most powerful and ominous weapon, the Death Star.

genres: item Action, item Adventure, item Fantasy, item Sci-Fi
rated: PG
title: Star Wars
imdb_url: http://www.imdb.com/title/tt0076759/
directors: item George Lucas
writers: item George Lucas
actors: item Mark Hamill, item Harrison Ford, item Carrie Fisher, item Peter Cushing, item Alec Guinness, item Anthony Daniels, item Kenny Baker, item Peter Mayhew, item David Prowse, item Phil Brown, item Shelagh Fraser, item Jack Purvis, item Alex McCrindle, item Eddie Byrne, item Drewe Henley

Figure 3.3: *Visual Mapping through the Design Environment.*

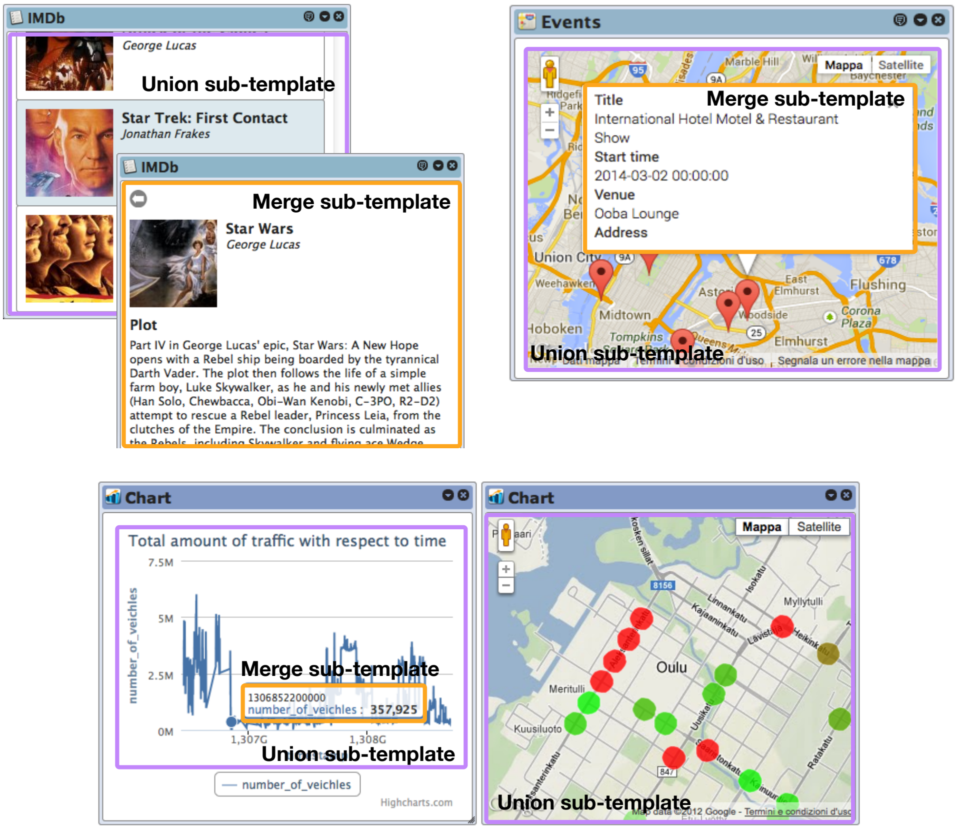
registered in the platform repository and selects a service publishing information about movies, e.g., *IMDB*. Guided by the visual environment, s/he can also set the parameter values to query the service, for example inserting "Star" as search key. The data panel on the left displays data retrieved by dynamically querying the service according to the defined settings.
S/he thus chooses a list-based UI template, like the one represented in figure, to display in form of a list (which we call *global sub-template*) the content items about movies. Each list item in the UI template is represented by three visual elements, namely $< title, subtitle, picture >$. As better explained in Chapter 4, these are the so called *visual renderers* that the user instantiates with the data extracted by the data component. Visual renderers are generic receptors of data that, based on the user choice, act as placeholders for data visualization during the component execution. The component schema consists of a set of *visual renderers*, each one representing the integration of data items coming from different resources. The way visual renderers are displayed according to a given layout then depends on the visual template selected by the user at design time, which in turn implies a specific mapping between the visual renderers in the abstract model and the widgets in the concrete visualization layout.
The UI template also includes a second view (a *merge sub-template*) that allows the user to map, on an arbitrary number of additional visual render-

ers, further attributes that s/he might want to add as details of each item in the union sub-template. In other words, the merge sub-template collects the result of joins among different service result sets, based on some *key attributes* in the union sub-template (e.g., the title and subtitle in the case of the list-based UI template). At run-time, the merge sub-template is constructed *on demand*, i.e., only for a specific item selected by the user in the union sub-template, thus reducing the complexity of fusing data and identifying duplicates when multiple services are involved, which would be required if instead acting on the the whole union result set.
In general, we assume that any UI template adopted for composition, and its corresponding concrete view instantiated at runtime, be structured into these two sub-templates, because this organization of the UI also determines our lightweight integration strategy that will be illustrated in the sequel of this paper. Nevertheless, this UI structure is also effective with respect to the usability of the resulting applications. Indeed, it is coherent with the well-known "global-detail" pattern commonly adopted in data visualization [25]. This pattern suggests providing a data overview first, then details-on-demand based on the selection of a given item in the overview. It can be applied for organizing the UIs on different client devices. It is for example very common in Web applications design [30], and can be effectively adopted in mobile apps design as well [14]. It is also adoptable in almost any type of visualization style.
For example, Figure 3.4 highlights the adoption of the pattern, and the resulting union and merge UI sub-templates, also in other UI templates currently available in our platform, namely map-based and chart-based UI templates:

- *Map-based UI template*: the map globally displays the result set items, each one as a Point Of Interest (POI) highlighted on the map; a pop-up balloon then displays the details for a selected POI.
- *Chart-based UI template*: the result set is shown as a series on a chart and the selection of one of its points then lead to more detailed information.

3.2.2 Visual mapping

After the selection of data components and UI templates, the user proceeds with the *visual mapping* activity, during which s/he maps items from the returned data sample onto the visual renderers of the UI template. The visual mapping can be operated starting from multiple data sources; in this



George Lucas

Star Trek: First Contact
Jonathan Frakes

Star Wars
George Lucas

Plot
Part IV in George Lucas' epic, Star Wars: A New Hope opens with a Rebel ship being boarded by the tyrannical Darth Vader. The plot then follows the life of a simple farm boy, Luke Skywalker, as he and his newly met allies (Han Solo, Chewbacca, Obi-Wan Kenobi, C-3PO, R2-D2) attempt to rescue a Rebel leader, Princess Leia, from the clutches of the Empire. The conclusion is culminated as

Title
International Hotel Motel & Restaurant
Show
Start time
2014-03-02 00:00:00
Venue
Ooba Lounge
Address

1306852200000
number_of_veichles : 357,925

Figure 3.4: *VT Components adopting List, Map and Chart UI Templates.*

case the visual mapping actions performed by the end users, from multiple result sets over the elements of a single UI template, define the integration schema for the construction of the integrated result sets. Based on the visual mapping an application schema is generated. During the execution phase, the schema, interpreted by a runtime engine, guides the execution of queries to the involved services and the generation of the concrete UI visualizing the integrated result set.

Through drag&drop actions the user thus associates interesting data items, selected from the left panel, to elements of the right-hand UI template panel. In other words, the user "manipulates" data to instantiate the visual elements that s/he will see and interact with in the final application. Also the UI template panel allows the user to immediately get realistic examples of the way data will be fused into a unified visual representation.

The user can select more than one data source for the UI template completion (e.g., another service publishing movies information, such as *Netflix*).

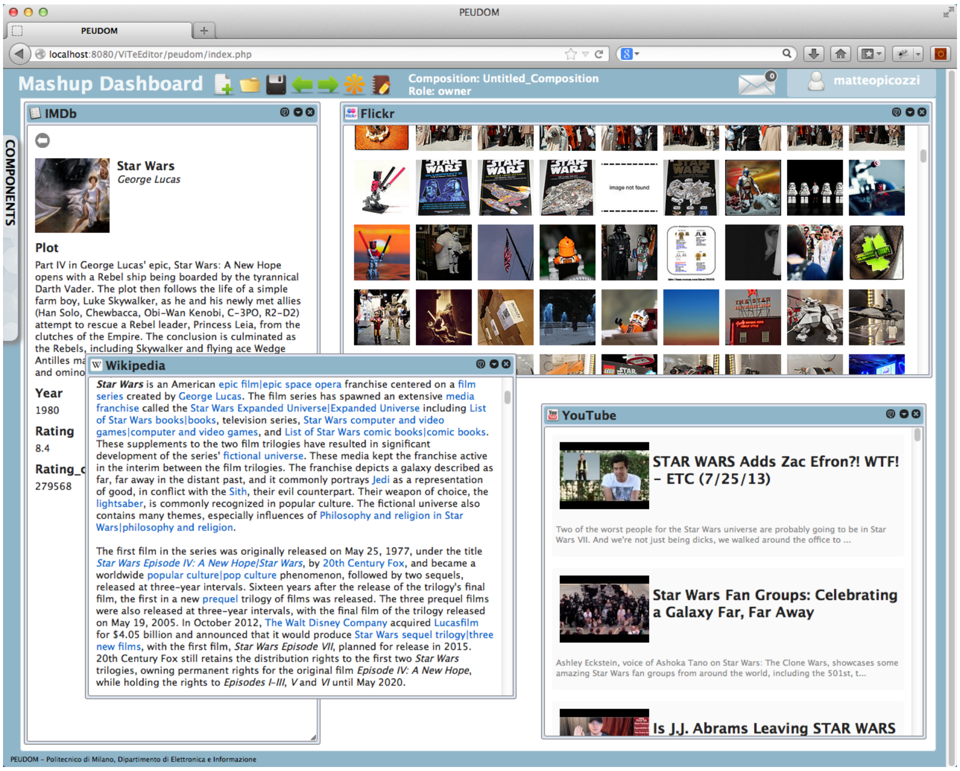
In this case the user-defined visual mapping not only specifies associations of data with visual elements; as better explained in Chapter 4 it implicitly specifies union or merge *join of data* for integrating the contents coming from the different services.

3.2.3 Schema generation and execution on multiple devices

As we will illustrate in Chapter 4, each visual mapping action contributes to the generation of a platform-independent schema that, for each element in the visual template in charge of rendering data specifies: *i)* the *binding* with the data sources and the corresponding query, as programmed by the visual associations operated by the user; *ii)* the invocation *data integration* operations in case of data coming from multiple sources. Thanks to model-to-code generative techniques, such schema then guides a client-side execution engine to dynamically generate the designed app, through the construction of the app UI, based on the adopted UI template, and the population of the visual renderers with data queried by the different services. The generated code can thus play the role of a standalone mashup application (this applies for example to the execution on mobile devices where apps are generally simpler), or of a component to be used for the creation of larger Web mashups.

3.2.4 Component synchronization within UI mashups

Our framework also provides an environment for the creation of mashups that synchronize UI components at the presentation layer [23]. Figure 3.5 shows the visual workspace where this kind of composition can be performed. In this case, the granularity of the composition is not related to data items and single visual elements of a larger UI template; rather the platform provides a repository of self-contained components equipped with their own UI, both user-created through the environment described above, or created by manually programming wrappers for public or private APIs and services to manage the visualization of data through a UI and the event-driven logic (i.e., capturing and propagating UI events) that enables our UI synchronization paradigm. Such UI components can then be visually composed by coupling exposed *events* and *operations* in an event-driven, publish-subscribe fashion (see Section 4.1.1). As we will better illustrate in the following, especially at this level where UI integration is addressed, our composition environment is characterized by a visual paradigm, with an immediate feedback about the effect of the user composition actions: the addition of each component is indeed immediately followed by the vi-


PEUDOM
localhost:8080/ViTeEditor/peudom/index.php
Mashup Dashboard
Composition: Untitled_Composition
Role: owner
matteopicozzi
COMPONENTS
IMDb
Flickr
Wikipedia
YouTube
image not found
PEUDOM - Politecnico di Milano, Dipartimento di Elettronica e Informazione

Star Wars
George Lucas

Plot

Part IV in George Lucas' epic, Star Wars: A New Hope opens with a Rebel ship being boarded by the tyrannical Darth Vader. The plot then follows the life of a simple farm boy, Luke Skywalker, as he and his newly met allies (Han Solo, Chewbacca, Obi-Wan Kenobi, C-3PO, R2-D2) attempt to rescue a Rebel leader, Princess Leia, from the clutches of the Empire. The conclusion is culminated as the Rebels, including Skywalker and flying ace Wedge Antilles ma
and omino

Year
1980

Rating
8.4

Rating_c
279568

Star Wars is an American epic film|epic space opera franchise centered on a film series created by George Lucas. The film series has spawned an extensive media franchise called the Star Wars Expanded Universe|Expanded Universe including List of Star Wars books|books, television series, Star Wars computer and video games|computer and video games, and List of Star Wars comic books|comic books. These supplements to the two film trilogies have resulted in significant development of the series' fictional universe. These media kept the franchise active in the interim between the film trilogies. The franchise depicts a galaxy described as far, far away in the distant past, and it commonly portrays Jedi as a representation of good, in conflict with the Sith, their evil counterpart. Their weapon of choice, the lightsaber, is commonly recognized in popular culture. The fictional universe also contains many themes, especially influences of Philosophy and religion in Star Wars|philosophy and religion.

The first film in the series was originally released on May 25, 1977, under the title *Star Wars Episode IV: A New Hope|Star Wars*, by 20th Century Fox, and became a worldwide popular culture|pop culture phenomenon, followed by two sequels, released at three-year intervals. Sixteen years after the release of the trilogy's final film, the first in a new prequel trilogy of films was released. The three prequel films were also released at three-year intervals, with the final film of the trilogy released on May 19, 2005. In October 2012, The Walt Disney Company acquired Lucasfilm for $4.05 billion and announced that it would produce Star Wars sequel trilogy|three new films, with the first film, *Star Wars Episode VII*, planned for release in 2015. 20th Century Fox still retains the distribution rights to the first two *Star Wars* trilogies, owning permanent rights for the original film *Episode IV: A New Hope*, while holding the rights to *Episodes I–III*, *V* and *VI* until May 2020.

STAR WARS Adds Zac Efron?! WTF! – ETC (7/25/13)

Two of the worst people for the Star Wars universe are probably going to be in Star Wars VII. And we're not just being dicks, we walked around the office to ...

Star Wars Fan Groups: Celebrating a Galaxy Far, Far Away

Ashley Eckstein, voice of Ashoka Tano on Star Wars: The Clone Wars, showcases some amazing Star Wars fan groups from around the world, including the 501st, t...

Is J.J. Abrams Leaving STAR WARS

Figure 3.5: *Visual synchronization of UI components.*

sualization of the component UI, through which the user can start accessing and browsing the component data set. Possible synchronizations that can be defined with other components already in place in the composition workspace are automatically identified, based on compatibility rules, and visually highlighted, e.g., through different colors (red or green) of the component border. The definition of component couplings is then supported by dialog windows where events and operations are described in terms of the effect they produce (e.g., "Event: Location available", and "Operation: A marker will be added to the map"), not in terms of parameter name and type required by the definition of event and data flows. The definition of a coupling is immediately operated, by updating the status of the subscribed components, based on the defined synchronization rule. In this way, the users explore and easily actuate different design alternatives, and interactively and iteratively compose their applications.

3.2.5 Composition assistance mechanisms

In order to support users during the development process, it is useful to adopt mechanisms that can alleviate the composition effort hiding technological aspects and complexity to users. We identify as critical activities: *(i)* the component selection; *(ii)* the similarity assessment of components; *(iii)* the compatibility between events and operations in the synchronization of components.

To support these activities, we provide mechanisms that give visual feedbacks to users if they are trying to connect suitable components or not.

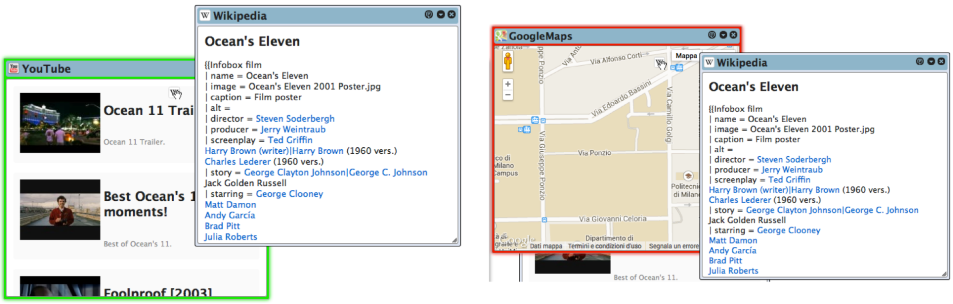
Figure 3.6a shows the highlighting mechanisms in component coupling. When a user wants to couple two components, the border of the target component is highlighted, in green if the component is compatible and in red otherwise.

Also, once an event (or an operation) is selected in the dialog box illustrated in Figure 3.6b, a color code helps the user to understand which operations (or events) are compatible with the previously selected event (or operation).

When composing a mashup, the composer first needs to identify the "right" components. The selection can be based on the syntactic and semantic fitness of each component within the mashup under construction, but also on quality measures. As soon as components are added into the composition, the assessment of the quality of the overall composition can indeed take place, and recommendations can be provided to the user accordingly.

Figure 3.6c shows examples of the recommendations generated by our tool, to guide the user in the selection of alternative or additional components that can maximize the quality of the overall mashup. The starting point for computing the quality indexes on which the recommendations are based is the quality of each single component [17]. However, the composition schema is also taken into account, to identify the *role*, i.e., the importance, that components play within the composition [18]. The quality of the composition can thus be evaluated as an aggregation of the quality of the single components, weighed on the basis of the components' role [22]. The analysis of the composition can also provide indications about the richness of the integration logic, revealing whether the composition introduces information spaces, functionality sets and visualizations that are richer than what would be achieved by accessing separately individual components. Recommendations can thus be generated, by ranking the components available in platform on the basis of their attitude to increase the quality and the value of the mashup.

In the following chapters the adopted quality model and the recommenda-


YouTube

Ocean 11 Trai

Ocean 11 Trailer.

Best Ocean's 1 moments!

Best of Ocean's 11.

Foolproof [2003]

Wikipedia

Ocean's Eleven

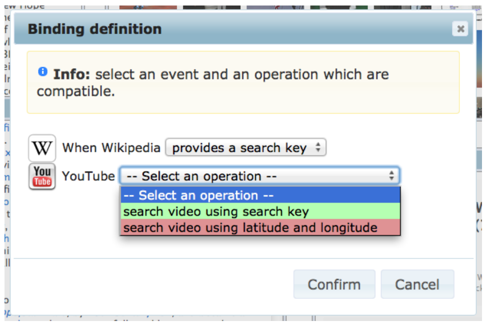
{{Infobox film
| name = Ocean's Eleven
| image = Ocean's Eleven 2001 Poster.jpg
| caption = Film poster
| alt =
| director = Steven Soderbergh
| producer = Jerry Weintraub
| screenplay = Ted Griffin
Harry Brown (writer)|Harry Brown (1960 vers.)
Charles Lederer (1960 vers.)
| story = George Clayton Johnson|George C. Johnson
Jack Golden Russell
| starring = George Clooney
Matt Damon
Andy García
Brad Pitt
Julia Roberts

Wikipedia

Ocean's Eleven

{{Infobox film
| name = Ocean's Eleven
| image = Ocean's Eleven 2001 Poster.jpg
| caption = Film poster
| alt =
| director = Steven Soderbergh
| producer = Jerry Weintraub
| screenplay = Ted Griffin
Harry Brown (writer)|Harry Brown (1960 vers.)
Charles Lederer (1960 vers.)
| story = George Clayton Johnson|George C. Johnson
Jack Golden Russell
| starring = George Clooney
Matt Damon
Andy García
Brad Pitt
Julia Roberts

(a) *Highlighting of compatible components in coupling definition.*


Binding definition

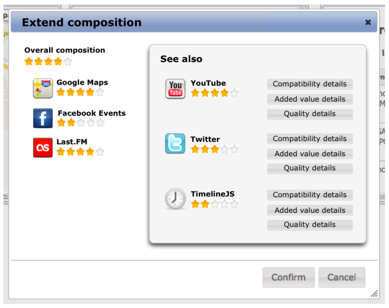
Info: select an event and an operation which are compatible.

When Wikipedia: provides a search key

YouTube: -- Select an operation --

- -- Select an operation --
- search video using search key
- search video using latitude and longitude

(b) *Coupling creation dialog.*


Extend composition

Overall composition ★★★★☆

Google Maps	★★★★☆
Facebook Events	★★☆☆☆
Last.FM	★★★★☆

See also

YouTube	★★★★☆	Compatibility details / Added value details / Quality details
Twitter	★★★☆☆	Compatibility details / Added value details / Quality details
TimelineJS	★★☆☆☆	Compatibility details / Added value details / Quality details

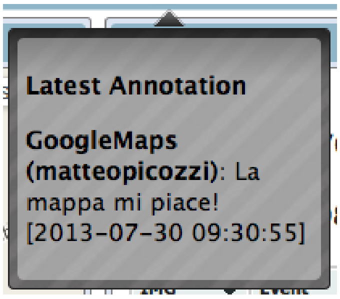
(c) *Example of quality-based recommendations for additional components.*

Figure 3.6: *Composition assistance mechanisms.*

tion mechanisms are described in detail.

3.2.6 Collaboration

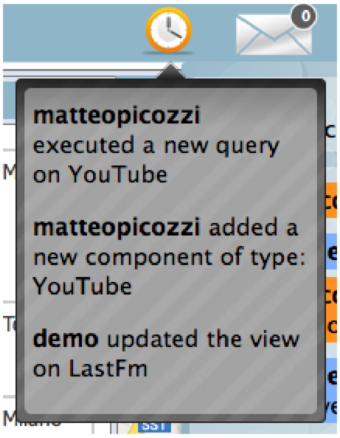
The introduction of collaboration mechanisms that allows users to share and co-create with other users the components and the composition they are working on is another important aspect of our paradigm. This is so important because users are nowadays used to collaborate with others, also through Web applications. In particular, in the last years, Web applications and tools that enable collaboration have been more and more diffused.
In our approach users can share their artifacts in order to show them to others or to collaborate. For example, we performed a field study in the Cultural Heritage domain. Through this study we assessed a guide of an archaeological park would be interested to share an application where s/he have collected all the material for the visit with the visitors but in the same time s/he would also like to share the work done with her/his colleagues in order to collaborate in the selection of contents. The extensions of our approach to cover collaboration requirements have been conceived to address these needs that we identified in all the context and application domains when our platform has been adopted.
Once an artifact is shared, users can collaborate asynchronously or synchronously with different mechanisms.
Figure 3.7 shows the main collaboration mechanisms. For the asynchronous collaboration, we considered annotations and activity logs. *Annotations* (see Figure 3.7a) enable users to leave comments on a particular resource (i.e., a service, a component or a mashup composition) or on its contents. *Activity logs* (see Figure 3.7b) are the history of the actions performed by all the users on a given shared resource.
For the synchronous mechanisms, users can instead use chat (Figure 3.7c), presence and actions awareness techniques. Instant messaging tools like *chat* help users to communicate synchronously and share comments, ideas, knowledge and directions to other users. *Presence and action awareness* techniques show to users how many users are working on the same resource, who they are and what they are doing. In other words, we provide the list of online users and visual feedbacks like highlighting (see Figure 3.7d) a part of a resource where other users are working on and a live synchronization of the actions performed by another user.



Latest Annotation

GoogleMaps (matteopicozzi): La mappa mi piace! [2013-07-30 09:30:55]

(a) *The last annotation on the GoogleMaps component.*

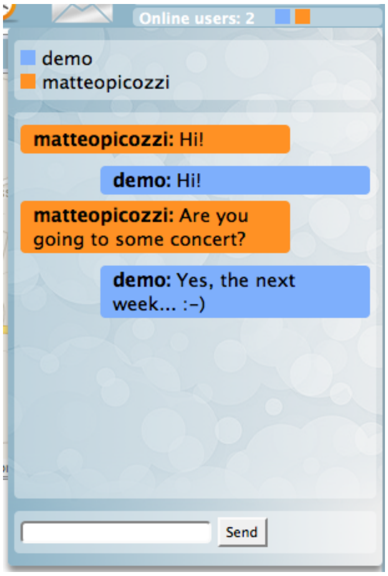


matteopicozzi executed a new query on YouTube

matteopicozzi added a new component of type: YouTube

demo updated the view on LastFm

(b) *The activity log.*



🟦 demo
🟧 matteopicozzi

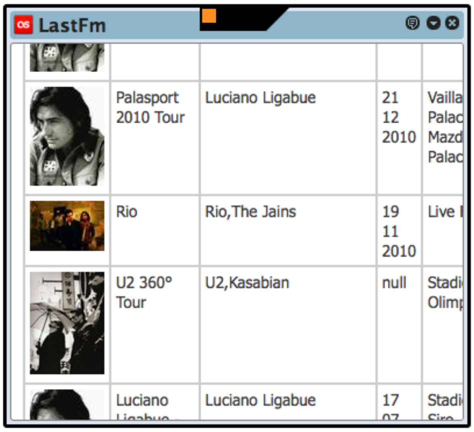
matteopicozzi: Hi!

demo: Hi!

matteopicozzi: Are you going to some concert?

demo: Yes, the next week... :-)

(c) *Chat and presence awareness mechanisms.*



	Palasport 2010 Tour	Luciano Ligabue	21 12 2010	Vailla Palac Mazd Palac
	Rio	Rio,The Jains	19 11 2010	Live
	U2 360° Tour	U2,Kasabian	null	Stadi Olim
	Luciano	Luciano Ligabue	17	Stadi

(d) *Action awareness highlighting of the component the other users are working on.*

Figure 3.7: *Collaboration mechanisms.*

CHAPTER 4

Models

In this chapter, we systematically define the modeling abstractions that characterize our approach. In Section 4.1 we describe the *Models for UI Component synchronization* that allows creating new composite applications by synchronizing the behavior of different UI Components at the presentation level. In Section 4.2 we illustrate peculiarities of our *Visual Integration Model* that, based on the completion of UI Templates, enables a light-weight paradigm for integrating Data and UI Components to create multi-device UI Components and mashups. The main elements that will be defined in Section 4.1 and Section 4.2 are schematically represented in the meta-model represented in Figure 4.1. Than, in Section 4.3, we present our *Quality-aware Models* for mashup composition that introduce the quality dimension to the composition of mashups. Finally, in Section 4.4, we introduce the *Collaboration Models* on which are based the proposed asynchronous and synchronous collaboration mechanisms.

4.1 Models for UI Component synchronization

Our models for UI Component synchronization capitalizes on models for UI synchronization defined in [19, 23, 96]. We here report the most salient

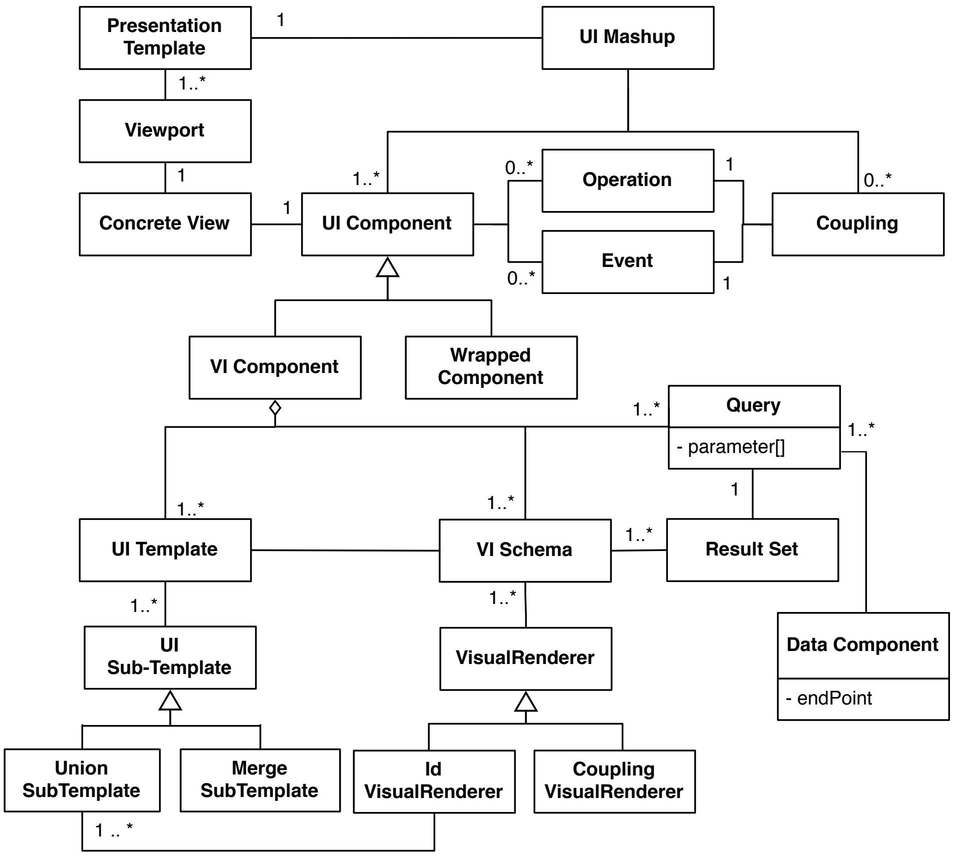


Figure 4.1: *Main elements of our UI-centric composition paradigm.*

features of such model, to facilitate the comprehension of the overall composition paradigm.

Definition 1. ***UI Component.*** A *UI Component* is a self-contained software module that is bound to one or more services providing data and/or functionality and is equipped with its own user interface (UI) (its *concrete view*). A UI Component also exposes an event-driven logic characterized by a set of *events*, E, that can be generated by the user interaction with its concrete view, and a set of *operations*, O, that some other components' events can activate to change its status when a synchronized behavior within a composite application is needed.

The specificity of UI Components, that characterizes them with respect to other components, for example Web services, is the presence of a UI as a means for the users to interactively navigate and manipulate the compo-

nent's content and to invoke business logics operations. Therefore, besides adding a presentation layer which is missing in Web and data services, the interaction with the UI in a sense replaces the invocation of Web services operations through SOA protocols.
In Section 2.1.2 we have already introduced the concept of UI mashup. In this Section we indeed provide a formal definition of such applications with respect to the elements that compose the proposed UI-centric composition paradigm.

Definition 2. ***UI Mashup.*** A UI mashup can be defined as

$$uim =< UIC, C >,$$

where UIC is the set of UI Components involved in the mashup, C is the set of *components' couplings* that determine the synchronized behavior of components within the mashup.

Component couplings are defined based on an event-driven, publish-subscribe integration logic [19, 23, 96]. Couplings are channels for inter-component communication, based on which the occurrence of a *published event* causes the execution of a *subscribed operation*, thus a state change in the subscribed component. Therefore, *couplings* can be defined as in the following:

Definition 3. ***Components' Coupling.*** Given two UI Components, uic_s and uic_t, a *coupling* synchronizing their behavior is a pair $< e_{uic_s}, o_{uic_t} >$ representing the subscription of an *operation* of the target UI Component, o_{uic_t}, to an *event* raised by the source UI Component, e_{uic_s}.

These last definitions particularly stress the integration dimension within mashups, which in our case is achieved through UI synchronization mechanisms. On the other hand, the UI Mashup definition intentionally does not cover the presentation of the composite application itself, which is typically achieved as the aggregation of each single component's UI. For example, in a UI Mashup executed on the Web, the presentation can be managed by a grid-based HTML layout where each component's UI is visualized within a *viewport*[1], i.e., a window or any other viewing area on the screen, generally implemented by means of an HTML `div` or an `iframe`, where the visualization and execution of the component takes place. In other words, our definition stresses the importance of each single component's UI.

[1]`http://www.w3.org/TR/CSS21/visuren.html`

4.1.1 UI Component synchronization

The adopted UI integration paradigm allows creating new composite applications by synchronizing the behavior of different UI Components at the presentation level [96].

4.1.1.1 Component Model

The paradigm is based on a component model where the application logic and the user interface cooperate with each other: any user action (e.g., the user click on a button) or also operation requests from other services result in the invocation of application logic functions (for example a new query over the component data set) that cause changes in the service state and are also mirrored in the user interface (for example an updated data set is displayed).
To achieve a composite application where different UI Components get synchronized, state changes must be captured and propagated to the other components to update their state. For these reasons, as defined above (see Definition 1), each UI Component is characterized by *events* that can also bring *parameters*, and by *operations*, whose invocation is triggered by events and enact component state changes.
Generally, only some of all the possible events that can be generated by the UI are selected and exposed as component events in charge of propagating state changes. Operations are derived by the component functionalities, in particular most of them are the functions that can be invoked through the service interface the component is bound to. They generally require a set of input parameters and returns set of values retrieved by the underlying service. As for the definition of component events, UI Components generally expose a subset of the functions offered by the underlying service - those ones that show some utility for the synchronization at the presentation level.

4.1.1.2 Synchronization Model

The synchronization of different UI Components is thus achieved through an event-driven, publish-subscribe model [23,96] based on the definition of couplings (see Definition 3) which are channels defining intra-component communications, based on which the occurrence of a *published event* causes the execution of a *subscribed operation*, thus a state change in the subscribed component.
As represented in Figure 4.2 such a composition logic can be managed through an event bus that allows a component to communicate and share its

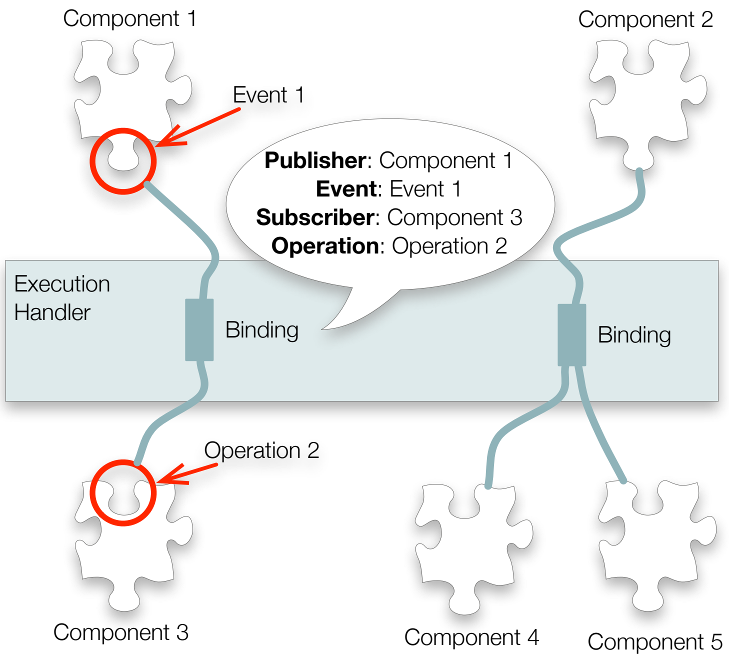


Figure 4.2: *PEUDOM composition metamodel.*

own UI events, or to observe and react to UI events of other components through its subscribed operations. Each component keeps running according to its own application logic, for example within the scope defined by an HTML `div`. As soon as the events specified in the defined couplings occur, the involved components publish them and the subscribed components capture them thus triggering the execution of their operations and updating their UI accordingly.

4.1.2 Domain Specific Languages

To guide the mashup execution, both the Component and the Synchronization Model have an XML-based specification.
Component properties, namely the binding with the actual service/API, the events and the operations are expressed by means of the UISDL (UI Service Description Language) [96], which provides a uniform model to coordinate the mashup composition and execution, which obviates the heterogeneity of service standards and formats by embedding only the information needed for synchronizing services at the presentation level. Listing 4.1 shows the part of the GoogleMaps UI Component descriptor that provides information about properties, events and operations. The last part contains quality

annotations that are described in Section 4.3.

```
<component name="GoogleMaps" label="Google Maps"
           ref="Components/googlemap"
           image="Components/googlemaps/icon.png">

  <!-- Parameters, events and operations-->
  <property name="centerLatitude" value="45.47841"></property>
  <property name="venterLongitude" value="9.23003"></property>
  <property name="venterLongitude" value="9.23003"></property>
  <property name="zoom" value="10"></property>

  <event name="provideGeo"
         label="provide the location latitude and longitude">

    <param name="lat" type="float"
           sem="Geography.owl/#Latitude"></param>
    <param name="lon" type="float"
           sem="Geography.owl/#Longitude"></param>
  </event>

  <event name="provideLocation"
         label="provide the location as search key">

    <param name="location" type="string"
           sem="Geography.owl/#Location"></param>
  </event>

  <operation name="addMarker"
             label="add a marker to the map">

    <param name="lat" type="float"
           sem="Geography.owl/#Latitude"></param>
    <param name="lon" type="float"
           sem="Geography.owl/#Longitude"></param>
    <param name="title" type="string"
           sem="General.owl/#Title"></param>
    <param name="text" type="string"
           sem="General.owl/#Content"></param>
    <param name="urlImg" type="string"
           sem="General.owl/#Image"></param>
    <param name="url" type="string"
           sem="General.owl/#URL"></param>
    <param name="date" type="string"
           sem="General.owl/#Date"></param>
  </operation>

  <operation name="centerMapPosition"
              label="center the map to the given coordinates">

    <param name="lat" type="float"
```

```
        sem="Geography.owl/#Latitude"></param>
   <param name="lon" type="float"
        sem="Geography.owl/#Longitude"></param>
  </operation>

  <operation name="centerMapLocation"
             label="center the map to the given location">

   <param name="location" type="string"
        sem="Geography.owl/#Location"></param>
  </operation>

  <operation name="searchByGeo"
             label="center the map to location geographic
                coordinates">

   <param name="location" type="geo"
        sem="Geography.owl/#Location"></param>
  </operation>

  <operation name="searchByKey"
             label="get location position using the search key">

   <param name="location" type="geo"
        sem="Geography.owl/#Location"></param>
  </operation>

  <!-- Quality annotations -->
  ...

</component>
```

Listing 4.1: *An example of UISDL DSL.*

The synchronization model is expressed by means of the XPIL (eXtensible Presentation Integration Language) XML-based language [23, 96]. Listing 4.2 is shown an XPIL document. In this example there are four UI Components and four couplings. The Wikipedia and Last.fm UI Components are coupled with YouTube and Flickr UI Components. The event `provideSearchKey` is raised when a link in the Wikipedia UI Component is clicked and the text of the link is passed as parameter to the coupled operation. The events named `provideEventGeoLocation` and `provideArtistName` are raised when a concert of the list provided by Last.fm is selected and one passes as parameters the latitude and the longitude of the concert to YouTube that searches videos by the position while the other passes the artist name to Flickr.

```
<xpil>
```

```
<!-- Component declaration -->
<component ref="../components/Wikipedia/wrapper.uisdl"
   id="Wikipedia" address="wikipedia"/>
<component ref="../components/YouTube/wrapper.uisdl"
   id="YouTube" address="youtube"/>
<component ref="../components/Flickr/wrapper.uisdl"
   id="Flickr" address="flickr"/>
<component ref="../components/LastFM/wrapper.uisdl"
   id="LastFM" address="lastfm"/>

<!-- Coupling declaration -->
<coupling id="1"
   publisher="Wikipedia"
   event="provideSearchKey"
   subscriber="YouTube"
   operation="searchByKey"/>
<coupling id="2"
   publisher="Wikipedia"
   event="provideSearchKey"
   subscriber="Flickr"
   operation="searchByKey"/>
<coupling id="3"
   publisher="LastFM"
   event="provideEventGeoLocation"
   subscriber="YouTube"
   operation="searchByGeo"/>
<coupling id="4"
   publisher="LastFM"
   event="provideArtistName"
   subscriber="Flickr"
   operation="searchByKey"/>

</xpil>
```

Listing 4.2: *An example of XPIL DSL.*

4.2 Visual Integration Model

One limit of the UI synchronization approach described above is the difficulty in adding a new UI Component into the platform. This indeed requires the intervention of expert programmers for the definition of adapters (or wrappers), to mediate the invocation of the service operations and managing synchronization events, and the creation of the component UI itself. Since programming a wrapper and developing a UI by hand is not a task affordable by non technical users, it results that, despite their intrinsic utility as tool for the creation of Web applications, mashup platforms are often perceived as closed environments, difficult to personalize through the in-

clusion of new components. The key idea, which we already illustrated in the previous section, is to instrument the composition platform with a set of pre-defined UI templates, and to ask the user to fill in such templates with selected data retrieved from interesting data sources. This mechanism ensures the confluence of data items retrieved through different services into a unified presentation layer. It therefore enables a kind of "integration-by-example" paradigm, with the visible advantage that the user sees a representation, at the extensional level, of the data sources, and also immediately achieve a representation of the way data will be integrated and displayed into the final applications. It let the users expressing their desiderata on *(i)* the selection of interesting data (e.g., how to query data sources), *(ii)* how to fuse data coming from multiple services (e.g., how to integrate data) and *(iii)* how to display the integrated result set.
Such a composition paradigm is based on a *Visual Integration* (VI) model, i.e., a set of abstractions enforcing the UI-centric nature of our composition paradigm and introducing a *lightweight data mediation* for the integration of data coming from multiple sources into unified views at the presentation level.
Since our approach also supports the creation of UI Components starting from basic data sources, we distinguish between Wrapped UI Components and VI Components (Visual-Integration based components).
Wrapped UI Components are pre-packaged by expert developers who manually program wrappers for accessing public or private APIs. When the accessed resources are not natively equipped with a UI, wrapping is also aimed at creating UI and especially at managing the event-driven logic needed for the synchronization of the resulting component at the presentation level [23, 96].
A *VI Component* is instead created by the end users by mapping result sets, extracted by one or more data components, on UI Templates. A *Data Component* provides read-only access to a data source, remote or local, by means of a suitable programming interface. Multiple queries can be defined on a same data source. The user can select one or more such queries to create the data set of a UI Component. Then a *UI Template* is an abstract representation of the VI component concrete view, which is adopted during the visual mapping activity to guide the selection and integration of data retrieved through data components. Formally:

Definition 4. ***Data Component.*** A *Data Component* is a pair

$$dc =< ep, Q >,$$

where ep represents the service *endpoint*, e.g., the URI of a RESTful service, and Q represents the set of *pre-defined parametric queries* over the data services.

Definition 5. ***UI Template.*** A *UI Template* can be characterized as the pair

$$uit =< VR, TE >,$$

where:

- VR is a set of *visual renderers*, vr_k, i.e., elements that provide visual placeholders for single data attributes, or for the aggregation or fusion of data attributes extracted from multiple data components. The way vrs are displayed in the final application is specific for each UI Template (e.g., a POI in a map, a text field for a list-based UI). However, at a higher level of abstraction, each vr can be considered merely as a "receptor" of data attributes.

- TE is the set of events that at runtime can be raised by the selection of the template visual renderers. The VI components making use of the template inherit this set of events.

Visual renderers are grouped in two different sub-templates.

- In the *union sub-template*, the vr_k (i.e., uvr_k) are in charge of displaying some key attributes as identifiers of data instances (*ID visual renderers*), plus few other representative attributes. At runtime, the data attributes associated with the *ID visual renderers* are exploited to detect and manage duplicates within the result sets. For this reason, each UI Template adopted for the rendering of the final application is characterized by a fixed set of uvrs.

- In the *merge sub-template*, the vr_k (i.e., mvr_k) display additional attributes providing details of the data instances selected in the union sub-template. Different from the union sub-template, the number of merge visual renderers is not defined a priori, and the users can arbitrarily add visual renderers, or even chose not to add a merge sub-template. Some mvr_k can play the role of *coupling visual renderers*, meaning that their selection will generate an event involved in the definition of one or more couplings.

Given the availability of Data Components and UI Templates, a VI Schema then consists of the following elements:

Definition 6. ***VI Schema.*** A VI Schema is

$$vis =< Q, uit, M >,$$

where:

- Q is the set of queries that the user selects from each involved Data Component to gather the VI Component data;
- uit is the user-selected UI Template associated to the component for the visualization of its integrated data set;
- M represents the set of mappings between data items, extracted through the queries in Q, and visual renderers characterizing the uit; it specifies the way multiple result sets are integrated into the selected uit. Therefore, independently of the adopted UI Template, a VI schema is represented by tuple mappings, M,

  $$M =< m_1, m_2, \dots, m_n >,$$

  where each m_k represents the mapping of data belonging to the results of a query $q \in Q$ onto the visual render vr_k.

A VI Schema is therefore an *abstract description*, i.e., independent of any specific visualization layout chosen for data display. It represents the way the visual elements that compose a selected *concrete view* in the final application display *fused* data items coming from multiple services according to the visual mapping operated by the users during the component design phase. The visual elements users can associate data with are specific for each UI template (e.g., a POI in a map, a text field for a list-based UI). However, at an abstract level, each visual element can be considered merely as a “receptor” of data attributes. An example of VI Schema is reported in Figure 4.3.

Definition 7. ***VI Component.*** A *VI component* can be described as the tuple

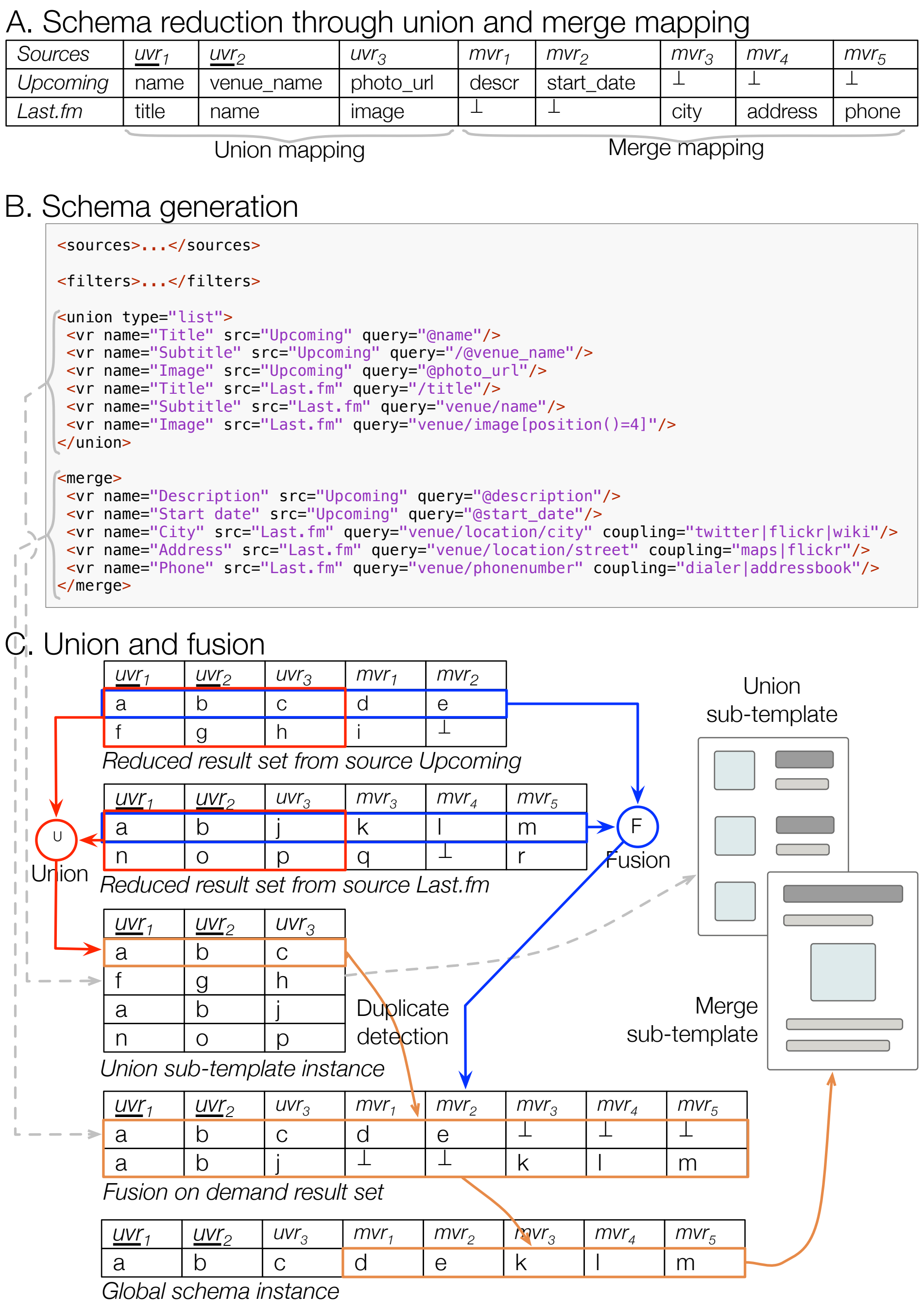
$$vic =< vis, E, O >,$$

where vis is a VI Schema denoting the basic elements of the component, while E and O are the sets of events and operations exposed by the component to make it comply with the event-driven logic needed by our UI synchronization paradigm. $E \subseteq TE_{uit}$, i.e., E is derived from the events associated to the UI Template visual renderers (see Definition 5). $O \subseteq Q$, i.e., O is derived from the set of queries that can change the status of the component by updating its content.

Therefore, while for Wrapped UI Components the component logic is blurred in the programmed wrapper, and depends on the opportunistic strategy adopted by the programmer, in VI Components a unique execution logic, replicated according to the technology of the target execution environments, is used to interpret the created VI Schema and generate, through model transformations, the code for the component execution.

4.2.1 UI-based integration of data sources

The creation of VI components is based on the integration of result sets coming from different data components; thus a data integration problem has to be managed. Data integration is usually modeled as a triple $< G, S, M >$, where G is the *global schema*, S is the *source schema*, and the M is the *mapping* between G and S [61]. In particular, the mapping associates each element of G with a query over S [61]. In order to increase the flexibility of the supporting tools and also to give to the users the freedom of composing their own applications without any constraints, we assume that the source schemas for the registered data components are not “known a-priori” (i.e., specified/described by an expert designer); rather they are derived by interpreting the returned result sets.
Therefore, the *result set representation* in the data panel of our design environment (see Figure 3.3) gives an idea of the content retrieved through a data component and also provides users with a “situational” *source schema S*, that corresponds to the whole set of attributes retrieved through the executed query. The *global schema G* is structured according to the set of visual renderers in the UI template. The user selection of interesting data attributes in the data panel then defines the source *local schema*, $ls \subseteq$ S, that is based on the mapping M between the source schemas and the global schema G. Note that the global schema is not completely defined a priori. What is known in advance is the set of uvrs that characterize the union sub-template in the selected UI template, while the structure of the merge sub-template is totally undefined and is constructed as soon as the user adds attributes in the merge sub-template.
More formally, let us consider the set of data sources $\{s_1, \ldots, s_j\}$ selected by the user. Associating some attributes of a data source s_i with the visual renderers of a UI template corresponds to specifying assertions of the form $VR \rightarrow Q_{s_i}$ expressing that the data visualized by each element vr_k is retrieved by a projection query Q_{s_i} over s_i. The set of all the attributes $\{s_i.a_h\}$ extracted through Q_{s_i} determines the source *local schema*, ls_i, i.e., a reduction of the original source schema that actually contributes to the



A. Schema reduction through union and merge mapping

Sources	$\underline{uvr_1}$	$\underline{uvr_2}$	uvr_3	mvr_1	mvr_2	mvr_3	mvr_4	mvr_5
Upcoming	name	venue_name	photo_url	descr	start_date	$\perp$	$\perp$	$\perp$
Last.fm	title	name	image	$\perp$	$\perp$	city	address	phone

B. Schema generation

```
<sources>...</sources>

<filters>...</filters>

<union type="list">
  <vr name="Title" src="Upcoming" query="@name"/>
  <vr name="Subtitle" src="Upcoming" query="/@venue_name"/>
  <vr name="Image" src="Upcoming" query="@photo_url"/>
  <vr name="Title" src="Last.fm" query="/title"/>
  <vr name="Subtitle" src="Last.fm" query="venue/name"/>
  <vr name="Image" src="Last.fm" query="venue/image[position()=4]"/>
</union>

<merge>
  <vr name="Description" src="Upcoming" query="@description"/>
  <vr name="Start date" src="Upcoming" query="@start_date"/>
  <vr name="City" src="Last.fm" query="venue/location/city" coupling="twitter|flickr|wiki"/>
  <vr name="Address" src="Last.fm" query="venue/location/street" coupling="maps|flickr"/>
  <vr name="Phone" src="Last.fm" query="venue/phonenumber" coupling="dialer|addressbook"/>
</merge>
```

C. Union and fusion

$\underline{uvr_1}$	$\underline{uvr_2}$	uvr_3	mvr_1	mvr_2
a	b	c	d	e
f	g	h	i	$\perp$

Reduced result set from source Upcoming

$\underline{uvr_1}$	$\underline{uvr_2}$	uvr_3	mvr_3	mvr_4	mvr_5
a	b	j	k	l	m
n	o	p	q	$\perp$	r

Reduced result set from source Last.fm

$\underline{uvr_1}$	$\underline{uvr_2}$	uvr_3
a	b	c
f	g	h
a	b	j
n	o	p

Union sub-template instance

$\underline{uvr_1}$	$\underline{uvr_2}$	uvr_3	mvr_1	mvr_2	mvr_3	mvr_4	mvr_5
a	b	c	d	e	$\perp$	$\perp$	$\perp$
a	b	j	$\perp$	$\perp$	k	l	m

Fusion on demand result set

$\underline{uvr_1}$	$\underline{uvr_2}$	uvr_3	mvr_1	mvr_2	mvr_3	mvr_4	mvr_5
a	b	c	d	e	k	l	m

Global schema instance

Figure 4.3: *Schema construction and data union and fusion strategy based on the specification of* vrs *and* visual mappings *in a Visual Integration schema.*

construction of the integrated data set.
Coherently with the structure of the UI templates adopted for data visualization, the local schema also consists of two parts:

- $uls_i = \{uvr_1, \ldots, uvr_m\}$, a reduction of the local schema that corresponds to the mapping of the attributes with the union sub-template: $\forall s_i.a_h \in uls_i, uvr_k \to s_i.a_h$;
- $mls_i = \{mvr_o, \ldots, mvr_n\}$, a reduction of the local schema that corresponds to the mapping of the attributes with the merge sub-template: $\forall s_i.a_h \in mls_i, mvr_k \to s_i.a_h$.

To better clarify the procedure to build the local schema, let us consider the example illustrated in Figure 4.3, where a list-based UI template is adopted. Data are gathered from two sources $s_1 = Upcoming$ and $s_2 = Last.fm$. The user defines the local schemas ls_{s_1} and ls_{s_2} by selecting attributes from the two services and mapping them on the visual renderers of the list UI template. In particular, in the example in Figure 4.3.A the user identifies the following schema reduction

$$sr_{s_1} =< name, venue\_name, photo\_url, descr, start\_date >$$

and

$$sr_{s_2} =< title, name, image, city, address, phone > .$$

According to the UI template definition, the first three attributes of each schema reduction (i.e., $< name, venue\_name, photo\_url >$ for s_1 and $< title, name, image >$ for s_2) are associated with the same union visual renderers in the global schema ($< uvr_1, uvr_2, uvr_3 >$), while the other remaining fields are mapped onto distinct merge visual renderers. As reported in Figure 4.3.C, the result consists of the two local schemas

$$ls_{s_1} =< uvr_1, uvr_2, uvr_3, mvr_1, mvr_2 >,$$

and

$$ls_{s_2} =< uvr_1, uvr_2, uvr_3, mvr_3, mvr_4, mvr_5 > .$$

The table represents indeed how selected attributes, associated with visual renderers, are re-named according to the visual renderer names. In particular, we assign the same attribute names to all the attributes assigned to the union sub-template (e.g., uvr_1, uvr_2 and uvr_3 in the example), while we assign distinct names (e.g., from mvr_3 to mvr_5) to the attributes assigned to the merge sub-template. The two data sources are thus overlapped on the

basis of the attributes associated with the ID visual renderers in the union sub-template[2], in the example uvr_1, uvr_2.
The global schema G is thus obtained as a Universal Relation [76], i.e., as the union of all the the names of attributes in the local schemas. The assumption at the basis of the Universal Relation is that attributes with the same name in the global schema refer to the same property [76], thus there is no reason to replicate them. Therefore, as also represented in Figure 4.3.C, in the example described above

$$G = ls_{s_1} \bigcup ls_{s_2} =< uvr_1, uvr_2, uvr_3, mvr_1, mvr_2, mvr_3, mvr_4, mvr_5 > .$$

4.2.2 Domain Specific Language

The VI schema generated from the user through the visual mapping activity contains rules for the automatic instantiation of the final mashup. Figure 4.3.B shows a simplified fragment of the schema generated for our reference example.
The schema mirrors the UI-centric nature of our approach, since it describes the structure of the presentation layer in terms of the adopted visual renderers. All the other properties, that are necessary to retrieve and fuse data, are specified in correspondence of single visual renderers:

- For each vr_k, the `src` attribute specifies the data components providing the mapped data and the corresponding query; the settings to invoke the data component (URI and query parameters) are automatically added at the beginning of the schema (not reported in figure for brevity). Data component properties may also include the definition of data filters, defined by the user and proposed at runtime to progressively refine the result set. Also, if multiple data components and queries are specified for a given vr_k, this will imply at run time the application of corresponding data union and fusion policies.

- For each vr_k the `coupling` attribute specifies the list of additional UI components subscribed to the selection of merge visual renderers (in the example Twitter, Flickr and Wikipedia for the `city` visual renderer). Each coupling is then interpreted as a pair

  $$< mvr_k.selection, UIcomp_i.operation >,$$

  specifying that an operation of the UI component uic subscribes to the selection of the visual element mvr_k.

[2]The indication of which attributes are keys can derive from pre-defined settings in the UI template definition. The user can modify these settings through a functionality offered by the design environment.

This schema therefore specifies the behavior of a self-contained mashup, or it can also specify the properties of a UI mashup component that has to synchronize with other components within a larger composition. In the last case, the created component exposes: *(i)* as *events* the selection of visual renderers, carrying as parameter the displayed data; *(ii)* as *operations*, the selection queries at the basis of the component creation (the one selected by the users).Under this assumption, VI components are aligned with our UI Component model characterized by events and operations [23].

4.2.3 Union and fusion

The models previously introduced are complemented with a policy for data union and fusion (Figure 4.3.C) that determines the way instances coming from different services, but referring to a same real-word entity, have to be managed. At run time, the union sub-template is instantiated with the extracted data, based on the union local schema, the uls_i, of each involved data component, without checking, at least initially, if duplicates exist. This check is performed only when the user selects a specific instance from the union sub-template, thus a *Data Fusion on Demand* policy is adopted. The aim is to reduce the computational effort, by limiting it only to the actual need of eliminating redundancies related to the user selection.

The adopted algorithm is reported in the Alghoritm 1 Listing. In a first phase, the local schemas of the involved sources and the global schema for the result set under construction are initialized. In particular, starting from the user selected instance, si, we derive the local schema of the service from which si is extracted, ls_{si}.

In order to identify duplicates, a comparison set, CS, is built by adding all the instances belonging to the result sets, RS_i, of all the other services involved in the mashup. The key attributes of the user-selected instance si, K_{s_i} are then compared with the key attributes of all the instances in CS, K_{cs_i}. In our current implementation, the similarity among instances is evaluated by adopting the Soundex metrics [8]. In particular, the function $IsSimilar(K_{si}, K_{cs_i})$ computes a similarity score, sc, based on this measure. If sc is greater than a specific threshold, τ (in our experiments $\tau = 0.965$), the items are considered similar. The merge attributes of cs_i are added as merge attributes of the selected instance. As also represented in Figure 4.3.C, the final instance is thus composed of the attributes belonging to union local schema of si, the attribute in the merge attributes of si, and the attributes in the merge local schemas of all the similar instances.

Once the user, after accessing the details of the selected item, returns back

ALGORITHM 1: Data Fusion On Demand

RS_i: result sets for the i-th data component
si: the instance selected by the user at runtime
CS: comparison set, including all the instances to be compared with si to identify duplicates
cs_i: i-th item of CS

$GetLocalSchema(cs_i)$: returns the local schema of the origin data source of the instance cs_i
$GetPrimaryKey(cs_i)$: returns the primary key values of the instance cs_i
$GetMergeAttributes(ls_i)$: extracts from the local schema ls_i the attributes in mls_i
$AddMergeAttributes(mls_i, mls_j)$: adds the attributes in mls_j to mls_i
$IsSimilar(K_i, K_j)$: returns true if two passed primary keys are similar, based on the adopted similarity measure

```
begin
    // Initialization of ls for the origin data source of $si$
    $ls_{si} \leftarrow GetLocalSchema(si)$
    // Initialization of the comparison set $CS$
    forall the $RS_i$, $si \notin RS_i$ do
        add $RS_i$ to $CS$
    end
    // Search for similar items with the comparison set CS
    $K_{si} \leftarrow GetPrimaryKey(si)$
    forall the $cs_i \in CS$ do
        $K_i \leftarrow GetPrimaryKey(cs_i)$
        // Similarity Evaluation
        if $IsSimilar(K_{si}, K_i)$ then
            $ls_i \leftarrow GetLocalSchema(cs_i)$
            Fuse($ls_{si}$,$ls_i$)
            remove $cs_i$
        end
    end
end

Fuse($ls_{si}$,$ls_i$) {
    $mls_i \leftarrow GetMergeAttributes(ls_i)$
    $mls_{si} \leftarrow GetMergeAttributes(ls_{si})$
    $AddMergeAttributes(mls_{si}, mls_i)$
}
```

to the visualization of the union sub-template, our policy for handling duplicates also updates the visualized union set. In particular, the identified duplicates are filtered out, and the only instance selected by the user is visualized.
It is evident that the complexity of the previous algorithm is especially related to the comparisons needed for the identification of duplicates. However, our choice to concentrate on the only instance selected by the user and compare it with all the other instances in CS allows us to keep the complexity linear to the cardinality of CS.

4.3 Models for quality-aware mashup composition

Mashup applications generally consist of a single page, with a simplistic presentation layer, usually deriving from the combination of the layout of each individual component, and an application logic mainly deriving from the operations exposed by the involved components [16]. The additional integration logic has a limited complexity.
One could assume that, being mashups "simple" Web applications, their quality could be addressed by the methods so far proposed for traditional Web applications. This is partially true: traditional principles must not be neglected; however, models need to be re-purposed to capture the salient characteristics of these applications. This is the conclusion we reached by analyzing about 100 mashups available on programmableWeb.com, by applying criteria and metrics related to traditional dimensions of the perceived quality of Web applications, e.g., accessibility and usability [16].
We compared the results achieved through such traditional metrics with the results of a heuristic evaluation conducted by a pool of five independent evaluators, PhD students and researcher acquainted with Web Technologies and Web mashups. The study revealed a discrepancy between the two assessments, highlighting that understanding the quality of mashups requires models that takes into account the specifics of such applications. For example, several applications were ranked as good on the basis of Web quality metrics, but the expert inspection revealed that they just "embedded" some APIs without any attempt to define an integration logic, which is instead a typical aspect of mashups.
Starting from these considerations we tried to understand how the quality of a mashup can be characterized, and haw we could exploit a quality model to assist the end user in the composition of quality mashups. The starting point or our investigation was a quality model specific for mashup components [17]. Beeing mashups composite applications our assumption was that it is

fundamental to base our approach on the quality of elementary components. We then identified the quality dimensions of component aggregations. Independently of the adopted quality model, we assume that within our tool quality properties are visible through ad-hoc descriptions on which the choice of components by mashup composers can be based. For example, in our mashup tool components are made available in a repository where *component descriptions* specify both the functional properties of the component, shown in Listing 4.1, (e.g., exposed operations, I/O parameters, their syntactic and semantic categories to assess component compatibility and similarity), and also quality annotations.

```
<component>

   <!-- UISDL Parameters, events and operations-->
   ...

   <!-- Quality annotations -->
   <quality>
      <reputation>1</reputation>
      <languages>
         <language></language>
      </languages>
      <dataFormats>
         <dataFormat>XML</dataFormat>
         <dataFormat>JSON</dataFormat>
      </dataFormats>
      <security>no autentication</security>
      <timeliness>0.9</timeliness>
      <accuracy>0.9</accuracy>
      <completeness>0.8</completeness>
      <availability>1</availability>
   </quality>

</component>
```

Listing 4.3: *Excerpt of a descriptor adopted in PEUDOM for the specification and quality properties of a component.*

Listing 4.3 reports a simplified example of such a descriptor. Besides events and operations, which characterize our publish-subscribe composition paradigm, the descriptor also specifies quality annotations, expressing quality properties such as the complexity of the component's technological properties (e.g., languages and data formats enhancing operability and interoperability), the richness and completeness of the provided data, the UI usability. These properties (e.g., the available languages and formats) are partly derived from the documentation of services and APIs, disclosed by the component developers - if any. Some other properties may also derive from evaluations that the administrator of the mashup platform performs at the component registration time. Also, quality and popularity data disclosed by public ranking services (e.g., Alexa (`http://www.alexa.com`)) can be taken into account.

4.3.1 Dimensions for quality-aware assisted composition

Let us assume that, using a mashup-maker tool, the mashup composer can access a component registry C in which each component c_i is associated with a *component descriptor* specifying functional properties [19, 96], and a *quality vector*, $QV_i = [qa_{i1}, qa_{i2}, \ldots, qa_{in}]$, storing the values computed through the metrics associated with a set of component quality attributes. It is thus possible to define the value of a *quality index* for the i-th component (QI_i) as an aggregation of the different qa_i, possibly weighed to privilege some quality attributes over others. The QI computed for each single component is the basis for the generation of quality-aware recommendations. When the user starts the composition, and the workspace is empty, components are first ranked based on the value of their QIs. Each time the user adds a new component, all the other components in C are classified and ranked according to the criteria that we describe in the following.

4.3.1.1 Component compatibility and similarity

Inconsistencies at the composition logic level can cause a low quality of the mashup. When the user extends the current composition with new components, the *compatibility* of such components with the current status of the composition is an important factor; components in C can be therefore analyzed and scored accordingly. In particular, compatibility can be estimated as the combination of syntactic and semantic compatibility:

- *Syntactic compatibility* checks for *type compatibility* among the operation parameters exposed by one candidate component and the parameters of all the other components already in the composition.
- *Semantic compatibility* subsumes the syntactic compatibility, and checks whether the operation parameters of the candidate component belong to the same (or a similar) semantic category of at least one of the operations exposed by the other components already in place.

Component similarity can also help determine in which measure some components in C can functionally substitute the ones already in the composition. This property can be useful to recommend alternative components that can improve the composition quality.

Compatibility and similarity can be verified on the basis of annotations that enrich the components' descriptor with semantic categories (based on ontological entities) for both operations and parameters. Examples of such annotations in a component descriptor are represented in Figure 5.4a, where the specification of events and operations are enriched with "similarity" tags expressing semantic meanings.

4.3.1.2 Role-based aggregated quality

Compatibility and similarity can ensure that a more consistent composition is produced. An estimation of the mashup quality can be then achieved by aggregating the QIs of the individual components. In particular, as soon as new components are added into the composition workspace, the compatible components in C can be ranked based on their capacity to increase the quality of the overall mashup.
The quality of the overall composition cannot be simply quantified as a plain aggregation of the individual QIs; rather the aggregation must take into account the *role* that each component plays in the composite logics. By analyzing the most popular mashups published on programmableWeb.com we identified a number of composition patterns, in which two component roles emerge [18]:

- In most cases one component assumes a central role in the composition, being the service the user interacts with the most. We call this component *master*. The master is the starting point of the user interaction causing the other connected components to react and synchronize accordingly, in practice implementing a "star" composition pattern.
- A *slave* is then a component whose behavior depends on another component; its state is mainly modified by events originating in a master component. Many mashups also allow the user to interact with slave components. However, the filtering of the content displayed by slave components depends on the user's interaction with the master component and occurs by automatically propagating synchronization information from the master to the slaves.

It emerges that master components, being central points of synchronization, have a major influence on the mashup quality - a master could even degrade the quality of the other components that depend on it. Therefore, the aggregation of the different QIs must be adequately weighted.
In order to identify master-slave dependencies during mashup composition, we model a mashup as an directed weighted graph $G = (V, E)$, where each

vertex $v_i \in V$ represents a component and each arc $e_{ij} \in E$ represents that one binding is defined between the two connected components, and therefore that v_i is master with respect to v_j.
Based on the analysis of all the paths in the composition graph, which reflect the defined bindings, for each component v_i we then define the *centrality* of a component v_i, $Centrality_i$, as a variant of the *betweenness centrality measure* [41], weighting all the shortest paths inversely proportional to their length [12].
This measure, applied to each component in the composition and normalized with respect to the maximum centrality, provides the weights to be used to aggregate the different QIs. The quality of the overall composition is therefore defined as:

$$QC = \sum_i Centrality_i * QI_i$$

QC is computed every time a new component c_i is added to the composition to identify in which measure c_i and all its similar components increase the quality of the composition.

4.3.1.3 Added value

In our analysis of mashups, we also tried to capture the concept of *added value* of a composition, conceived as the set of additional features (data, functions, visualizations) that the composition logics introduces with respect to accessing single services separately [16].
Mashups are developed in order to offer a set of functions that we call MFS (Mashup Function Set), and consequently retrieve and give access to a data set, MDS (Mashup Data Set), exploiting a set of visualization mechanisms, MVS (Mashup Visualization Set). Each single component c_i is also characterized by its own data set DS_i, a function set, FS_i, and a set of visualization mechanisms, VS_i [3]. When used within a mashup, smaller, *situational* portions of such sets, SDS_i (the component's Situational Data Set), SFS_i (the component's Situational Function Set) and SVS_i (the component's Situational Visualization Set) are considered, depending on the specific needs that the mashup is supposed to satisfy [18]. The composition thus provides an added value if the number of features that it offers, at least at one of the three layer (data, function, visualization), is greater than the

[3]Depending on their nature, components may or may not provide all the three layers. For example, map APIs expose data, functions and multiple visualizations. On the other hand, other components may just offer one of these layers. For example, an RSS Feed is a plain data source for which functionality, e.g., feed filtering, and visualizations have to be provided by other components.

amount of those offered by the single components, i.e., if

$$\bigcup_i SFS_i \subset MFS \vee \bigcup_i SDS_i \subset MDS \vee \bigcup_i SVS_i \subset MVS.$$

Operatively, to evaluate the added value of the composition at the *data layer*, it is possible to consider the *richness of data formats*, e.g., whether the mashup provides plain, multimedia or social data, which can be assessed on the basis of a classification of components reflecting the nature of their data sets. Also, it is possible to consider the *richness of additional information* deriving from the join queries enabled by the synchronization among the different components. This measure can be quantified as the ratio between the data bindings actually defined among components, and the number of all the possible data bindings. The former are evaluated by analyzing the composition model, where the data bindings actually defined are specified; the latter are instead evaluated by defining and analyzing a compatibility matrix, derived from the component descriptors, to identify all the possible join queries that could be achieved combining output parameters produced by a component with compatible data filtering operations exposed by other components. The added value at the data layer thus aggregates the richness of data formats with the richness of additional information.
The *functionality layer* addresses the *richness of functionality*. Similarly to what we propose at the data layer, this measure can be quantified as the ratio between the number of function bindings actually defined and the number of all the possible function bindings, which can be derived respectively from the components and the composition descriptions.
At the *presentation layer*, we then relate the multiplicity of visualizations with the multiplicity of data sources. In particular, we classify components as *data sources* when they have an own data set, and *viewers*, when they only provide visualizations on top of any external data set[4]. The added value can be assessed by considering the *richness of visualizations*, i.e., the number of different viewers associated with the involved data sources. In fact, in many cases, the analysis of the same phenomenon from the different perspectives that different visualizations can offer can better support decisions. Symmetrically, in other situations the analysis can be improved by aggregating heterogeneous data into a unified visualization. Hence, we also take into account the *cohesiveness of visualizations*, i.e., the capacity of a viewer to convey integrated data.
The aggregated measure of the composition added value can then be achieved by averaging the three distinct values, possibly assigning different weighs

[4]Some components, e.g., GMaps, can fall in both categories.

to the three layers. Such a measure is especially useful to understand whether the value of the current composition can be increased by adding new components.

4.3.2 Community perceived quality

The notion of recommendations can be further extended to take into account the best practices adopted by communities of mashup developers, so exploiting collaborative filtering mechanisms. If large repositories of components are available, then the most frequent associations among (categories of) components can be mined and exploited to suggest typical composition patterns that get consensus in a given community, and that therefore reflect the users' perception of the quality of the created mashups.
With the exception of few mashup-maker tools available online (e.g., Yahoo! Pipes), it is uncommon for a mashup platform to have large repositories with a number of components and compositions adequate for the mining tasks. In order to perform experiments on quality dimensions, we therefore mined recurrent composition patterns from a data set crawled from programmableweb.com, the largest collection of mashups and mashup components currently available on the Web. From the so-achieved database, we selected mashups with at least two components; then we mined the most recurrent combinations of components. Given the huge number of distinct components (more than 6000 at the time of our experiment), and the difficulty to reproduce such a large variety in a local repository, we identified the most recurrent categories; the extracted association rules thus reflect recurrent combinations at the category level. We thus defined a technique that exploits such extracted knowledge to guide the production of recommendations.
We assume all the components available in the local repository be classified according to the same categories used for the mining tasks. As soon as the state of the composition evolves, the set of categories of the involved components guide the filtering of pertinent association rules, i.e., of those rules where the set of categories in the antecedent part corresponds to the set of categories in the composition. Thus the categories appearing in the consequent part of the rule with higher confidence are projected over the local repository, to identify the components falling into those categories.

4.4 Collaboration model

When users are collaborating it is very important to allow them to communicate synchronously and asynchronously for achieving coordination. We have therefore identified an annotation system and a synchronous notification mechanisms [5, 6]. With this respect it is fundamental to identify the elementary objects on which coordination has to take place. We therefore model each mashup under construction as an interactive document, which indicates any instance of a schema (or *Document Object Model*) defining an identifiable unit of interaction. In this perspective, it is possible to identify three main types of document:

- The mashup *composition model*, describing the integration of the services forming the mashup;
- The mashup *UI Template*, describing the presentation aspect of the mashup;
- The actual contents presented at each moment by the different extracting data from services. We assume that each service will present contents according to some service-specific schema.

Assuming that schemata are expressed in an XML-based language, each document can be represented as a tree, where nodes define organizational structures and leaves present the actual content the users interact with. Hence, an annotation process (or respectively a live editing session) consists in the production of user-generated information (editing action) associated with some node or leaf in such a tree. In order to fix these concepts, we provide here an abstract view of this process for handling annotations.
Given a document $\mathcal{D}$, an $intervention$ on D aimed at synchronous or asynchronous collaboration is the process by which a collection $A_{\mathcal{D}} = \{a_1, \ldots, a_n\}$ additional information items is created, together with a map describing the relation of each item a_i with D. As discussed above, $\mathcal{D}$ is modeled as a composition of *content items*, according to a schema $S_{\mathcal{D}}$, so that its composition is defined by a tree $T_{\mathcal{D}}$, according to a PART_OF relation. We call $N_{\mathcal{D}}$ the set of nodes in $T_{\mathcal{D}}$, with each node corresponding to a content item. An additional information item can refer to D as a whole, or to any subtree or leaf in its composition. The structure of each information item is defined by a schema SA, supporting at least the following data [11]:

- the item *author*, as identified during the interaction;
- the *timestamp*, captured when the annotation any editing action on it is committed in the mashup repository;
- the *source*, as identified by the interactive selection of the part of the document to which the collaboration item refers;

- the actual *information* added by the author for the item through the collaboration intervention;
- its *visibility*, either private, public, or group-based.

An interactive process can exploit specific tools to identify the source, for example mouse pointing, selection of text, or sketching of shapes. In general, arbitrary fragments of the original content, or sets of fragments across the tree structure, could be selected and associated with an information item. For example, one could draw a shape to identify some areas of a picture, or select several areas and refer collectively to them [2]. Hence, the *annotation map* describing the relation of $\mathcal{A}_{\mathcal{R}}$ to $\mathcal{D}$ is defined by a function $am : \mathcal{A}_{\mathcal{D}} \rightarrow \wp(H_{\mathcal{D}})$, where $H_{\mathcal{D}} = N_{\mathcal{D}} \times POS$, and POS is a set of locations defined over the content items, specifying the source of the information item. Given an information item a_i we have $map(a_i) = \{(n_j, \{pos_1, \ldots, pos_{k_i}\})$, where n_j represents the node in $N_{\mathcal{D}}$ which is the root of the subtree containing all the items to which a_i refers, and each pos_k defines the location of a fragment in n_j identified by the collaboration process.
While interacting with the mashup, users can perform two types of collaborative activities: annotations and live editing, distinguished according to the nature of the info descriptor and to their usage within the platform.
With *annotations*, the info descriptor is of an arbitrary nature and can consist of any kind of digital data. The info added in an annotation can be used to generate an independent document, without corrupting the original mashup. The new document can be annotated in turn, thus supporting forms of *asynchronous collaboration*, for example by constructing annotation threads. Moreover, annotations can be used to request modifications of parts of the mashup itself, delegating the realization of the request to authorized users, i.e., users with visibility on the annotation. In particular, every user can access his or her private annotations, and all public annotations. Moreover, users belonging to some group can access annotations posted to that group.
With *live editing*, the info describes a set of modifications to the composition of a mashup, which are immediately activated on the instance in use by their author but also reflected to the aspects of mashup composition and behavior shared with other users, defining a form of *synchronous collaboration*. Both processes, annotation and live editing, are enabled by special mechanisms through which the actions on any instance of the mashup are captured and propagated to the other active instances of the mashup. As described in the next section, this is possible thanks to the interaction

between client-side modules, managing the composition and execution of each mashup instance, and a server-side module managing persistency and evolution of the mashup schema.
The combination of live editing and annotation poses some specific issues, concerning the problem of *orphan interventions*, i.e., interventions referring to content items which are no longer present in the document. While in principle some mechanism could be devised to retrieve content which has simply been moved in a different position [10], our strategy is to adopt a conservative stance, whereby if an intervention i refers to a node n in a subtree which has been eliminated by a live editing process e, we remove the intervention. A check on the values of the first components n_a and n_e (i.e., those in in $N_{\mathcal{D}}$) of $map(i)$ and $map(e)$ identifies these situations, by checking that $n_a \in subtree(n_e)$. Note that this problem is relative to interventions by the collaborating users on the composition and visual template models, while interventions on some content provided by a service can be retrieved if the same service provides the same content at some subsequent request, or if references to specific content instances are maintained by the platform.

CHAPTER 5

PEUDOM

In this chapter we present PEUDOM (Platform for End User Development of Mashups), from a functional and architectural point of view. In Section 5.1 we describe the overall functionalities and the overall architecture of the platform introducing the main modules. Section 5.2 describes the composition environments that are used to create visual template-based components and mashups. The execution environments, included the ones that enable the execution on different devices, are presented in Section 5.3. Finally, modules that provide composition assistance mechanisms and enable the collaborative co-creation of mashups are presented in Section 5.4 and in Section 5.5, respectively.

5.1 Overall functionalities and architecture

PEUDOM is the mashup-maker platform that makes concrete the models and the techniques described in the previous chapters by enabling end users to develop their mashups.
Figure 5.1 summarize the platform functionalities. The overall system allows the management of UI Components and UI Mashups through: CRUD functions (Create, Read, Update and Delete), a versioning system, and their

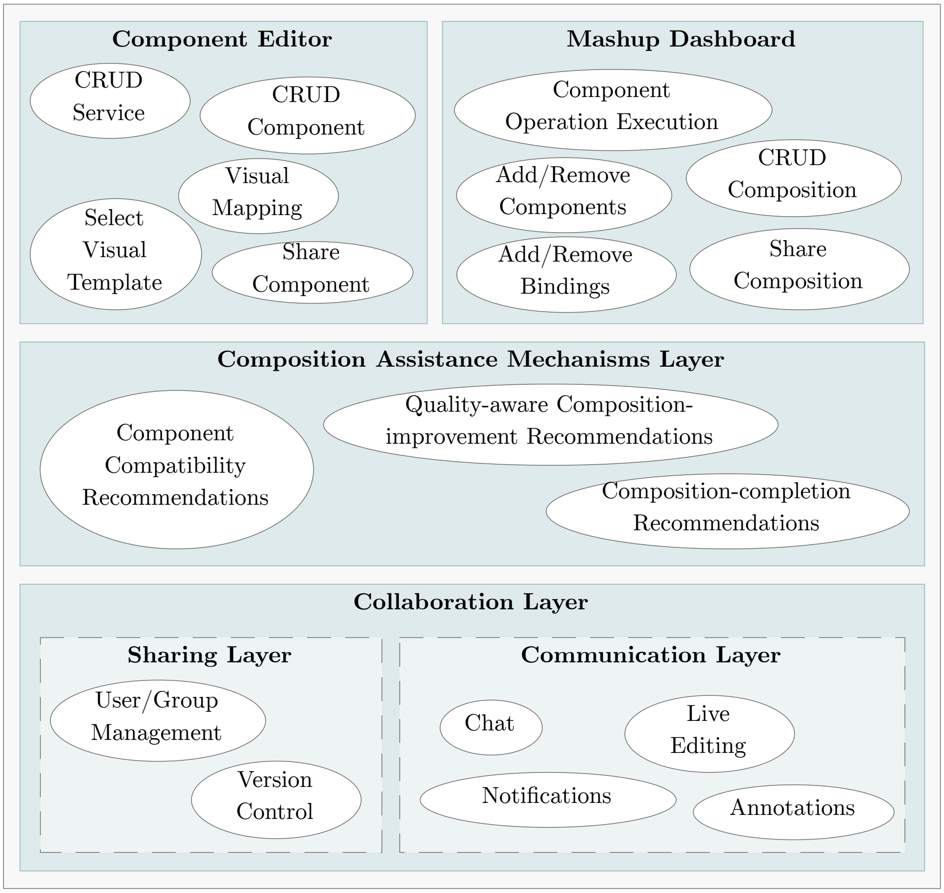


Figure 5.1: *PEUDOM overall functionalities.*

publication as shared resources in the platform repositories. The platform supports users in the choice of components and in the definition of couplings through Composition Assistance Mechanisms, e.g., by suggesting which components and which couples of events and operations are compatible. Through quality-aware recommendations users are supported to add new components or to substitute already added ones in order to improve the quality of the overall composition. With PEUDOM end users can also collaborate synchronously and asynchronously. Concerning synchronous collaboration, users that are working on the same resource can chat together and see the actions of the other users through live editing mechanisms. To collaborate asynchronously, users can annotate available resources or specific areas to indicate specific needs, e.g., requesting service versioning, or to ask for information to more experienced partners.

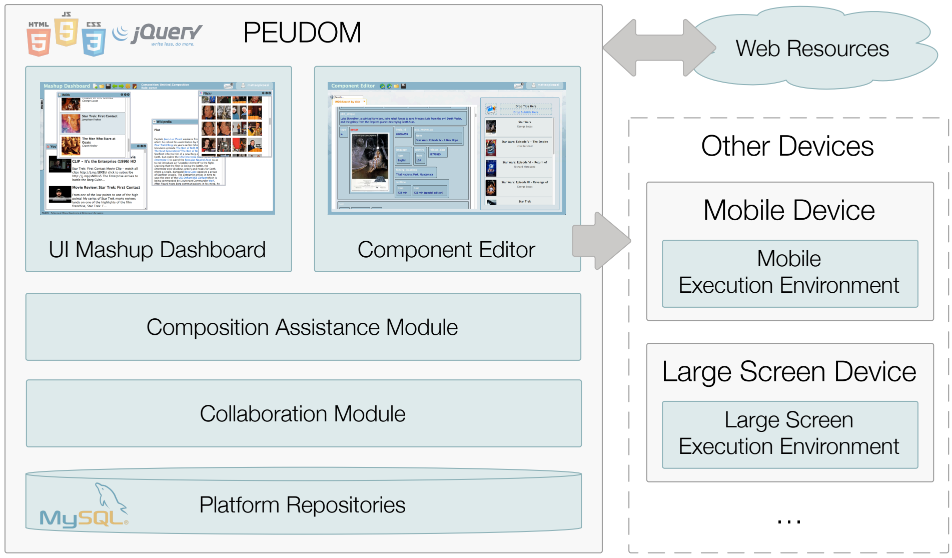


Figure 5.2: *PEUDOM overall architecture.*

Through a specific notification service, users become aware of similar requests from other partners, or of answers to their requests. Finally, they can directly interact with the working platform to retrieve particular versions of components or services and configure them in their personalized space.

Figure 5.2 shows the overall architecture of our platform. PEUDOM is composed by two main environments and some additional modules that are in charged of managing complementary functionalities. The *UI Mashup Dashboard* manages the creation and usage of mashups by adding, removing, synchronizing and interacting with components. The *Component Editor* enables registering, annotating and querying services, thus the creation of VI Components through mapping of the results of service queries onto the selected visual templates. Each created component has a platform-independent descriptor, the VI Schema, that allows to edit and execute it on multiple devices (e.g., mobile devices, large multi-touch displays or multimedia interactive blackboards). Besides the main composition environments, the *Composition Assistance Module* provides quality-based recommendations on syntactically and semantically compatible components, while the *Collaboration Module* implements collaboration mechanisms, e.g., live editing, chat and the other mechanisms for user-awareness already discussed in Chapter 3. Finally, the Platform Repositories layer includes:

- A *Service Repository* that contains the descriptions of registered services that are used by the Component Editor to query the data component. Services can be private or shared.

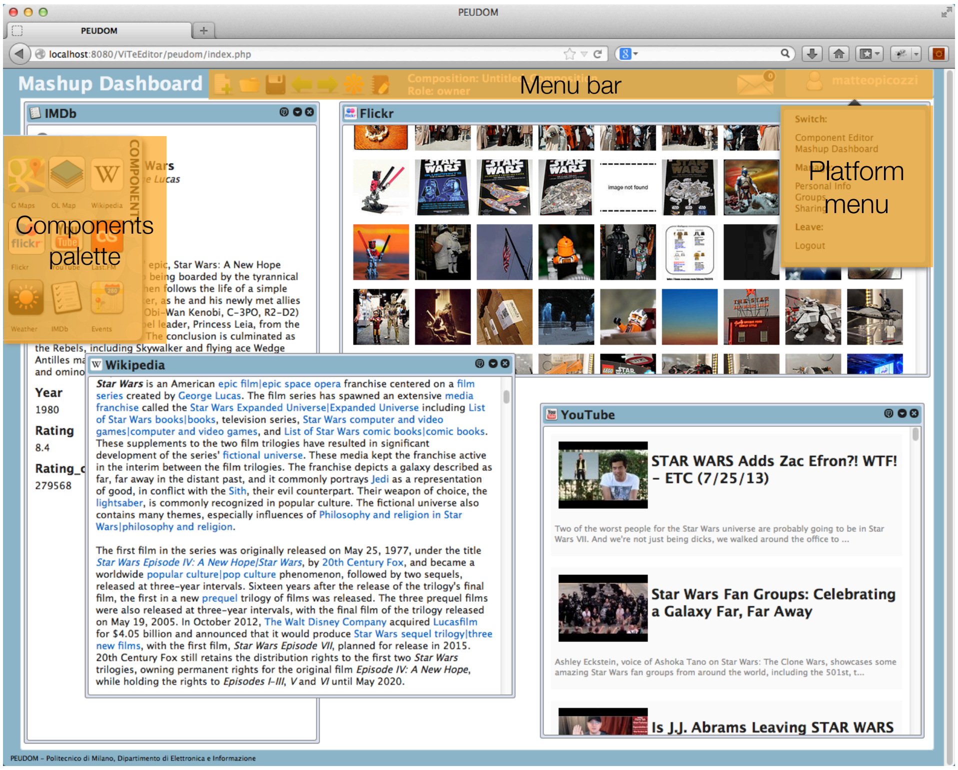
- A *Component Repository* stores the UI Component descriptors already presented in Section 4.1.1, together with with their sharing policies – they can be either private or shared.
- A *Composition Repository* contains the mashup compositions as described in Section 4.1.1. Compositions can be private or shared. The information about the sharing policies applied for a composition are stored in the Composition Repository as well.
- The *Assisted Composition Repository* includes all the data needed by the composition assistance mechanisms. It contains the Association Rules, the Compatibility and Similarity Matrices and the Quality Vectors (see Section 4.3).
- The *Collaboration Repository* contains all the information related to component and composition sharing and collaboration mechanisms (e.g., users, groups of users, chat sessions, annotations and live editing sessions).

5.2 Composition environments

This section describes the composition environments. Both the composition environments are Web applications; on the client side they are implemented with HTML5, CSS3 and JavaScript and are based on the jQuery framework. Server side JSP technology is adopted and a MySQL DBMS is than used to manage the platform repositories.

5.2.1 UI Mashup Dashboard

The UI Mashup Dashboard allows the users both to compose and use mashups, providing an intermixing of these two phases. This intermixing help users in the mashup development, because they can immediately see the results of their work in a progressive evaluation that helps reducing the design barriers. In this Section we focus only on the composition aspects, while mashup execution is discussed in Section 5.3.
In Figure 5.3 a screen shot of the UI Mashup Dashboard is shown. The main area of the interface is the workspace. In the workspace some components are instantiated, i.e., IMDB, Wikipedia, Youtube and Flickr. In the left side of the screen, there is the component palette that shows all the



Wars
e Lucas

epic, Star Wars: A New Hope
being boarded by the tyrannical
hen follows the life of a simple
er, as he and his newly met allies
Obi-Wan Kenobi, C-3PO, R2-D2)
el leader, Princess Leia, from the
The conclusion is culminated as
the Rebels, including Skywalker and flying ace Wedge
Antilles ma
and omino

Year
1980
Rating
8.4
Rating_c
279568

Star Wars is an American epic film|epic space opera franchise centered on a film series created by George Lucas. The film series has spawned an extensive media franchise called the Star Wars Expanded Universe|Expanded Universe including List of Star Wars books|books, television series, Star Wars computer and video games|computer and video games, and List of Star Wars comic books|comic books. These supplements to the two film trilogies have resulted in significant development of the series' fictional universe. These media kept the franchise active in the interim between the film trilogies. The franchise depicts a galaxy described as far, far away in the distant past, and it commonly portrays Jedi as a representation of good, in conflict with the Sith, their evil counterpart. Their weapon of choice, the lightsaber, is commonly recognized in popular culture. The fictional universe also contains many themes, especially influences of Philosophy and religion in Star Wars|philosophy and religion.

The first film in the series was originally released on May 25, 1977, under the title *Star Wars Episode IV: A New Hope|Star Wars*, by 20th Century Fox, and became a worldwide popular culture|pop culture phenomenon, followed by two sequels, released at three-year intervals. Sixteen years after the release of the trilogy's final film, the first in a new prequel trilogy of films was released. The three prequel films were also released at three-year intervals, with the final film of the trilogy released on May 19, 2005. In October 2012, The Walt Disney Company acquired Lucasfilm for $4.05 billion and announced that it would produce Star Wars sequel trilogy|three new films, with the first film, *Star Wars Episode VII*, planned for release in 2015. 20th Century Fox still retains the distribution rights to the first two *Star Wars* trilogies, owning permanent rights for the original film *Episode IV: A New Hope*, while holding the rights to *Episodes I-III*, *V* and *VI* until May 2020.

STAR WARS Adds Zac Efron?! WTF! – ETC (7/25/13)

Two of the worst people for the Star Wars universe are probably going to be in Star Wars VII. And we're not just being dicks, we walked around the office to ...

Star Wars Fan Groups: Celebrating a Galaxy Far, Far Away

Ashley Eckstein, voice of Ashoka Tano on Star Wars: The Clone Wars, showcases some amazing Star Wars fan groups from around the world, including the 501st, t...

Is J.J. Abrams Leaving STAR WARS

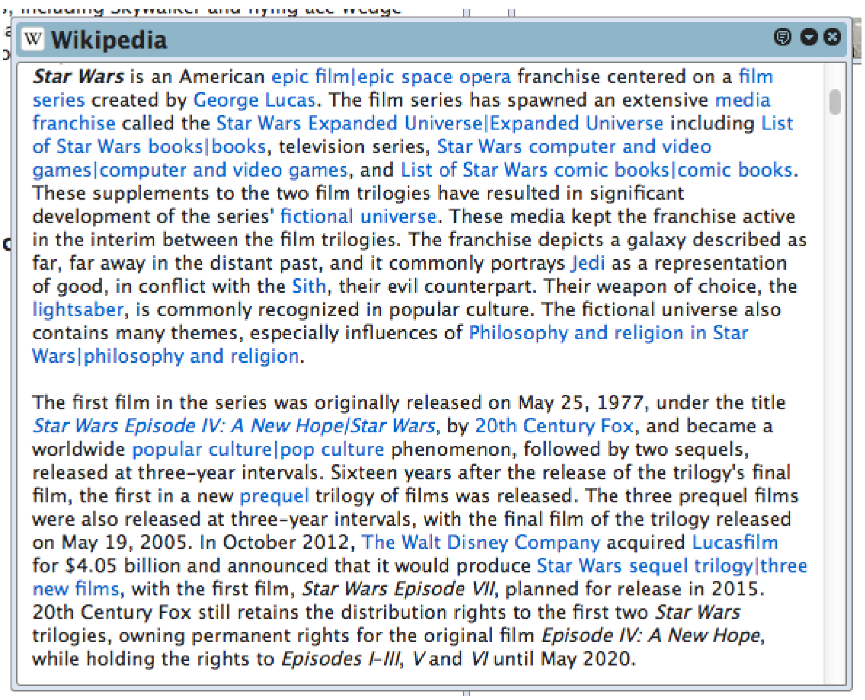
Figure 5.3: *The UI Mashup Dashboard workspace. The left-hand component palette, the menu bar and the platform menu are highlighted.*

available components. On top, a menu bar contains buttons for generic utilities (e.g., save, load, back, forward, composition graph and annotations), the name of the composition, the platform menu that links to the Component Editor, the sharing preferences, the profile info of the logged user and groups management.

5.2.1.1 *Functionalities*

The UI Mashup Dashboard provides several functionalities, which can be grouped in three main macro-functionalities:

- **Component management** includes adding, deleting and executing operations. In the component palette (see Figure 5.4b) all the available components are listed, both Wrapped Components and VI Components. By clicking on an icon, the corresponding component, will be added to the composition. Components are shown to users in a window in the workspace context, which can be moved and resized.

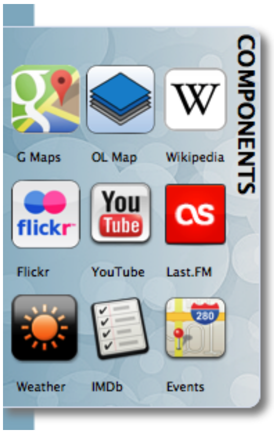


Wikipedia

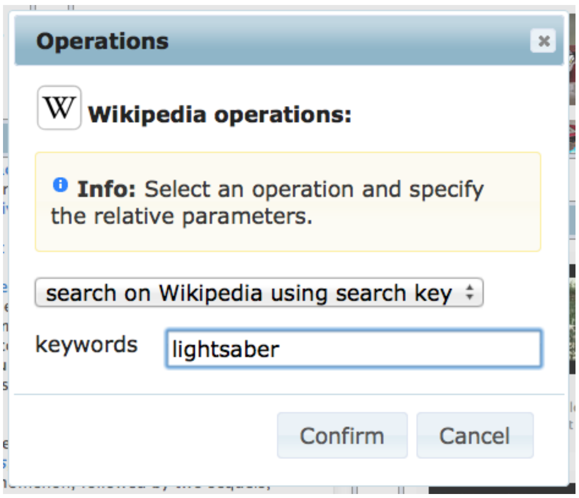
Star Wars is an American epic film|epic space opera franchise centered on a film series created by George Lucas. The film series has spawned an extensive media franchise called the Star Wars Expanded Universe|Expanded Universe including List of Star Wars books|books, television series, Star Wars computer and video games|computer and video games, and List of Star Wars comic books|comic books. These supplements to the two film trilogies have resulted in significant development of the series' fictional universe. These media kept the franchise active in the interim between the film trilogies. The franchise depicts a galaxy described as far, far away in the distant past, and it commonly portrays Jedi as a representation of good, in conflict with the Sith, their evil counterpart. Their weapon of choice, the lightsaber, is commonly recognized in popular culture. The fictional universe also contains many themes, especially influences of Philosophy and religion in Star Wars|philosophy and religion.

The first film in the series was originally released on May 25, 1977, under the title *Star Wars Episode IV: A New Hope|Star Wars*, by 20th Century Fox, and became a worldwide popular culture|pop culture phenomenon, followed by two sequels, released at three-year intervals. Sixteen years after the release of the trilogy's final film, the first in a new prequel trilogy of films was released. The three prequel films were also released at three-year intervals, with the final film of the trilogy released on May 19, 2005. In October 2012, The Walt Disney Company acquired Lucasfilm for $4.05 billion and announced that it would produce Star Wars sequel trilogy|three new films, with the first film, *Star Wars Episode VII*, planned for release in 2015. 20th Century Fox still retains the distribution rights to the first two *Star Wars* trilogies, owning permanent rights for the original film *Episode IV: A New Hope*, while holding the rights to *Episodes I-III*, *V* and *VI* until May 2020.

(a) *A component window. In the top bar the icons allow one to close, annotate component, execute component operations and close the component window.*



(b) *The component palette lists the available components and, clicking on the icons, the components will be added in the composition workspace.*



Operations

Wikipedia operations:

Info: Select an operation and specify the relative parameters.

search on Wikipedia using search key

keywords: lightsaber

(c) *The "execution operation" dialog. Users can select one of the operations exposed by the component and invoke it manually, so changing the component state.*

Figure 5.4: *Key features of components management.*

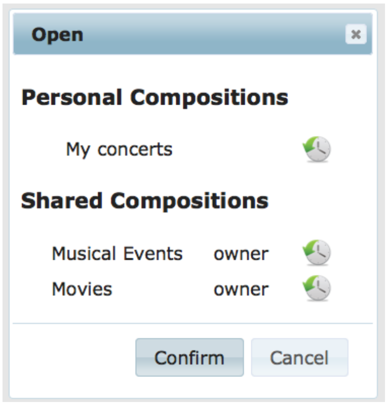
- **Composition management** includes: creation of a new composition, loading of existing compositions, saving the current composition, coupling creation and composition monitoring through the composition graph. The most significant features are reported below:

  - *Loading of existing compositions.* It is possible to load previously saved compositions, both personal or shared (see Figure 5.5a). If a shared composition is loaded, a user can have permission restrictions in the usage of components or in the editing of the composition, e.g., could be prevented users from adding, deleting components or modifying component preferences.

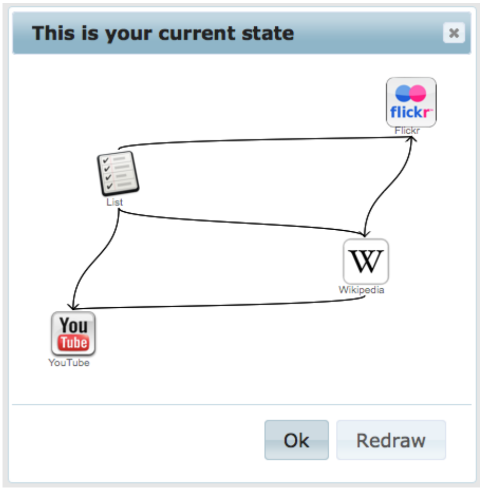
  - *Coupling creation.* In order to create a coupling between two components, users have to drag and drop the top left icon of the component window on another component window. If those components are compatible (a colored border indicates the compatibility, green if they are compatible and red if not) the dialog in Figure 5.5c is shown. From this dialog users create the coupling by selecting, from two drop-down lists, the event and the operation involved in the coupling. Even in this case a green-red color code indicates if the selected event (or operation) is compatible with the operation (or event) that the user has been selecting. This coupling mechanism hides to users the complexity of the underlying publish-subscribe pattern.

  - *Composition graph.* In order to control the composition, especially the created couplings, it is necessary to provide a mechanism to represent the composition schematically. We therefore provide a graph-based visualization, where components are the nodes and couplings are the directed edges. Figure 5.5b shows an example of composition graph. Going over the edges with the mouse pointer, a tool-tip shows details of the coupling, i.e., the event and the operation the edge is referring to.

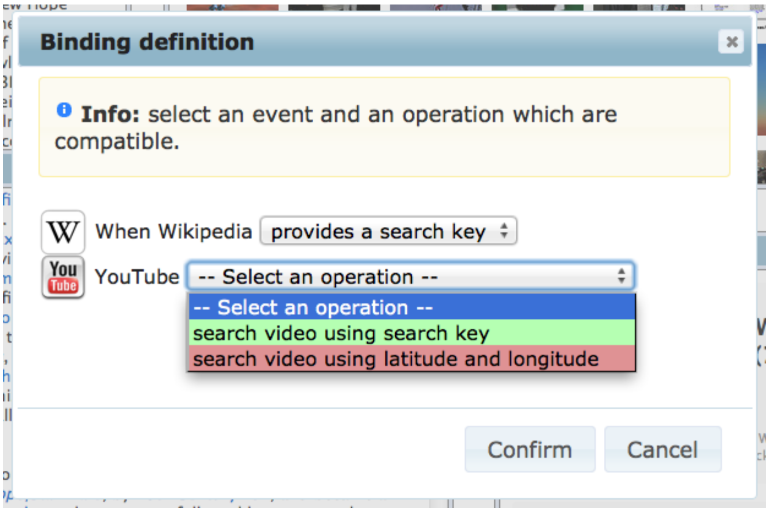
  - *Saving a composition.* It is possible to save the composition. For each component, it is possible to keep track of the current status, i.e., to save the result of operations that have been performed on it, in order to re-create the same state when the composition is opened in the future.



(a) *Dialog box for composition loading: it is possible to open compositions, whether shared or not, and also to select previous versions of a specific composition.*



(b) *The composition graph represents the composition as a graph where components are the nodes and the created coupling are the edges.*


Binding definition
Info: select an event and an operation which are compatible.
When Wikipedia
provides a search key
YouTube
-- Select an operation --
-- Select an operation --
search video using search key
search video using latitude and longitude
Confirm
Cancel

(c) *Dialog for the creation of couplings, users have to select, from two drop-down menus, which event they want to bind to which operation. Once the coupling is created, when the event on the first component is raised, the bound operation on the other component is executed.*

Figure 5.5: *Key features of composition management.*

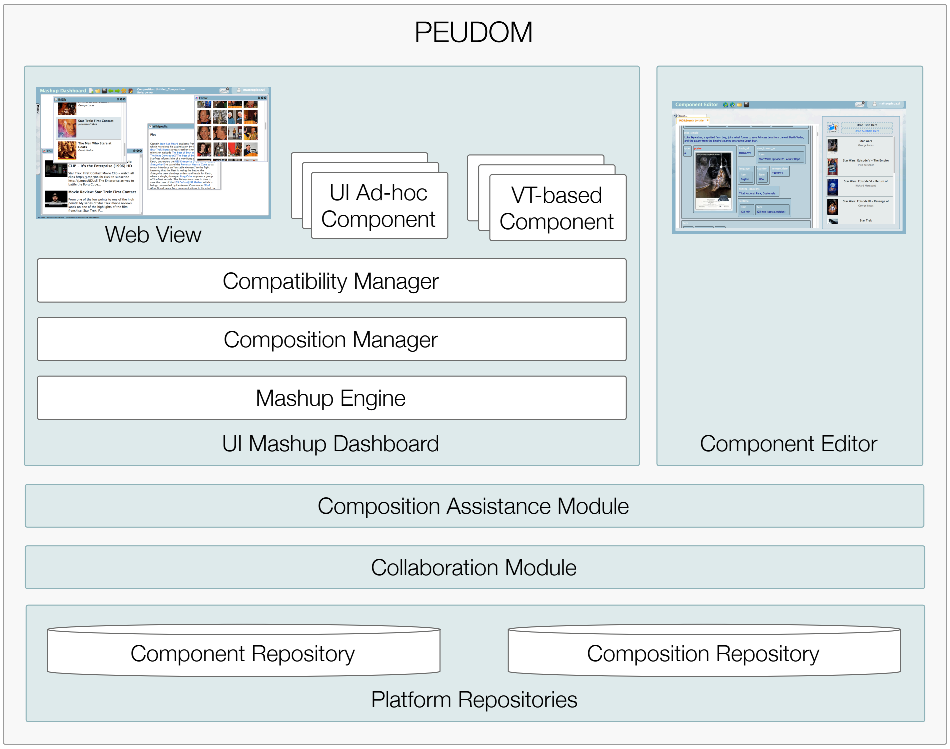


Figure 5.6: *The architectural modules of UI Mashup Dashboard.*

5.2.1.2 *Architecture*

As represented in Figure 5.6, from an architectural point of view, the UI Mashup Dashboard is composed by several modules that implement the functionalities described in the previous section.

- *Web View* is the main view of the platform. It contains the workspace and all its sub-views described above, i.e., the component palette, the coupling dialogs and the composition graph.

- *Compatibility Manager* is the module that, every time a coupling has to be defined, checks the compatibility between two components: an event-publisher component and a condidate subscriber component. Once two components are selected, it establishes the compatibility among their events and operations. Compatibility rules are stored in a *Compatibility Matrix*, stored in the *Composition Assistance Repository*. Each entry of the matrix is a tuple, $< c_s, e, c_t, o >$, where c_s is the source component, i.e., the component that raises the event e, and

c_t is the target component where the operation o is executed. Such a matrix is updated every time a new component is added into the Component Registry.

- *Composition Manager* provides all the functionalities that are related to the composition, i.e., creating new compositions, saving compositions, loading compositions, adding and removing components, displaying the composition graph and creating couplings.
- *Mashup Engine* is the module that provides the synchronization among components, handling events and operation calls. This module is better described in Section 5.3, being it related to the executive of the created mashups.

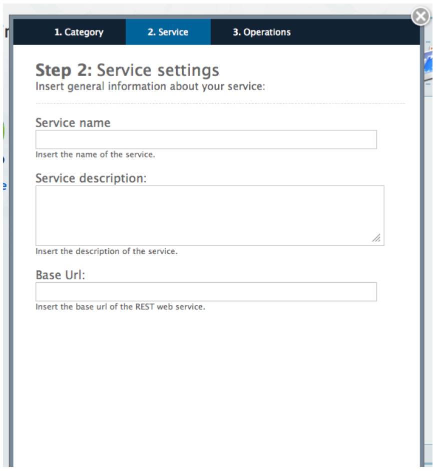
5.2.2 Component Editor

The *Component Editor* allows end users to register REST services in the platform and to query them in order to use their data to develop VI Components (see Chapter 4). Components can be exported as platform-independent XML-based descriptors, thus interpreted and executed on the *UI Mashup Dashboard* or in standalone applications, e.g., mobile apps, coded according to the target device technology.

5.2.2.1 Functionalities

The functions offered by the *Component Editor* can be clustered in three groups:

- **Service management** includes registration of services and service selection. In order to use services for component creation, services must be registered in the platform (see Figure 5.7a). The registration of a service means creating a description that can be used to select and to query it. The description include the name of the service, the URL endpoint and the possible queries and their parameters. Thanks to this description it is possible, also for end users, to make requests (or queries) to services. Once a service is registered it is possible to select it in order to create components.
- **Component management**. The Component Editor makes it possible to save and load components. The saving process produces a descriptor that is used to export the component also on other devices of platforms.


1. Category | 2. Service | 3. Operations

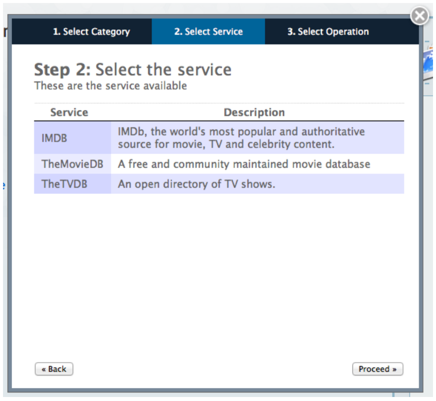
Step 2: Service settings
Insert general information about your service:

Service name
Insert the name of the service.

Service description:
Insert the description of the service.

Base Url:
Insert the base url of the REST web service.

(a) *The dialog for the registration of services.*

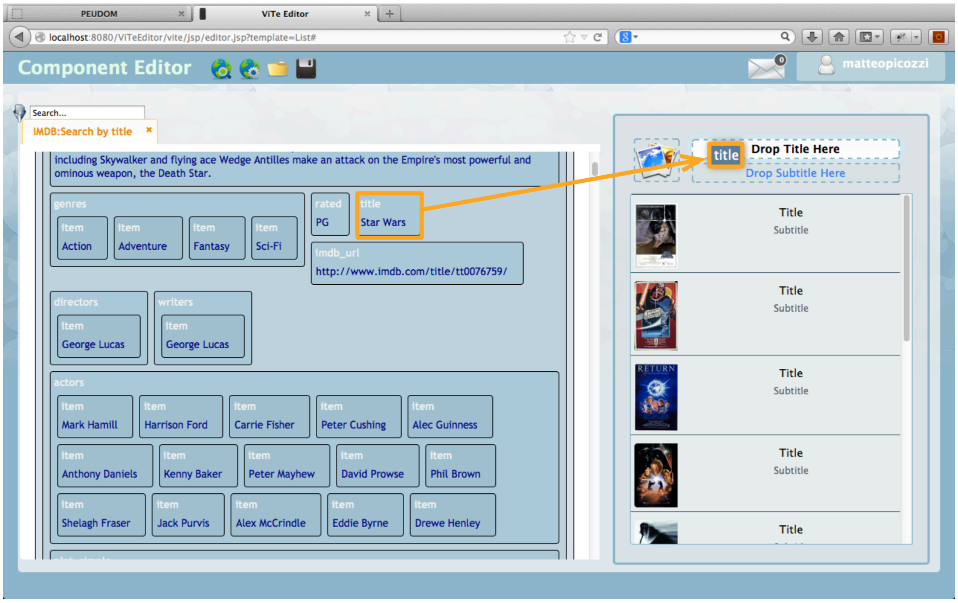

1. Select Category | 2. Select Service | 3. Select Operation

Step 2: Select the service
These are the service available

Service	Description
IMDB	IMDb, the world's most popular and authoritative source for movie, TV and celebrity content.
TheMovieDB	A free and community maintained movie database
TheTVDB	An open directory of TV shows.

« Back | Proceed »

(b) *List of services for the category "Cinema". By selecting a category, it is then possible to select the desired service.*


PEUDOM | ViTe Editor

localhost:8080/ViTeEditor/vite/jsp/editor.jsp?template=List#

Component Editor | matteopicozzi

Search...
IMDB:Search by title

including Skywalker and flying ace Wedge Antilles make an attack on the Empire's most powerful and ominous weapon, the Death Star.

genres: item Action | item Adventure | item Fantasy | item Sci-Fi
rated: PG
title: Star Wars
imdb_url: http://www.imdb.com/title/tt0076759/
directors: item George Lucas
writers: item George Lucas
actors: item Mark Hamill | item Harrison Ford | item Carrie Fisher | item Peter Cushing | item Alec Guinness | item Anthony Daniels | item Kenny Baker | item Peter Mayhew | item David Prowse | item Phil Brown | item Shelagh Fraser | item Jack Purvis | item Alex McCrindle | item Eddie Byrne | item Drewe Henley

title
Drop Title Here
Drop Subtitle Here

Title
Subtitle

Title
Subtitle

Title
Subtitle

Title
Subtitle

Title

(c) *When dragging a data item from the data panel to the visual template, the item is mapped onto the corresponding visual renderer.*

Figure 5.7: *Key features of the Component Editor.*

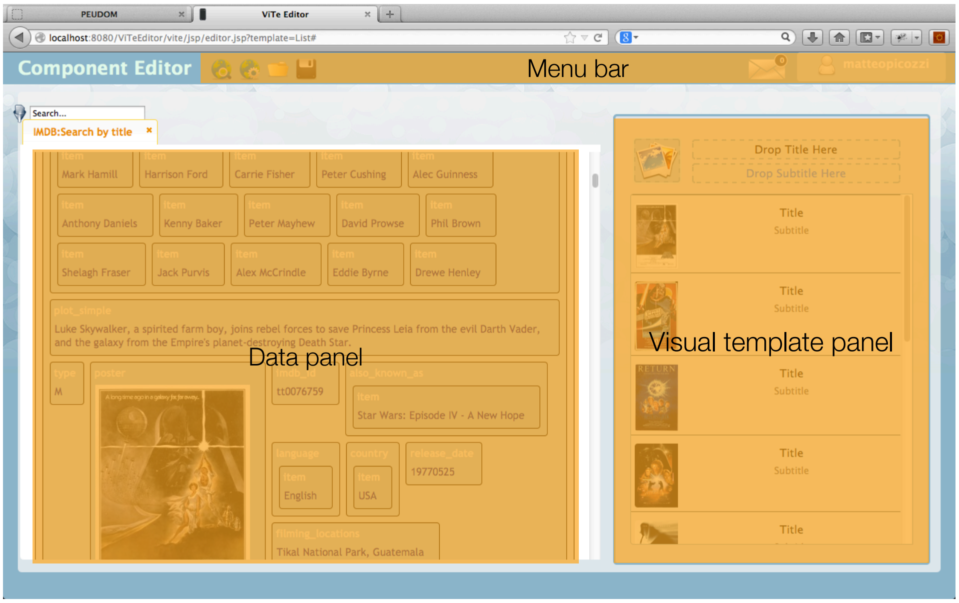


Figure 5.8: *A screen shot of the Component Editor. On the left the data panel is highlighted, while on the right it is placed the visual template panel.*

- **Component editing** includes the possibility to invoke queries on data components, also specifying parameters value, and to map data onto a selected visual template, according to the visual integration paradigm illustrated in Chapter 3.

5.2.2.2 Architecture

Figure 5.9 shows the Component Editor architecture and highlights it main modules.

- *Web View* is the main view of the environment and contains the menu bar and the elements that allow to select services and templates and to map the data onto the visual renderers.

- *Service Manager* offers support for querying REST services, displaying the retrieved results in a visual format, and visually defining selection and projection queries over such results.

- *Visual Mapping Manager* translates the visual mapping actions into composition rules included in an XML-based component schema.

- *Component Manager* manages all the aspects related to loading and saving components.

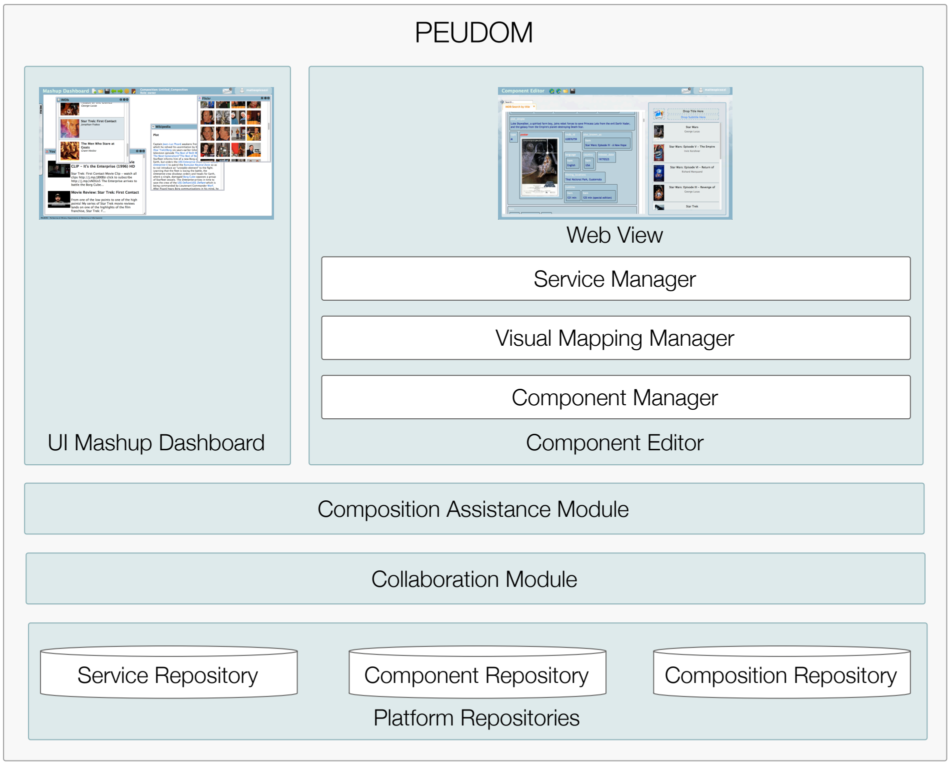


Figure 5.9: *The Component Editor architecture.*

5.3 Execution environments

In this chapter we describe the engines managing the mashup execution, in particular the synchronization among multiple UI Components and the execution of the VI Components on different devices.

5.3.1 UI Mashup Engine

As described in Section 4.1.1, in our mashup platform, UI Component synchronization complies with an event-driven, publish-subscribe paradigm. Some architectural modules are in charge of capturing events and propagating them to govern synchronization at the UI level.
The events occurring during mashup execution are managed by the *UI Mashup Engine* through the *Event Broker*, shown in Figure 5.10. Such events can be synchronization events or composition events. The synchronization events are managed by the *Execution Handler* that, based on the composition schema, it acts as an event bus: it listens and handles the events

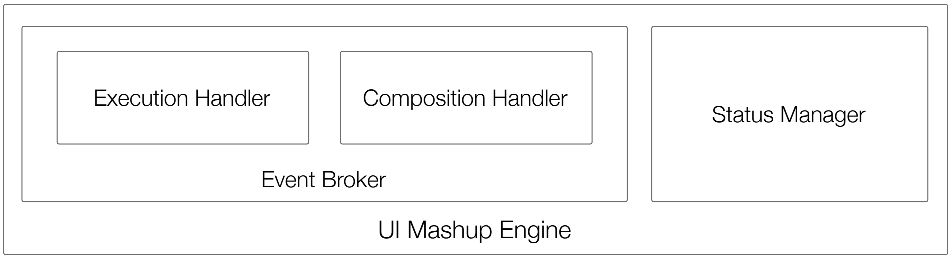


Figure 5.10: *UI Mashup Engine architecture.*

raised by the interaction with each single component, and activates the subscribed operations as prescribed by the listeners in the composition schema. The *Composition Handler* manages composition events. In particular, it automatically translates the addition of a component or the creation of a new coupling, updating the current composition model accordingly. The Composition Handler dispatches composition events to the *Status Manager*, a module in charge of maintaining a mashup state representation. Combined with the composition model, the mashup state is useful to recover a previously defined mashup for a later execution, but especially to let users monitor their composition and modify it on the fly. State variables, related to UI Components or UI Mashups, are: default or specific parameter values (e.g., the value of a parameter for querying a data source), layout properties (e.g., the position of a component in the workspace and the colors used to show values on a chart) or any other property that the user can set to control the component data and appearance.

5.3.2 Multi-device execution engines

The VI schemata created through the Component Editor, can be interpreted and executed on different platforms.
Multiple execution engines can be developed for the different devices. In the context of this research we implemented execution engines for the PEUDOM Web platform, and for Android smart-phones, and for multi-touch screens – based on the MultiTaction technology. All these engines comply with the same architecture, although they are implemented through the native technology on the target devices.
Figure 5.11 illustrates the main architectural elements. The *Schema Interpreter* parses the application schema created at design time. Hence, *Data Manager* is able to compose the query through *Query Manager*. When the response is provided, *Data Extractor* selects data from the result set ac-

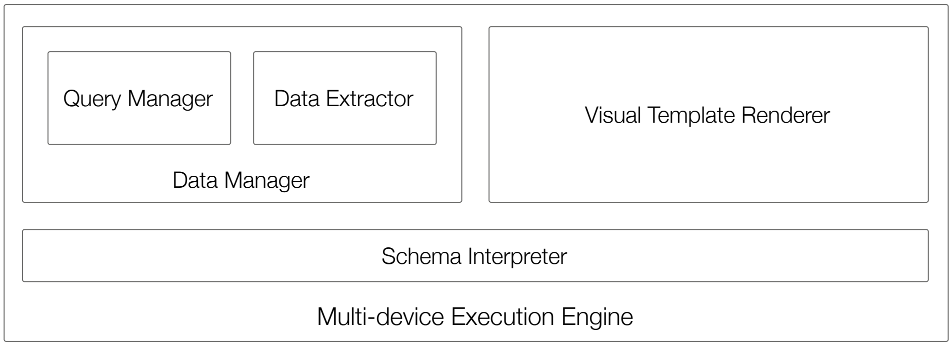


Figure 5.11: *General architecture of execution engines.*

cording to the application schema and, finally, *Visual Template Renderer* renders data in the UI.
Especially when more than one service is included in the component schema, it is possible that some of the items retrieved are duplicated. The Data Manager module also manages the detection and elimination of duplicates, by means of the Soundex similarity algorithm [8], and the fusion of corresponding data attributes into the global sub-template (see Section 4.2.1).
In the following, two examples of execution environments are presented but the device independence of the generated schemata allows to use such a VI Components also on other devices, e.g., large displays or multimedia interactive blackboards.

5.3.2.1 Web execution environments

Web execution environments are included in Web applications and it is possible to have standalone execution environments or integrated execution environments. Standalone execution environments are the ones that just display the contents extracted by the interpretation of the exported schema, while integrated execution environments are integrated in a container Web application that allows to use these contents for synchronizations with other contents or services.
The Web execution environments are developed in form of component wrappers: there is a wrapper for each visual template that is able to read, interpret and display data in a component window. Once a VI Component created trough the Component Editor is saved, the component is instantaneously available for composition and its icon appears in the component palette. When Wrapped Components are used, it is possible to predict the compatibilities among their events and operations but, when components

are instantiated at runtime, it is more difficult to assess the compatibility. To solve this issue, as first solution, for each VI Component we expose general purpose events and operations, e.g., the `provideSearchKey` event and the `searchByKey()` operation. The events are generated by user interactions with the UI like an item the selection, caused by a click on a visual renderer. We are planning for the future some more sophisticated mechanisms like coupling among operations of the target component and their parameters, taken from the set of data provided by the source component.

5.3.2.2 Mobile execution engines

An example of mobile execution engine is MobiMash [21]. It is able to execute a mashup composed of VI Components, Wrapped Components, plus “local” components managing the coupling with device functions (e.g., the agenda or the map navigator). The users download from the platform repository the VI Schema. The execution engine masters the instantiation of the VI Component as described above. In particular, if Android is the target OS, the application schema is translated into the Android layout markup language that will be used by the OS for the generation of screens – the so-called *activities*.
According to the target device implementation of the mashup application, it is possible also to couple the data and function provided by the execution environment to other general purpose services and invoking, by means of proper API wrappers, synchronized operations exposed by such services.

5.4 Composition Assistance Module

The techniques for quality assessment described in Section 4.3 have also been integrated in PEUDOM [19, 23]. Figure 5.12 illustrates the main architectural components, with particular emphasis on those in charge of executing the quality-based ranking algorithms.
In the platform back-end, the *Component Repository* stores the component descriptors specifying both the functional and quality properties (see an example in Listing 4.3) of components that are exploited by our recommendation paradigm. We defined a family of algorithms that provide recommendations on the basis of a two-phases process. The structures used to compute recommendations are updated off-line, thus at real-time the algorithm use them to compute recommendations. Every time a new item is added to the platform it is necessary to re-compute these structures. The functional properties augmented with semantic annotations are exploited to compute

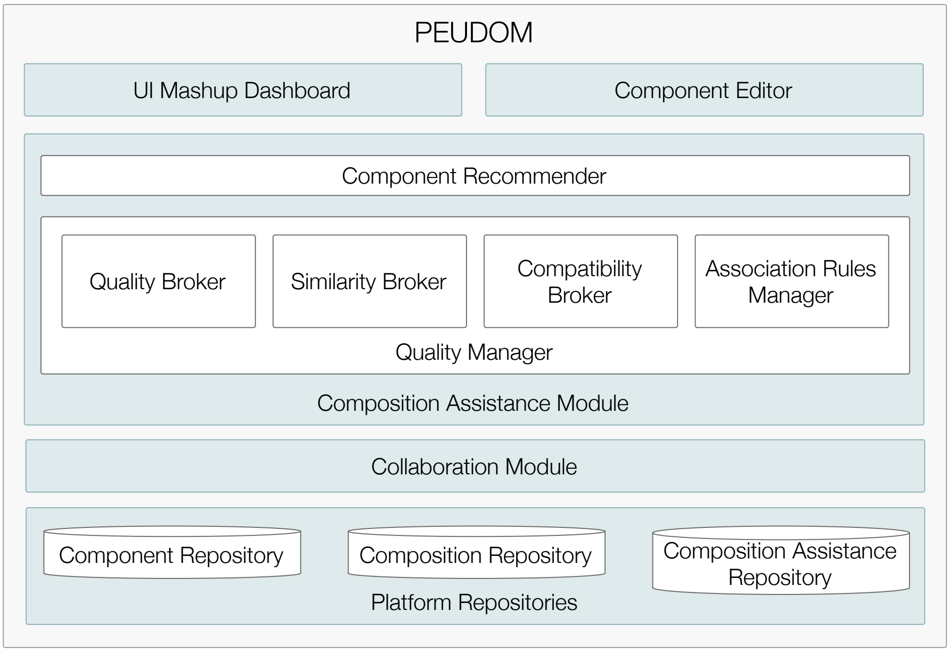


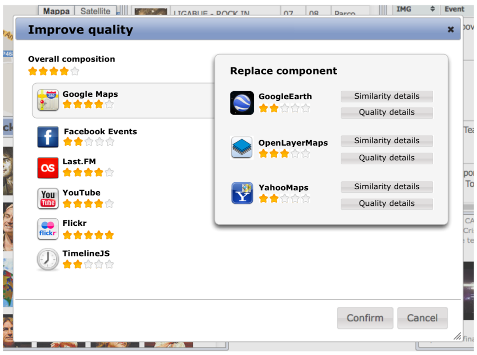
Figure 5.12: *Composition assistance module architecture.*

the *compatibility and similarity matrices* contained in the Composition Assistance Repository. *Association Rules* reflecting community-based composition practices are also computed off-line periodically, starting from the data crawled from mashup repositories, publicly available (e.g., programmableWeb.com) or local to the adopted mashup platform.

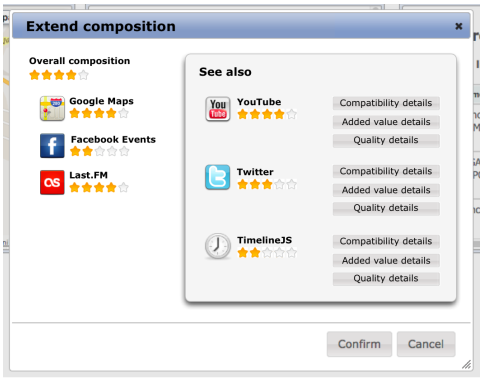
Our recommendation model is indeed composed by the Compatibility and Similarity matrices and by the Association Rules. Those matrices are used to assess the compatibility and the similarity among all the components in the repository. A semantic reasoner is used for this purpose[1]. The two matrices are computed at the first use of the platform, and updated every time a new component is added into or dropped from the Component Repository. The quality annotations specified in component descriptors are used to compute *quality vectors*, contained in the Composition Assistance Repository. A quality vector stores the quality measures achieved by computing metrics, such as those defined in the quality model for mashup components reported in [17], starting from the quality annotations.

The *Component Recommender* generates the component ranking. It analyzes the association rules, to discover the component categories to recom-

[1]Our current implementation uses the Pellet reasoner (`http://clarkparsia.com/pellet/`).



(a) *Example of quality-based recommendations for* alternative components.



(b) *Example of recommendations for* additional components *based on the assessment of the perceived quality and added-value dimensions.*

Figure 5.13: *Recommendation windows in the PEUDOM visual editor [16].*

mend for mashup completion, and identifies the components in those categories that are compatible and similar with the components already in the composition. It then exploits the *Quality Broker* to compute the aggregated quality and the added value indexes, based on the analysis of the quality vectors and of the composition model. The result is a ranking of components, based on the quality and the added-value increment that components can give to the composition under constructions.

Figure 5.13a shows an example of a quality-based recommendation for alternative components, i.e., on the right components that can improve the added value of the overall composition are suggested. The window displays on the left the components currently included in the mashup together with an evaluation of their quality expressed in form of stars. Internally, the tool exploits the graph-based representation of such a composition, to identify the existing inter-component dependencies and scoring the composition quality according to the QC metric. If one component is selected, for example Google Maps in figure, the right panel shows a list of components that are similar to Google Maps and compatible with the other components in the composition. These components are ranked by taking into account their capacity to increase the composition quality if replaced to Google Maps. Further details about their quality and their similarity with Google Maps are also given. Quality measures are normalized with respect to a selected scale. Figure 5.13b shows an example of completion recommendation that our tool produces on the basis of the added-value measure. We express measures in a scale from 1 to 5, and visualize them in form of stars. Assuming that the current composition consists of the components listed in the left-hand side of the window, our technique identifies candidate components (reported in the right-hand panel) that can increase the composition value. In the example, the suggested components would provide content of different nature, i.e., multimedia data (YouTube) and user-generated contents (Twitter), and an additional timeline visualization (TimelineJS) that in the example mashup would support the temporal characterization of the music events.

5.5 Collaboration modules

Collaborating for mashup development in scenarios for end user development means exploiting the co-creation of information spaces as a means to communicate and share information and ideas with the goal of developing new knowledge.
In the previous chapters we have described the adopted collaboration mechanisms (see Section 3.2.6) and the models on which they are based (see Section 4.4). Taking into account these models and mechanisms, we clarify in the next of this section the sharing policies and the architecture of the platform collaboration modules.

5.5.1 Sharing resources

We discuss below what is possible to share in our platform, the sharing polices and the roles of users with respect to the adopted sharing polices.

***What* to share.** According to the elements that characterize our composition paradigm, when working on a mashup a user can share different kind of objects:

- *Registered services*: a user can share basic services (e.g., REST Web services), that can be invoked from her/his personal profile and that show a common utility for a group of users for the creation of meaningful resources on which collaboration can be established. What is actually shared in this case is the *service descriptor*, which specifies the information needed to invoke the service, e.g., its URI) and to execute basic queries to retrieve an initial data set on which to coordinate (e.g., search keys and parameter values). Such descriptors is made accessible to the sharing users in their *Component Editors*. Depending on the roles assigned to the sharing users, service descriptors can then be jointly modified, for example to refine the result set through new and expanded queries. In our approach sharing services – and their descriptors – is fundamental: services are indeed the building block to compose any other resource, e.g., VI Components through the Component Editor and also mashups through the Mashup Dashboard.

- *VI Components*: besides the basic services, also resources packaged through the *Component Editor* by integrating multiple data sets into visual templates can be the object of sharing. Even in this case a *component descriptor* is shared; assuming the presence on the user device of an execution module, the component descriptor specifies the logic to query the basic services, integrate the result sets into UI Templates, and then dynamically create and instantiate the component UI on the executing device. Depending on users' roles, components can be just used as they are within the UI Mashup Dashboard for creating or modifying mashups, or they can be jointly modified within the Component Editor.

- *UI MAshups*: Finally, the mashup resulting form component integration within the UI Mashup Dashboard can be shared, to allow other users to access the integrated content, or to let them participate to the co-creation and evolution of the mashup.

***Who* to share with.** Sharing of the previous objects can be allowed to different actors, both single users and groups. Each actor being assigned with one or multiple roles among those reported in the following:

- `owner`: the user who initially creates the object to be shared; s/he has all the permissions on the object and it cannot unsuscribe from the shared object.

- `super-user`: a user with all the privileges of the owner, with the difference that s/he can leave the sharing.

- `user`: s/he does not have the full permissions, i.e., s/he can perform only a reduced set of modification actions, for example s/he cannot add or remove objects contained in the shared ones, but can only modify some of her/his configuration settings.

- `viewer`: a user who can access and use the object but cannot modify it.

In order to clarify in detail roles and sharing policies we have to deeply describe what can be done with the sharable objects. Each kind of object (services, component and composition) must be handled in a different way except for the possibility to share them or to leave the object sharing. Table 5.1, Table 5.2 and Table 5.3 describe in detail our sharing polices for the three kinds of considered objects.

	Share	**Leave sharing**	**Modify params**	**Use**
owner	✓	✗	✓	✓
super-user	✓	✓	✓	✓
user	✗	✓	✓	✓
viewer	✗	✓	✗	✓

Table 5.1: *Sharing policies for registered services*

	Share	**Leave sharing**	**Modify param values**	**Modify mapping**	**Use**
owner	✓	✗	✓	✓	✓
super-user	✓	✓	✓	✓	✓
user	✗	✓	✓	✓	✓
viewer	✗	✓	✗	✗	✓

Table 5.2: *Sharing policies for components*

	Share	**Leave sharing**	**Save as**	**Save**	**Save status**	**Add/Rem components**	**Add/Rem couplings**	**Execute operation**	**View**
owner	✓	✗	✓	✓	✓	✓	✓	✓	✓
super-user	✓	✓	✓	✓	✓	✓	✓	✓	✓
user	✗	✓	✓	✗	✓	✗	✗	✓	✓
viewer	✗	✓	✗	✓	✗	✗	✗	✗	✓

Table 5.3: *Sharing policies for mashup compositions*

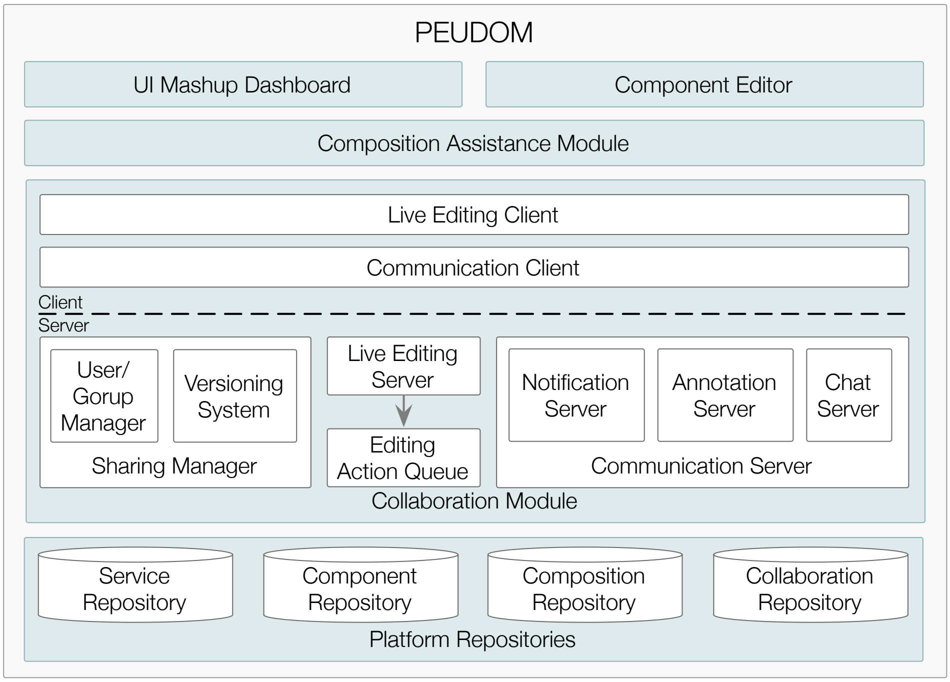


Figure 5.14: *Collaboration module architecture.*

In particular, *services* can be modified (in terms of base URL, operations and their parameters) or used, i.e., chosen for a component creation and their result set items could be mapped onto visual renderer elements.

Components can be modified (in terms of values of services parameters and mapping onto visual render elements) or used, i.e., exported trough their schema or used in a mashup composition.

The possible actions on *compositions* are: *(i)* view/use the composition without modifying query parameters, just interacting with the visible components; *(ii)* execute operations or modify queries; *(iii)* add or remove couplings; *(iv)* add or remove components; *(v)* save the status of the composition, in terms of executed queries and results retrieved by components; *(vi)* save the composition as an increment of version, like a commit in a versioning system; *(vii)* save the composition as a new branch of a versioning system, in this case, the user who saves the composition becomes its owner even if the permission level of the components already included is kept the same and is not promoted to the owner level.

5.5.2 Architecture

As highlighted in Figure 5.14, additional clientside modules, together with a server-side layer, take care of managing the persistence and the co-evolution of schemas, as required by the collaborative functions illustrated above. While the client manages the interaction with users capturing their actions and updating the view as a response of other users' actions, the server manages resource sharing and communication in both its synchronous and asynchronous aspects. To support multiuser access to resources, the server also hosts the required schema repositories (for registered services, components and composition) and databases (for example to manage user access to resources in different versions).
We can group the collaboration modules in three clusters:

- **Sharing Manager** includes the *User/Group Manager*, which is in charged to manage users and the subscriptions to groups of users, and the *Versioning System*, which manages the compositions and components schema versioning.

- **Live Editing Module** manages the live editing aspects by capturing relevant user actions on instances of the mashup composition, and propagating them to a server module and involves *Live Editing Client*, *Live Editing Server* and *Editing Action Queue* modules.

- **Communication Module** is composed by Communication Client and Communication Server and contains the modules for notification, annotation and chat management.

For live editing, in order to facilitate the appropriation of the paradigm by the end users, we adopt a technique in use in other collaboration platforms (e.g., Google Drive), notifying every relevant modification on a mashup execution to any other composition instance. In particular, the Live Editing Client captures the elements describing the required modification and propagates them to the Live Editing Server, which takes care of the composition schema evolution by maintaining a representation of the distributed editing actions: every editing session on a given mashup composition has an associated Editing Action Queue.
As illustrated in Figure 5.15, any active instance of a mashup composition periodically queries the Live Editing Server, to know whether new actions, generated by other executions, are available. Any action propagated to the different clients is represented as a pair `<modifiedObject, notification>`. The first element represents the argument of the action

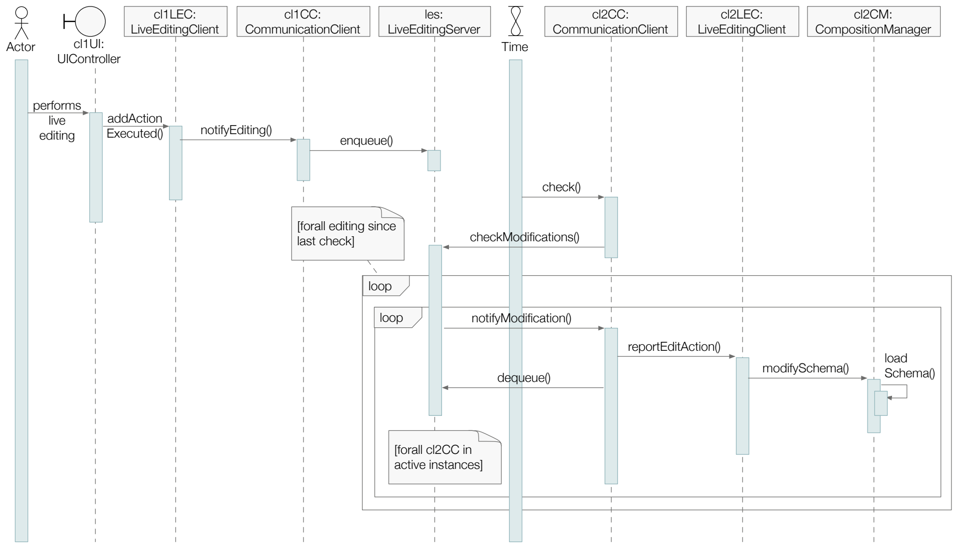


Figure 5.15: *Sequence diagram reporting the communications following a user action on a client and the propagation of such action to the other clients involved in a live editing session.*

(e.g., a component, a coupling, a query parameter) and all its properties. The second represents some metadata (e.g., the ID of the user who performed the action), needed for notifying the change. The modifiedObject properties have effect on the composition schema; the notification element is used by the Live Editing Client to highlight visually the action. The Live Editing Client interprets the received actions and triggers events to let the Composition Manager accordingly modify the composition schema and reload the composition schema. Changes are highlighted in the workspace, based on the notification meta-data. To reflect composition changes with minimal delay, the Live Editing Client periodically checks the composition status representation through the Live Editing Server, to verify whether some actions must be loaded and rendered within the mashup instance. The sequence diagram in Figure 5.15 illustrates communication between clients and server for publishing (client `cl1`) and retrieving (client `cl2`) edit actions.

The Communication Client and Communication Server modules manage aspects for synchronous (e.g., by chat and notifications) and asynchronous (e.g., by annotations) communication.

Annotations are persisted in an annotation manager and are retrievable by authorized users when uploading the original document. Annotations can also be presented to users in a live form. To this end, the instance of the composition can inquire with the server if there are annotations, for some of the nodes currently present in the instance composition, whose timestamp is more recent than the last check for annotations. This form of synchronization of all the active mashup composition instances implies a "distributed" representation of the composition schema, which is maintained at each client. Server-side management of the action queue ensures synchronized evolution of all the active composition instances.

CHAPTER 6

Validation of the proposed approaches

In order to validate our approach, we assessed its portability and performance on different execution environments. Since our approach stresses the importance of providing composition paradigms adequate for the end users, we also conducted some user studies to validate the suitability of the design environments and the composition paradigm with respect to the capability and characteristics of laypeople, not expert in computer science in service composition. In performing these studies, we followed the incremental flow described in Chapter 1. Finally, to complete the validation of the proposed approaches, a field study on multi-device execution environments and a test on assistance mechanisms are presented.

6.1 Performance evaluation

In order to assess the feasibility of our approach from a technological point of view, we conducted some experiments to evaluate the performance of some critical procedures, like the event-driven, publish-subscribe mashup composition and execution of the UI mashups created through the Web dashboard, and the data-fusion on demand at the basis of the fruition of our multi-device UI components. We measured the complexity of such procedures in terms of time and memory consumption on the desktop Web platform. To conduct those experiments we used Firefox 24.0 on a MacBook Pro with OS X 10.8.5, 2.5 GHz Intel Core i5 CPU and 8GB of DDR3 RAM. For the data fusion on demand we also evaluated its performance on mobile devices: a Samsung Galaxy S II with a dual-core 1.2 GHz ARM Cortex-A9 processor with 1 GB of RAM.

6.1.1 Mashup composition and execution

Our mashup composition and execution is executed client-side. We decided not to use a server-side mashup engine because mashups are interaction-intensive applications, whose execution generally generates a high number of requests to the server in order to instantiate all the included components and synchronize their behaviors. However, a client-side implementation could be critical because of the lower performances of clients if compared to servers. In order to demonstrate that our approach is lightweight and client-side execution is feasible, we therefore conducted some experiments. We do not consider recommendations and collaboration since these are extensions of the platform we are still working on from the technological point of view. At the time of writing, we just evaluated the efficacy and effectiveness of such mechanisms with respect to the user performance.

6.1.1.1 Testing scenario

The performance test of the Web dashboard was conducted executing 4 scenarios with increasing complexity where each scenario is included in the following one:

1. The first scenario was about the inclusion of a VT UI Component called IMDb, which gets data from an unofficial API of IMDb[1] called *MyMovieAPI*[2], and the Wikipedia Wrapped Component. The two components were coupled and a new search was performed on the IMDb component with search key “ocean”. Than the first item of the returned list (“Ocean’s Eleven”) was selected raising the synchronization with Wikipedia and the link “George Clooney” was clicked in Wikipedia.

2. The second scenario included in the composition also the Flickr and YouTube Wrapped Components, coupling them with IMDb compo-

[1]`http://www.imdb.com`
[2]`http://mymovieapi.com`

nent. After a new search on IMDb, all the other three components were synchronized and there was a new interaction with Wikipedia.

3. The third scenario involves the same components with the same couplings but had a more complex interaction phase that raised a higher number of events.

4. The last scenario, that involves all the components and actions of the previous ones, coupled also the Wikipedia `provideSearchKey` event to Flickr and YouTube `searchByKey()` operations. In this way, when a link is clicked on Wikipedia, both Flickr and YouTube get synchronized according to the Wikipedia selection.

6.1.1.2 Procedure

We were interested in measuring times and memory consumption as performance indicators of our approach. In particular we captured *times* by instrumenting the JavaScript code with the Firebug[3] utilities that measure the time intervals between the call of `console.time()` and the call of `console.timeEnd()` functions. We measured time intervals for the component initialization, which occours when components are added into the composition, the main search operations of the components involved in the test scenario and the component synchronization when, on the raising of an event, the coupled operations must be retrieved and called. We were able to conduct 50 experiments thanks to the Selenium Firefox plugin, which allowed us to record and replay the execution of our Web application by recording the interactions with the browser. The tests were then grouped in test suites.
For the memory consumption we collected data from the Mac OS Activity Monitor and from the process status (`ps`) OS utility.

6.1.1.3 Results

Table 6.1 describes the results, in terms of computation times, of the experiment on the performance of mashup composition and execution. Since in our approach there is an intermixing between design and execution, in this evaluation we measured also operations that belongs to the design activity, i.e., `addComponent()`. It is possible to identify four clusters of measured operations. The first group is the *setup* of the components that includes the creation of the container from the Composition Manager and the `init()` function that is dependent on the implementation of components.

[3]Firebug is a Firefox plugin used to test and debug Web applications. `http://getfirebug.com`

Operation	Average time [ms]	# of experiments
addComponent()	130.62	185
init()	31.44	130
loadSchema()	8.20	55
displayGlobalView()	19.42	113
displayDetailView()	3.84	57
searchByKey()	830.78	701
triggerEvent()	15.72	999

Table 6.1: *Results of the experiments about mashup composition and execution.*

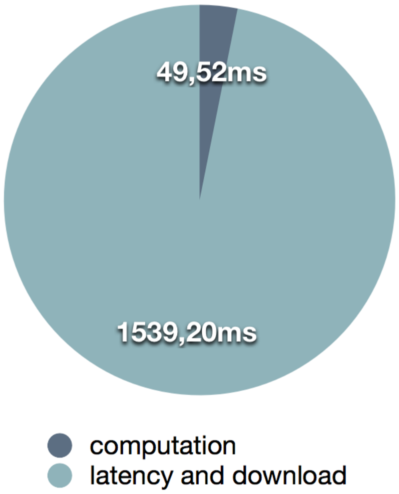


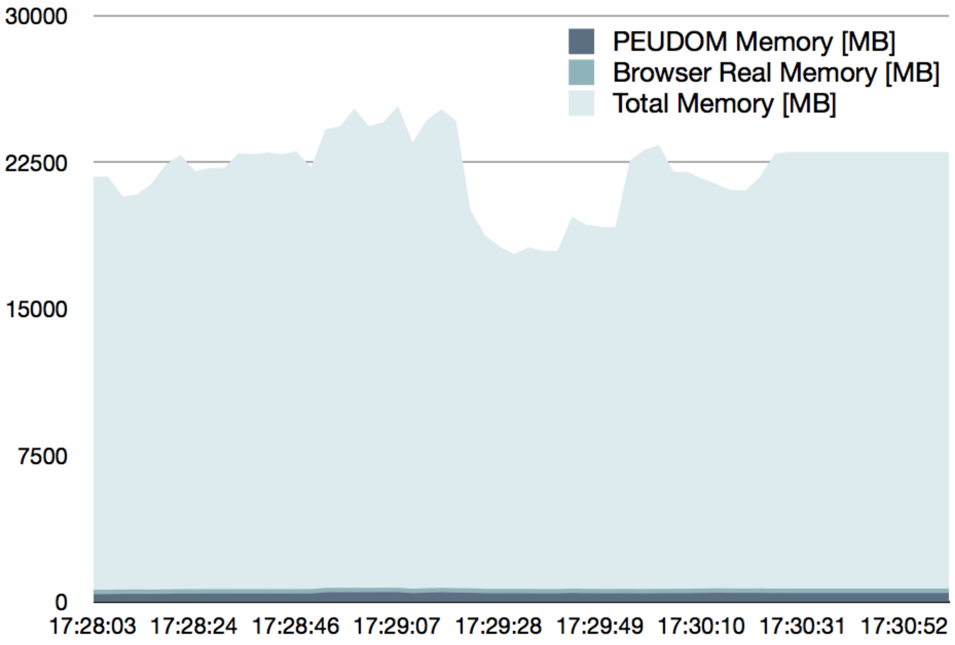
Figure 6.1: *Relation between latency and download with respect to computation times.*

The second group is about the *VI Components* in the phases of loading the component schema, displaying the global view and then the detail view, respectively. The third group is about the operations used to *retrieve data from services*, which include the service latency, the data download and the computations to display the data in the component container. The last group is about the `triggerEvent()` function that is the function of the Mashup Engine that is called by components to trigger the synchronization events.
Note that, in Table 6.1, the functions have a different number of executions; this is because they are called a different number of times in the execution of the testing scenarios. As expected, the most time-consuming function is `searchByKey()`. In fact this function implies the latency of the service and the download time of data, this last activity taking most of the time. In Figure 6.1 we show the percentage of time consumed by the aggregation of latency and download times over the computation time spent by the

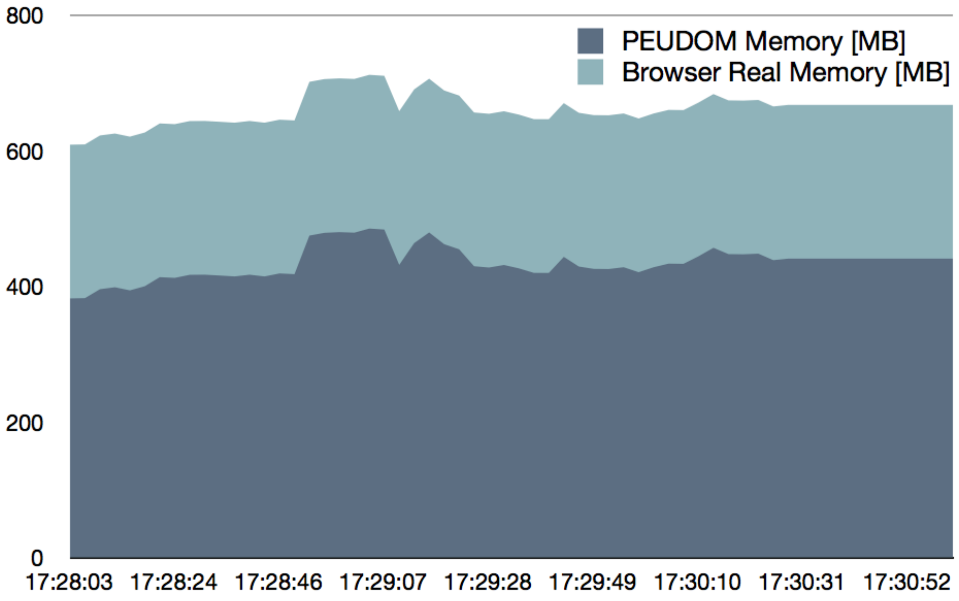
platform in order to render the data. It is evident that the latency and download time constitute almost the totality of the `searchByKey()` execution times while the computation time to enact the operation is irrelevant.
Another dimension we measured is the *memory consumption*. In Figure 6.2a we report the memory consumption of our platform with respect to the memory consumption of the browser and of all the active processes. Figure 6.2b shows instead the memory consumption of the browser with respect to PEUDOM during the execution of the testing scenarios.

6.1.2 Data-fusion

Another critical activity in our approach is data fusion, which is at the basis of content integration in the VI Components. As explained in Section 4.2.3, data fusion must compare elements from two different services and fuse the similar elements. To evaluate this aspect we measured the times for the whole Algorithm 1 considering the worst case in which all the elements, from all the services, should be compared.
For the experiment we considered 2 services (EventBrite and Eventful), which provide information about events of different kinds, and queried them using the search key "music". EventBrite returned 225 events and Eventful returned 6201 items, of which we considered the top 400 elements. In fact, in a real scenario, the user would not be able to scan more than few hundreds of elements. Thus the number of elements considered for the experiment in a sense represents an extreme case.
In order to measure the time intervals, we used the same Firebug utilities adopted for the time performance experiment. In order to evaluate the scalability of our algorithm we ran the experiment with an increasing number of items in the set of the elements to compare.
In Table 6.2 the results of the experiment are reported. We conducted 16 experiments by increasing the number of considered items, with an increment of 25 items up to the maximum number of retrieved items per service. These items are used in the creation of the selected-item set and comparison set. The *selected-item set* is the set of items selected by the user, which are used to search for duplicates in the comparison set. The *comparison set* is the set of all the elements that do not belong to the services of the selected items that may contain duplicates (we assume that there are not duplicate items provided by the same service). Note that, in the execution of the data fusion on-demand algorithm, the selected-item set has cardinality equals to 1 because the user select only one item. As explained above, for the experiment we are considering the worst case, in which all the items of a service



(a) *System memory consumption.*



(b) *Comparison of browser and PEUDOM memory consumption.*

Figure 6.2: *Memory consumption during the execution of the testing scenarios.*

Experiment #	Comparison-set items [#]	Selected-items set [#]	time [ms]	# of comparisons	# of fused elements
1	25	25	109	625	5
2	50	50	362	2,500	7
3	75	75	799	5,625	7
4	100	100	1,255	10,000	7
5	125	125	2,056	15,625	8
6	150	150	3,032	22,500	9
7	175	175	4,144	30,625	15
8	200	200	5,421	40,000	18
9	225	225	6,804	50,625	18
10	250	225	7,549	56,250	18
11	275	225	8,470	61,875	18
12	300	225	8,861	67,500	18
13	325	225	9,361	72,125	18
14	350	225	10,050	78,750	18
15	375	225	10,999	84,375	20
16	400	225	13,179	90,000	20

Table 6.2: *Results of the experiments about performances of data fusion.*

are selected and compared with all the items of the other services.
In Figure 6.3 we report the chart about the performances of the experiment on data fusion whose details are reported in Table 6.2. It is evident that the relationship among time and number of comparisons is linear. Finally, to adapt the experiment to a scenario in which our "on-demand" policy is applied, the reader should consider as execution time the one corresponding to the number of items in the comparison set. For example, if we consider a scenario in which there are 3 services with 300 retrieved elements each, and a user selects an item from one of those services, the performance of the data fusion on-demand is the time corresponding in the chart to $300+300 =$

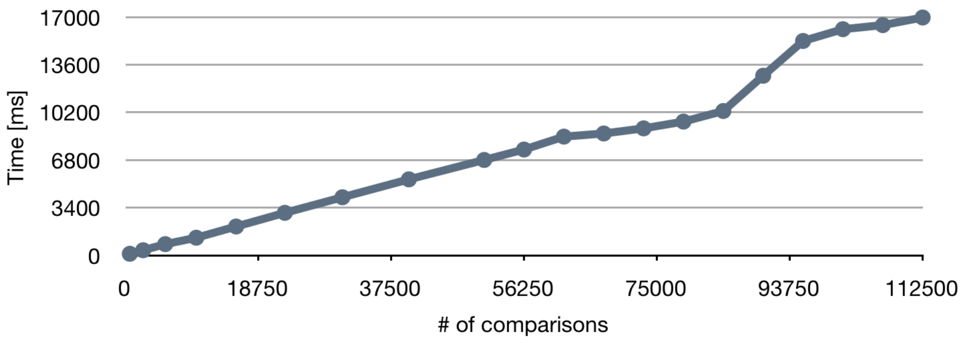


Figure 6.3: *Time of execution over number of comparisons in data fusion performance experiment.*

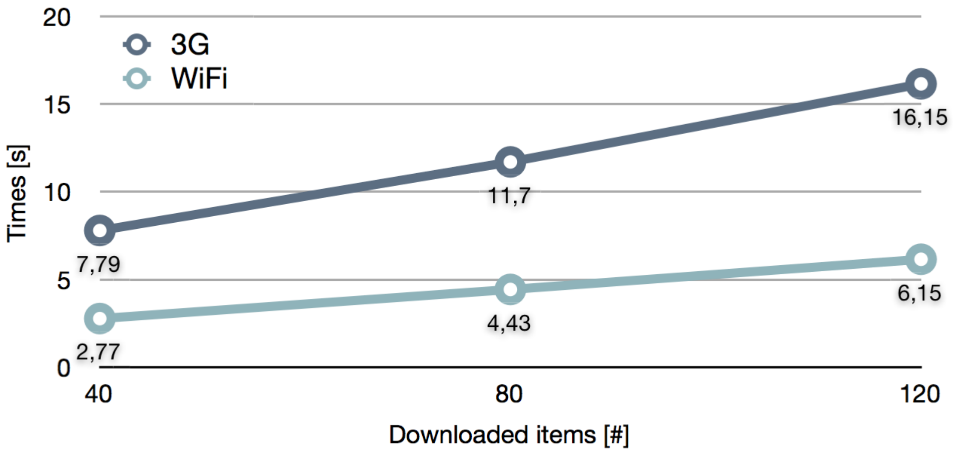


Figure 6.4: *Download and display times on mobile with 3G and WiFi connections.*

600 compared items; that is less than $100ms$ (see the first row of Table 6.2), which is a very good result.

6.1.3 Performances on mobile devices

Our approach allows also to create components that can be exported on different devices. Since the worst possible case is to execute the mashup on a mobile device, we conducted experiments also on small devices to test, in particular, the behavior of the download of data and of our data fusion algorithm.
To conduct the performance tests on mobile devices we used three services, i.e., *Last.FM*, *Upcoming* and *Do Staff Media*, considering two scenarios: the first used Last.FM and Upcoming, and the second all the three services. It is important to monitor download times because mobile devices often use mobile networks like 3G, with lower performances than WiFi or Ethernet connections that instead characterize desktop devices. In Figure 6.4 the performances of a 3G network are compared to the performances of a WiFi network in downloading data from the services. The values reported refer to the whole time, from the execution of requests to the services to the complete display of data on the device. However, since we can assume as constant the time to display data, the values on the chart can be considered as a measure of the download time. Note that, as expected, the download and display times are linear and WiFi has significant better performances than 3G.
The other critical aspect on mobile devices is the data fusion on demand.

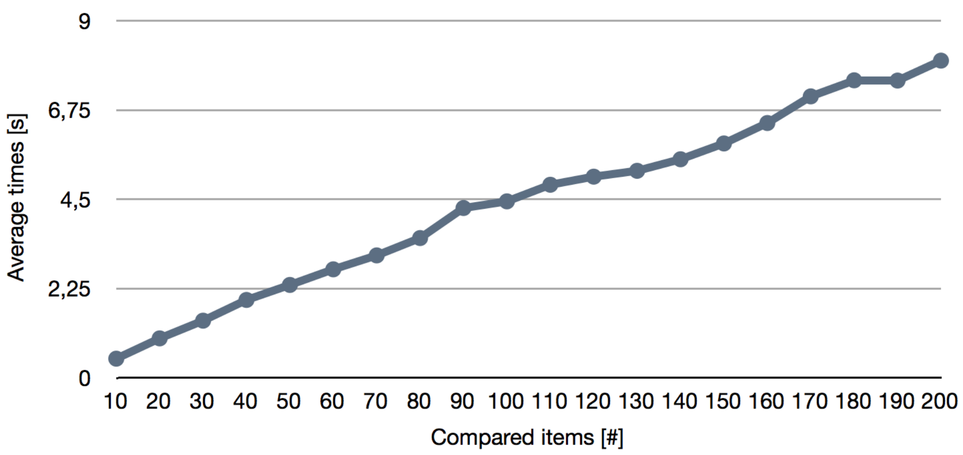


Figure 6.5: *Time of execution over number of comparisons in data fusion performance experiment on mobile devices.*

Figure 6.5 shows the average times of execution of data fusion with respect to the number of compared items. Times are measured from the tap on a selected element in the global view, to the display to the user of the merged details. For each test we chose data items with strings of different lengths in order to consider cases with different complexity[4]. We can observe that the represented relationship is a linear function and the maximum time, corresponding to 200 items, is less than 10 seconds, which, according to [80] is the maximum time a mobile user is disposed to wait.

6.2 User studies

In this section we describe the experiments we conducted in order to validate with users different aspects of our composition process. The first two studies, about the *mashup composition paradigm* and the *component creation*, are controlled experiments where we clustered the selected users in two groups: experts and non-experts, in order to explore whether technology expertise influenced users performance and satisfaction. Both of them adopt the same methodology described in Section 6.2.1, although they focus on different activities that the users can perform through our framework. The third is a preliminary study that aimed at validating the adopted *collaboration mechanisms* and understanding how to improve our first assumptions about opening mashups to collaboration. The fourth is a

[4]The most consuming operation is indeed the comparison among text data items in order to check their similarity.

preliminary *field study on a cultural heritage domain specification* of our approach that aims at validating multi-device execution environments and getting insights to improve the interaction paradigms.

6.2.1 Overall methodology

We adopted a common methodology for the validation studies we conducted with users, so as to be able to compare the results of the first three validation sessions. In the following the phases of the adopted methodology are described:

- *Pre-experiment questionnaire.* This questionnaire had the goal of collecting personal data about users like age and sex and data on their relationship with the technology, in particular the mean number of hours spent on the computer, how much they were used to surf the Internet and their familiarity with service composition and mashups. This questionnaire allow us to classify users and understand whether they were technology experts or not.

- *Tutorial and background.* Before asking users to perform the tasks we briefly explained the tool and its capabilities and functionalities in order to give them a background.

- *Experimental tasks.* In this phase users were asked to accomplish some predefined tasks with incremental difficulty in order to understand the degree of complexity they were able to use and master: if a user could execute the second task, s/he was also able to execute the first. During the task execution we measured the number of steps they were able to finish and the time they spent to complete the tasks.

- *Post-experiment questionnaire.* After the task execution the users were asked to fill a post-experiment questionnaire with the goal of measure two main dimensions: the *ease of use* and the *satisfaction.* In this questionnaire there were: 7-point scale questions (from a very positive to very negative) and open questions where users could express their opinion on the tool and suggest improvements. The scale of the close questions were inverted randomly in order to force users to keep attention and to improve the significancy of the answers.

Table 6.3 presents data about the user samples selected to conduct the studies. In total we selected 82 users with an age between 18 and 70 with different expertise. In particular 35 experts and 47 non experts (respectively the 43% and the 57%). Thus we were able to compare the results of experts

Experiment	# of users	Ages	Expert/Not-expert users
Mashup composition paradigm	35	21-53	13/22
Component creation	36	18-70	17/19
Collaboration mechanisms	11	21-64	5/6
Total	**82**	**18-70**	**35/47**

Table 6.3: *Demographic data about the experiments user samples.*

and not experts in order to study the technology biases on the usage of our solution.

6.2.2 Study on the mashup composition paradigm

A first study was conducted to evaluate the UI synchronization paradigm. Users were asked to add, remove, couple and synchronize components as described in Chapter 5. For this user study we considered a customization of our platform as a dashboard for a specific domain use case [23], i.e., the sentiment analysis of touristic cities in Europe. This use case was provided by the Milan Municipality that was interested in monitoring the city reputation about tourism in Milan with respect to the main cities in Europe. We analyzed the sentiment of data retrieved from social networks, e.g., *Twitter*, *Trip Advisor* and *Lonely Planet*, and provide analytics visualizations to users. Users can select sources and visualizations, and filter data according to some predefined categories. The application of a filter is thus implemented as a new query to the sentiment analysis data warehouse. This is an example of domain-specific adaptation of the platform because, in the domain in which it is adopted, analysts were more confident to sources, filters and viewers. With our platform, we enable the tourism officers to create their dashboard by composing and synchronizing information visualization-based UI Components.

The experiment was composed by two tasks, the first was about adding, create couplings and verify the synchronization of components by raising events in a simpler scenario while the second was more complex because was more articulated and gave less information about the composition procedure in order to leave users free to choose how to proceed.

The two tasks were comparable in terms of number of components to be integrated and composition steps. Task 2, however, required a less trivial definition of filters, to sift the involved data sources, and a more articulated definition of couplings. Also, while the formulation of Task 1 was more procedural (i.e., it explicitly illustrated the required steps), Task 2 just described the final results to be achieved, without revealing any details about the procedure required.

- **Task 1**. In the first task the user had to add two chart components to visualize sentiment measures and then had to create couplings between the components. After that, the user was invited to active the coupling by interacting with the UI in order to enjoy the experience of the intra-components synchronization. The last point in the task asked the user to add another chart and synchronize it with the charts included before.
- **Task 2**. The second task was articulated along only two point. Our intent was indeed to increase the difficulty with respect to Task 1 giving less information about the composition procedure, and leaving the user free to chose how to proceed. The task asked first to add chart components and then to couple them in order to obtain a dashboard that showed some particular dimensions with respect a domain specific analysis on the reputation of the tourism in Milan.

All the participants, both experts and not experts, were able to complete both the tasks without particular difficulties. No differences in task completion time were found between experts and novices. In particular, technology expertise was not discriminant for task 1 ($p = .161$) and for task 2 ($p = .156$). The lack of significant differences between the two groups does not necessarily mean that expert users performed bad. However, it indicates that the tool enables even inexperienced users to complete a task in a limited time. The average time to complete task 1 was about 2.5 minutes, while for task 2 it was about 2 minutes. This positive result is not surprising since novices can perform as good as experts even in the case of Web searches [56, 91].
The difference in completion times for the two tasks can be also used as a measure of learning [46]. This difference is about half a minute ($t = 28.2, p = .017$), i.e., a reduction of about 15%. This result highlights the learnability of the tool: although the second task was more critical compared to the first one, subjects were able to accomplish it in a shorter time.
The *ease of use* was further confirmed by the data collected through four questions in the post-questionnaire, asking users to judge whether they found it easy to identify and include services in the composition, to define couplings between services, and to monitor and modify the status of the mashups. We also asked users to score the general ease of use of the tool. Users could modulate their evaluation on a 7-point scale. The reliability of the ease of use questions is satisfying ($\alpha = .75$). The correlation

between the four detailed questions and the global score is also satisfying ($\rho = .58, p < .001$). This highlights the high external reliability of the measures. On average, users gave the ease of use a mark of 1.77 (the scale was from 1 very positive to 7 very negative). The distribution ranged from 1 to 4 ($mean = 1.77, MSE = .12$). We did not found differences between novice and expert users. This was especially true for perceived usefulness ($p = .51$).

The *user satisfaction* with the composition paradigm was assessed using two complementary techniques. A semantic-differential scale required users to judge the method on 12 items. Users could modulate their evaluation on a 7-point scale (1 very positive - 7 very negative). Moreover, a question asked users to globally score the method on a 10-point scale (1 very positive - 10 very negative). The reliability of the satisfaction scale is satisfying ($\alpha = 0.76$). Therefore, a user-satisfaction index was computed as the mean value of the score across all the 12 items.

The average satisfaction value is very good ($min = 1.3, max = 3.5, mean = 2.2, MSE = .09$). The correlation between the average satisfaction value and the global satisfaction score is satisfying ($\rho = .41, p < .015$). On average, users gave the composition method a mark of 2.9, with a distribution ranging from 2 to 4. We did not find differences between experts and novices. The moderate correlation between the satisfaction index and the ease of use index ($\rho = .55, p = .011$) also reveals that who perceived the method as easy also tended to evaluate it as more satisfying. This confirms that ease of use is perceived.

The last two questions asked users to judge their performance as mashup developers and to indicate the percentage of requirements they believed to have satisfied with their composition. This metric can be considered as a proxy of confidence [46]. On average, users indicated to be able to cover the 91% of requirements specified by the two experimental tasks ($min = 60\%, max = 100\%, MSE = 1.7\%$). They also felt very satisfied about their performance as composers ($mean = 1.8, MSE = .13$; 1 very positive - 4 very negative). We also found a direct correlation between the users perception of their performance as mashup developers and the global ease of use ($\rho = .57, p < .001$), meaning that the tool's ease of use improves user confidence.

6.2.3 Study on the component creation

This second study aimed at evaluating the activity of component creation through our tools. Users were asked to create components starting from the selection of preregistered services in two tasks with incremental complexity as in the previous test.

- **Task 1**. Users were asked to execute a task for their free time organization, by creating a map-based component. First they were asked to select *Eventful* and *Upcoming* services, which are services that provide information about events. From the data retrieved by the services, users had to select latitude and longitude coordinates to locate the event on the map (once added an event on the map it appears as a colored marker) and add details to the markers by adding the description and event address. At the end the users were asked to save the created component.
- **Task 2**. The second task was about the planning of a trip to Milan. Users had to select the map visual template and associate with it, from the *GooglePalces* service, the hotels and the underground stations. As details for the added items, they were required to associate, the name of the hotel or underground station.

All the participants were able to complete both the tasks without relevant difficulties. The average times for the completion of the two scenarios were respectively $3m30s$ for novices and $3m36s$ for technology experts for task 1 and $5m20s$ for novices and $4m46s$ for experts for task 2. Considering these data, we can notice that there is not a relevant difference between novices and experts. This hypothesis is confirmed also by the Wilcoxon-Mann-Whitney test ($U = 47, p = .4707, 1 - \alpha = 95\%$) because the two populations have the same distribution.
From the post-experiment questionnaire we monitored the tool's *easy of use* asking to users to express their opinion with respect to the selection of services and the creation of the components through the visual template mapping techniques. Users could modulate their evaluation on a 7-point scale. The reliability of the ease of use questions is satisfying ($\alpha = .73$). On average, users gave the ease of use a mark of 2.56 (the scale was from 1 very positive to 7 very negative). The distribution ranged from 1 to 4 ($mean = 2.56, MSE = 1.53$). Comparing the distribution of the two categories of users considered in these tests we can conclude that there is not a difference distribution for novices and experts ($t = 0.97, p = 0.45, \alpha = 5\%$).
The *user satisfaction* was assessed using, as in the previous study, two complementary techniques: a semantic-differential scale on 12 items (1 very positive - 7 very negative) and a question to globally score the method (1 very positive - 10 very negative). The reliability of the satisfaction scale is

satisfying ($\alpha = 0.75$).
The average satisfaction value is good ($min = 2.1, max = 3.6, mean = 2.7, MSE = 1.34$). On average, users gave the component creation method a mark of 2.7, with a distribution ranging from 1 to 4. Also in this study, we did not find differences between experts and novices because they have the same distribution ($t = 0.26, p = 0.80, \alpha = 5\%$). The moderate correlation between the satisfaction index and the ease of use index ($\rho = .55, p = .011$) also reveals that who perceived the method as easy also tended to evaluate it as more satisfying. This confirms that ease of use is perceived.
Finally, in the last two questions, users were asked to judge their performance as component developers and to indicate the percentage of requirements they believed to have satisfied with their composition. On average, users indicated to be able to cover the 89.6% of requirements specified by the two experimental tasks ($min = 50\%, max = 100\%, MSE = 1.65\%$). They also felt very satisfied about their performance as composers ($mean = 1.92, MSE = .39$; 1 - very positive, 4 - very negative).

6.2.4 Preliminary study on collaboration mechanisms

The evaluation of the collaboration mechanisms is a preliminary study that aims at evaluating the chosen mechanisms to understand how to improve our first assumptions about opening mashups to collaboration.
For this experiment, users were asked to execute three incremental tasks: from a base task, in which they had to complete simple mashup tasks and introduce some collaboration mechanisms, to an intermediate task with simple sharing activities, up to an advanced task where users are asked to personalize the sharing polices and use advanced collaboration mechanisms.

- **Task 1**. In the first task users had to compose a mashup including the *YouTube*, *LastFM* and *Flickr* UI Components and to use them without any coupling. Then another user, one of the test conductors, logged into the platform. Thus the user who was performing the experiment could use the presence awareness mechanisms to identify how many users were online and who they are. The two users started to interact via the instant messaging tool and at the end of the task the user was asked to locate the latest activities menu and to save the composition.

- **Task 2**. In the intermediate task the user did not received much guidances. S/Hewas was asked to create a composition with at least three components, to save the composition and to share such a composition with another user.

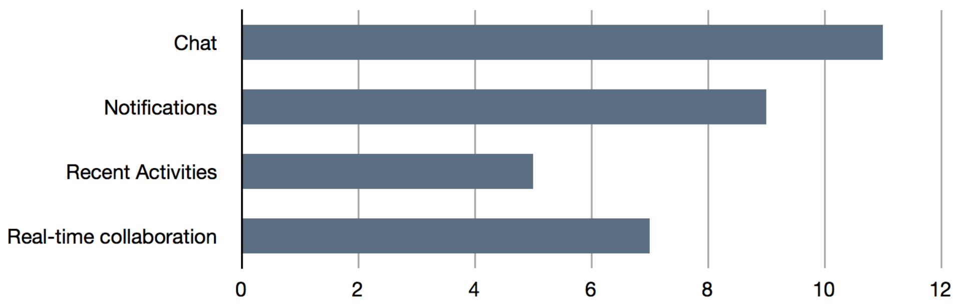


Figure 6.6: *Easy of use of the main collaboration mechanisms.*

- **Task 3**. In the last task users had to open a shared composition and try to modify it. The user did not have the permission to modify the composition and asked to the composition owner to give her/him such permissions. Than s/he created a new VT Component and added it to the composition. As last activity, the user was asked to change the permissions using the advanced settings of the sharing environment.

After the execution of the scenarios, we asked users to evaluate the sharing environment and the collaboration mechanisms. About the sharing environment, users expressed their opinion in a 5-values scale (1 very negative - 5 very positive). On average users gave to the sharing environment a mark between 2 and 5 ($mean = 3.75, MSE = 1.84$), thus users were globally satisfied by this environment. Figure 6.6 shows the results to the question about the easy of use and the immediacy of understanding of the main collaboration mechanisms. Real-time collaboration includes all the sub-mechanisms that allows this functionality, e.g., users awareness and real-time synchronization of the users' workspaces. As expected, chat and notifications are the best rated mechanisms because users are already used to use them in other collaboration (or communication) platforms while it is encouraging that real-time collaboration mechanisms were considered easy for the majority of users.

6.2.5 Preliminary study on domain-specific customization

We also conducted a field study to assess the validity of a domain-specific customization of our framework for the cultural heritage domain. We developed execution engines for two different types of devices: a multi-touch display (46") and Android tablets (7"). These are indeed the devices adopted at archaeological park of Egnathia, where we conducted the studies, as a support for the park guides [5]. In particular, the guides needed a platform to extract contents from heterogeneous (personal or third-parties) sources, and compose Personal Information Spaces (PISs) that can be ubiquitously executed on different devices and contexts of use. In this study, according to the user-centered design, we were interested in collecting feedback in a real context of use. We were interested in validating with end users our approach by evaluating pros and cons, and getting insights that can improve the current state of the prototypes of the multi-device execution environments. It is worth noticing that at this stage of our work, the field study is part of a set of formative evaluations that we are still conducting, and is mainly focused on qualitative data.

6.2.5.1 Participants and Design

The study involved 28 visitors and 2 professional tourist guides, named Achille (male) and Conny (female). Each guide accompanied a group of 14 visitors in the visit of the Egnathia park. Visitors were people who had booked a visit to the park. They were heterogeneous as regards age (from 21 to 50 year-old, plus an 8-year old child), gender, and cultural background. These visitors were randomly divided in two groups.

6.2.5.2 Procedure

The study consisted of two main sessions: *(i)* PIS composition for organizing the visit, and *(ii)* park visit. In the first session the two guides were given a 1-hour demonstration of the desktop application, accessible through a PC, to be used to create the components. After this, according to the co-discovery exploration technique [17], the two guides were invited to create together a component for visiting an archaeological park, named Monte Sannace, in Apulia region. In this way, the guides had the possibility to get familiar with the application. Then, they were asked to create the PIS to be used for the visit of the Egnathia archaeological park. The guides individually created their PIS by positioning on an interactive map of the park all the multimedia contents they would like to illustrate to visitors. At the end of the PIS composition session, the guides were interviewed together to gather impressions, problems and suggestions in order to improve the implemented composition mechanisms and the overall system. In the second session the park visit session was performed at the archaeological park of Egnathia. This session was composed of two different phases: 1) the briefing phase at the beginning of the visit, in which the guide accessed his/her PIS through a large multi-touch display (46 inches), placed at the entrance of the indoor park museum, 2) the tour phase, in which the guides accessed

their PIS on a tablet (7 inches) during the tour through the remains in the park. In the briefing phase, the guide interacted with his/her PIS on the multi-touch display to introduce visitors to the history of Egnathia and the remains that they were going to see in the park. After this, the tour phase began. During the tour, guides could use the tablet to show visitors the contents of their PIS regarding the remains. In both phases, guides could search for new contents and, possibly, update their PIS. The visit session lasted on average 1 hour and half. At the end of the visit, each visitors' group participated in a focus group where their impressions on the overall visit experience were collected. The two guides were interviewed together again at the end of the visit, to discuss in more details their experience, highlighting pros and cons of PIS use.
In order to analyze the guides' experience in composing their PIS and using it (the first objective of our study), we gathered data through naturalistic observation of the guides while: 1) composing their PIS on the desktop, 2) interacting with the created PIS on the multi-touch display during the briefing phase, 3) interacting with the tablet during the tour phase. These data were complemented by the guides' comments gathered during the interviews after the PIS composition and at the end of the visit of the Egnathia park. To get information on the visitors' experience we collected data through naturalistic observation of visitors during the visit and a focus group at the end of the visit.

6.2.5.3 Results

Results are presented in three different parts, depending on the phase they are referring to:

- **Composition phase**. In this phase, the guides were observed while composing their PIS for visiting the archaeological park of Egnathia using the desktop application. In general, the usability problems they experienced were not so serious to stuck them; they were indeed able to continue the PIS composition without the help of the two HCI experts. Both guides appeared disoriented by the few contents returned by some of the searches they had performed; they tried to refine the search by typing different keywords and, finally, added the most appropriate multimedia material available on the Web. At the end of this phase the two guides were interviewed together. As overall impression, they said they appreciated the ease of use of the application, in particular the possibility to quickly put the retrieved content on the park map. They were rather satisfied by the PIS they had created and they were confident that it would be a valuable support during the visit.

- **Briefing phase**. The briefing aimed at both introducing visitors to the history of Egnathia and providing some preliminary information. The briefing time, during which the guides used the multi-touch display, was different: 28 minutes and 35 seconds for Achille, and 11 minutes and 45 seconds for Conny. During his interaction with the multi-touch display, the guides experienced some interaction difficulties due to some limits related to the multi-touch display technology. Both guides appeared quite relaxed in using the multi-touch display. They illustrated the multimedia contents they had previously inserted in their PISs. They were able to search new content without difficulties related to the search functionality. During the briefing phase, all visitors appeared very interested in the multimedia contents illustrated by the guides on the multi-touch display: they asked questions to their guide, commented images among them, and in general appeared engaged and stimulated by the shown material. This was confirmed in the focus group, in which visitors explicitly expressed their positive opinion about the briefing phase. It is worth noticing that, when the search for new content required more than 2 minutes, visitors appeared to be disturbed and started chatting among them and looking around.

- **Tour phase**. In the tour phase, the two guides accompanied the visitor group through the remains in the outdoor park. The guides were free to use their tablet as well as the panels located in the park to better present the park remains. Achille and Conny used tablet and panels in different ways. Conny was more prone to the use of such tools; in fact she used the tablet 9 times and the panels 9 times. In total, she spent 7 minutes and 53 seconds to interact with the tablet, and 3 minutes and 58 seconds to show images on the panels. Achille used such tools very little: he used a panel once for 10 seconds and the tablet once for 1 minute. Both guides performed searches through the PIS on the tablet. Only one search did not get results of her interest. The only time Achille used the tablet was to perform a search. Both guides remarked that, during the tour, searches requiring more than 2 minutes interrupted the narrative and distracted visitors. Moreover, they said that, even if they did not feel uncomfortable during this waiting time, they would have preferred to get more material without the delay due to the Internet connection, i.e., from local repositories.

6.2.5.4 Discussion

The objective of this study was to assess the value of the PIS, accessible from different devices, in the specific context of the visits at archaeological parks. To this aim, we analysed the experience of the two actors involved in a visit: the guide and the visitor. The guide has a double role: 1) composer, i.e. s/he creates her/his PIS; 2) end user, i.e. s/he uses the PIS during the visit to illustrate the park remains to visitors. The visitors participate in an visit which is enhanced by the availability of different types of multimedia materials and, thanks to the possibility given by the PIS to search new content, their curiosity may be better satisfied than in traditional visit.

Composing the PIS with a desktop application did not create any problem to the guides. They appreciated the support of the PIS in organizing the material for the visit. However, the guides complained about the scarce material they were able to find when searching the services available in the platform. This is a problem common to all service-based applications, which have to rely either on contents made available by third-parties or on user-generated contents. To limit this problem, more sensible services should be added into the platform; they can be further third-parties or local and ad-hoc created collections of contents, maintained by domain experts and even fed by the end users themselves by adding self-produced material. Both guides and visitors appreciated very much the briefing phase with the support of the multi-touch display, even if it lasted much longer than in traditional visits, where the briefing to introduce visitors to the archaeological park is about 5 minutes at most.

Nobody complained about this longer phase; on the contrary they all said: “It’s worth it!". The multi-touch display is very valuable in this phase of the visit, since it allows the guides to present much more multimedia materials related to park elements, which enrich a lot their spoken presentation.

Summarizing, the study results showed a general appreciation of use of the multi-touch display in the context of the visit. However, a difficulty was generated by the position of the multi-touch display. It was positioned on a support 110 centimeters high. For this reason, some visitors could not see the whole display. In future installations, it would be better to use a higher support (at least 150 centimeters), placing it on a platform of at least 50 centimeters on which the guide will get on. In this way, the display will be better visible by all visitors. However, the fact that the visitors moved to see the display is a symptom of their interest in looking at the material showed by the guide’s PIS.

A negative aspect of the use of the PIS on the multi-touch display was the waiting time of a search for new content. This was in part due to the time for typing the search keywords and in part to the low connection speed. However, the search through composition tool is limited to the services made available in the platform (i.e. Flickr, YouTube and Wikipedia). Thus, the search for very specific material can often be unsuccessful, and this might easily bother guides and visitors.

6.3 Composition assistance mechanisms

In order to test the effectiveness of mining the most frequent associations from repository of mashups and mashup components, we performed a set of experiments in which we simulated the creation of a number of mashups, based on our assisted composition method. We selected a set of well-designed and popular mashups, top-ranked in the *programmalbleWeb* repository and we used such mashups as benchmarks. More specifically, we used such mashups for the identification of our *goal mashups*, i.e., a set of interesting mashups with respect to their popularity and quality. We wanted indeed to understand in which measure, given a properly selected set of components including the ones in the selected goal mashups, our mechanism would have led to the composition of quality mashups.
In order to identify the set of benchmark mashups, we considered the popularity ranking provided by programmableWeb and we mediated it with the judgment of five evaluators, achieved in a previous study [16] through heuristic evaluation sessions that did not take into account our quality metrics. For each goal mashup, we simulated our assisted composition assuming as starting point the inclusion of one of the components in the considered goal mashup. Our recommendation technique was able to suggests different alternative compositions, that in 80% of the cases were very close to our goal mashups, with a distance of maximum 2 components. In any final composition, the quality was not degraded with respect to the goal mashups.
We here describe one of the performed composition experiments, in which the goal was to create a mashup providing a visual encyclopedia: the idea is to enrich the textual content extracted by an online encyclopedia or dictionary with visual content. The benchmark mashup selected to verify the resulting composition is Shahi[5], a popular mashup which is also one of the top-ranked mashups among those that we have analyzed. In particular, Shahi is a visual dictionary that combines Wiktionary content with Flickr

[5]http://blachan.com/shahi/

Iteration no.	Present categories	Selected Association Rule	Suggested Component
1	Reference	Reference $\rightarrow$ Photo	Flickr
2	Reference, Photo	Reference, Photo $\rightarrow$ Social	Twitter
3	Reference, Photo, Social	Reference, Social $\rightarrow$ Search	Google Ajax Search

Table 6.4: *Selection of association rules in the construction for the Shahi sample mashup.*

images, and offers search facilities by using Google Ajax Search and Yahoo Image Search.
The assisted composition procedure mainly relies on the component registry, and on the association rule extracted from the analysis of mashup repositories. The component registry contains all the available components together with their descriptors in which operational and quality features are described. The association rules repository instead suggests valuable patterns for combining different components' categories. Once the user selects the first component to be included in the composition, the recommendation procedure starts, and proceeds according to two fundamental steps: i) the selection of the category of the component to adopt in order to enrich the current composition and ii) the selection of a specific component, within the selected category, to add. In the second step, the selection of the most suitable component is driven by the compatibility with the components already in place and the potential mashup quality. The two steps are repeated until the user reaches the desired goal, or the algorithm does not find any other components to include in the composition.
For example, in order to build the visual encyclopedia mashup, we consider Wikipedia as the first component; applying iteratively the procedure for the assisted composition we obtain the results summarized in Table 6.4. Here, the second column refers to the categories to which the components already involved in the mashup belong. The third column specifies the association rule that drives the selection of the category of the new component that can extend the composition. When more rules are identified, support and confidence are considered to identify a single rule. In case of more rules with the same value for such parameters, the different possible expansions of the mashups are ranked based on the quality and added value of the components falling in the involved categories. Finally, the fourth column contains the name of the component selected because it results to be the top-ranked, based on the assessment of quality measures.
After the third steps, the recommendation procedure did not find any other association rules to apply for expanding the current composition. Comparing the obtained mashup with our benchmark, Shahi, we noticed that both mashups address the same situational need, but our quality-aware assisted

composition also suggested to extend the composition with a “social” component (i.e., the Twitter API), not included in the original mashup. This in a sense proves that the aggregation of different quality dimensions lead to the consideration of different composition solutions that can improve the value of the final composition.

CHAPTER 7

Conclusions and future work

In this thesis we introduced our perspective over the EUD-based composition of mashups, discussing how adequate abstractions, mainly based on visual template completion, can enable a lightweight integration process, leading to the definition of unified data views, the synchronization between such data and remote APIs and custom device services, and the integration of UI components of the presentation layer.
With respect to the goals illustrated in Chapter 1, in this thesis we presented the following results:

- The definition of a *user-driven development process* that, according to the need of non-expert users to compose situational applications, hides the back-end complexity and simplify the task of information integration. Our assumption is that, in order to provide support for the EUD of mashups, we have to put the user in the center of the development process. Hence, we adopted WYSIWYG (What You See Is What You Get) visual mechanisms that immediately show to the users an example of the artifact they are creating [19, 23]. The closeness of the design artifacts with the "appearance" of the final application confers to our composition paradigm the possibility to help end users in understanding how to meet their needs through self-developed applications. This thesis describes in detail the activities of the proposed development process.

- The definition of UI-centric models for resource integration. A Visual Integration model [20] allows the user to associate data and function to predefined UI Templates, providing a support for resource integration. We also defined a *Domain Specific Language* (DSL) able to encapsulate, abstracting from UI styles and execution environments, the fundamental constructs of the Visual Integration paradigm. Through our models, we describe properties of elements of general validity in the mashup domain, like UI Components, events, operations, couplings, that can be easily adopted for lightweight integration of resources also in other platforms or contexts.

- The pervasive diffusion of technology in form of mobile devices and mobile applications, more and more service-oriented and integration-intensive, legitimates our approach on *multi-device deployment of composite resources*. This is possible thanks to the capability of our tools to generate descriptors based on our DSL. Users integrate "by example" the resources, obtaining artifact descriptors that can be then executed on different devices by client-side execution engines. We therefore developed different execution engines for desktop devices, mobile devices (both smartphones and tablets) and multi-touch large displays [4, 5, 20, 21].

- With respect to the *lightweight integration of resources*, our Visual Integration model supports at the client-side the combination of integrated views of data coming from multiple services [20, 21] and publish-subscribe, event-driven synchronization of data sources [19, 23]. In the first case, we adopt an "on-demand" data fusion [20], so that the efforts required for comparing and merging data are concentrated only on the data the user is really interested in. We introduce a lightweight integration paradigm that does not require any dedicated integration platform but is based mainly on integration mechanisms at the UI level that can be defined and executed on different client devices, where the client can also be a mobile device with limited capabilities.

- *Collaboration-oriented mechanisms* extend the basic composition paradigm to cover both synchronous and asynchronous collaboration, thus providing a comprehensive approach for users to exchange ideas and dynamically co-create integrated information spaces [5, 69]. As users become increasingly familiar with Web 2.0 mechanisms to exchange ideas and instantly communicate with peers, collaboration becomes a fundamental feature of modern Web-based applications. However, service-based Web composition environments offering this feature are still lacking. This thesis tries to fill this gap, by showing how services, service-based resources and composition models can be considered objects of collaboration. The presented collaborative features emerged from a series of studies where real users expressed their desiderata on possible collaborative aspects of the platform [5].

- As a *proof of concepts* we implemented a platform for end-user development of mashups [69] and mobile execution engines [21] running on mobile and multi-touch large displays [5]. We conducted performance tests on the integration of resources, and in particular on the paradigm for "data fusion on demand", under the limited computation capability of mobile devices. The results of the performance tests show that our approach is feasible also on limited computation environments. We therefore conducted three user studies (the first one is published in [23]) in order to validate the adequateness of the composition paradigm with respect to the abilities of the average end users. We thus proved that the lack of technical skills does not affect user performance in terms of effectiveness, efficiency and satisfaction. A field study conducted at an archaeological park [5] allowed us to analyze archaeological park guides' experience in composing and using Personal Information Spaces (PISs) through our platform, and the impact on the visitors' experience. The study showed that the guides appreciated the usefulness of PISs, in particular the good mapping with their working activities. On their side, visitors enjoyed the possibility of looking at pictures and videos that enhanced the guides' spoken presentation. Finally, we tested our quality-based recommendation of components [22, 82] through an experiment that returned encouraging results.

7.1 Impact of the developed framework

One of the strength of our work is that it is a general framework that can be adapted to different specific domains. In the context of a project founded by the Milan Municipality on sentiment analysis in the touristic domain, we adapted our framework to run as a dashboard to visualize and analyze sentiment analysis data [23]. Sentiment analysis focuses on understand-

ing market trends starting from the unsolicited feedback provided by users comments published on the Web. Components made requests to a local source containing the analyzed data. Users were enabled to add and synchronize components, and filter data on different dimensions. Raw data were aggregated according to the request parameters and provided to users with different information visualization techniques. This tool was installed at the Milan Municipality and was used by the real end users.
To respond to the needs of a community of an archaeological park's guides, we also implemented execution environments for different devices, and some park guides used both the composition environment and the execution environments for a real visit [4, 5]. As described in Section 6.2.5, we implemented an execution environment on a multi-touch large display, which was used by the guides in the visit briefing phase to show some related contents. Then, during the tour phase, guides used the other execution environment deployed on a tablet. The considered field study allowed us to validate our approach and collect very useful feedback from guides and visitors.
In 2012, in the context of a weekly project conducted during a summer school on ubiquitous computing and information visualization in Oulu, Finland, we used our platform to display traffic data by applying information visualization techniques to enhance the city traffic planning [83]. We were able to use our Mashup Dashboard, integrating data coming from city sensors and providing interactive visualizations.

7.2 Future work

We are planning to collect additional user feedback in order to improve further the interaction paradigms and to elicit new features that are not provided so far. However, we already identified some additional features that can contribute to complete our approach and improve its usability. We are currently working on refining our meta-design approach and revising the implementation of our platform accordingly. In particular, we are designing an environment to support the WYSIWYG creation of UI Templates (which currently have to be manually coded in HTML and JavaScript). We are also refining the techniques used to link the abstract representation of visual templates, as a set of generic visual renderers, to the concrete visualizations in the final applications. This can be achieved through the definition of an intermediate modeling layer highlighting which concrete visual renderers are exploited within the concrete visualizations. This mapping is now hard coded within the execution environment. Moving such a

specification to the model level enhances the decoupling between content, presentation and device our approach aims to.
The easy configuration of services and service integration presented in this thesis also assumes a set of default rules, for example related to the operations that can be invoked trough UI Components. We however believe that with few extensions, but keeping the same visual metaphor, our current platform can evolve towards a development environment for expert users, giving them the possibility to configure services and UI Components more flexibly, and also introducing more sophisticated conflict resolution policies in the construction of the integrated data views. Our future work is devoted to improve the design environment along this direction, trying to achieve a full-fledged approach, accommodating different expertise levels.
The implemented prototypes are sufficient to demonstrate the validity of our approach in terms of feasibility, easy of use and users' satisfaction but some technological aspects con be improved. In particular, we are looking for better performing technologies to manage communication and live editing collaboration techniques and we are planning to improve the degree of integration of the composition assistance mechanisms in PEUDOM, concerning the quality-aware recommendation of components.
It would be also interesting to investigate how our tools can be included or integrated with other end-user tools, e.g., Content Management Systems (CMS) like Wordpress or Drupal or enterprise portals like Liferay. Also for this reason, it could be useful to adopt some of the emerging standards for web applications like W3C Widgets and Web Components. While Widgets are a W3C Recommendation and platforms for the deployment of Widgets, like Apache Rave, have been developed, Web Components are still in a working draft status. Web Components are a more promising standard for mashups because Widgets do not allow the intra-widget communication, while Web Components do.
Finally, we are now investigating how to customize our paradigm in another specific domain by implementing a real-case scenario provided by the VINCENTE project (A Virtual collective INtelligenCe ENvironment to develop sustainable Technology Entrepreneurship ecosystems) in order to consider a new use case and to improve our approach.

7.3 Achievements

In 2010, in recognition of the innovative aspects promoted by the specialization of our platform for sentiment analysis, the City of Milan has been awarded with a prize for Innovation in Enterprise 2.0.

In 2012, the paper *Quality-aware mashup composition: issues, techniques and tools* was one of the candidate for the best paper award at the QUATIC (Quality of Information and Communications Technology) conference.
In July 2013, at the International Conference on Web Engineering (ICWE 2013) in Aalborg, Denmark, we won the Best Demo and Poster Award for the paper *PEUDOM: a Platform for End-user Development of Common Information Spaces*.

Bibliography

[1] Yahoo! pipes. `http://pipes.yahoo.com/pipes/`.

[2] M. Addisu, D. Avola, P. Bianchi, P. Bottoni, S. Levialdi, and E. Panizzi. Multimedia information extraction: Advances in video, audio, and imagery analysis for search, data mining, surveillance and authoring. chapter 24: Annotating Significant Relations on Multimedia Web Documents, pages 401–417. IEEE Computer Society Press, 2012.

[3] Saeed Aghaee, Marcin Nowak, and Cesare Pautasso. Reusable decision space for mashup tool design. In *EICS*, pages 211–220, 2012.

[4] C. Ardito, M.F. Costabile, G. Desolda, M. Matera, A. Piccinno, and M. Picozzi. Composition of situational interactive spaces by end users: a case for cultural heritage. In Mille et al. [72], page In print.

[5] Carmelo Ardito, Paolo Bottoni, Maria Francesca Costabile, Giuseppe Desolda, Maristella Matera, Antonio Piccinno, and Matteo Picozzi. Enabling end users to create, annotate and share personal information spaces. In *IS-EUD*, 2013.

[6] Carmelo Ardito, Maria Francesca Costabile, Giuseppe Desolda, Rosa Lanzilotti, Maristella Matera, Antonio Piccinno, and Matteo Picozzi. Personal information spaces in the context of visits to archaeological parks. In *Proceedings of the Biannual Conference of the Italian Chapter of SIGCHI*, page 5. ACM, 2013.

[7] Liam J. Bannon and Susanne Bødker. Constructing common information spaces. In *ECSCW*, pages 81–, 1997.

[8] Jens Bleiholder and Felix Naumann. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.

[9] Eric Bonabeau. Decisions 2.0: the power of collective intelligence. *MIT Sloan management review*, 50(2):45–52, 2009.

[10] Paolo Bottoni, Alessandro Cotroneo, Michele Cuomo, Stefano Levialdi, Emanuele Panizzi, Marco Passavanti, and Rosa Trinchese. Facilitating interaction and retrieval for annotated documents. *International Journal of Computational Science and Engineering*, 5(3):197–206, 2010.

[11] Paolo Bottoni, Stefano Levialdi, Nicola Pambuffetti, Emanuele Panizzi, and Rosa Trinchese. Storing and retrieving multimedia web notes. *IJCSE*, 2(5/6):341–358, 2006.

[12] Ulrik Brandes. On variants of Shortest-Path Betweenness Centrality and their Generic Computation. *Social Networks*, 30(2), 2008.

[13] Andreas Brodt and Daniela Nicklas. The telar mobile mashup platform for nokia internet tablets. In Alfons Kemper, Patrick Valduriez, Noureddine Mouaddib, Jens Teubner, Mokrane Bouzeghoub, Volker Markl, Laurent Amsaleg, and Ioana Manolescu, editors, *EDBT*, volume 261 of *ACM International Conference Proceeding Series*, pages 700–704. ACM, 2008.

[14] Stefano Burigat and Luca Chittaro. On the effectiveness of overview+detail visualization on mobile devices. *Personal and Ubiquitous Computing*, 17:371–385, 2013.

[15] Margaret M. Burnett, Curtis R. Cook, and Gregg Rothermel. End-user software engineering. *Commun. ACM*, 47(9):53–58, 2004.

[16] Cinzia Cappiello, Florian Daniel, Agnes Koschmider, Maristella Matera, and Matteo Picozzi. A quality model for mashups. In Sören Auer, Oscar Díaz, and George A. Papadopoulos, editors, *ICWE*, volume 6757 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2011.

[17] Cinzia Cappiello, Florian Daniel, and Maristella Matera. A quality model for mashup components. In Martin Gaedke, Michael Grossniklaus, and Oscar Díaz, editors, *ICWE*, volume 5648 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 2009.

[18] Cinzia Cappiello, Florian Daniel, Maristella Matera, and Cesare Pautasso. Information quality in mashups. *IEEE Internet Computing*, 14(4):14–22, 2010.

[19] Cinzia Cappiello, Florian Daniel, Maristella Matera, Matteo Picozzi, and Michael Weiss. Enabling end user development through mashups: Requirements, abstractions and innovation toolkits. In *IS-EUD*, pages 9–24, 2011.

[20] Cinzia Cappiello, Maristella Matera, and Matteo Picozzi. End-user development of mobile mashups. In Aaron Marcus, editor, *HCI (12)*, volume 8015 of *Lecture Notes in Computer Science*, pages 641–650. Springer, 2013.

[21] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Alessandro Caio, and Mariano Tomas Guevara. Mobimash: end user development for mobile mashups. In Mille et al. [72], pages 473–474.

[22] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Florian Daniel, and Adrian Fernandez. Quality-aware mashup composition: Issues, techniques and tools. In Mille et al. [72], page In print.

[23] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega, Donato Barbagallo, and Chiara Francalanci. Dashmash: A mashup environment for end user development. In *Proc. of Web Engineering - 11th International Conference, ICWE 2011, Paphos, Cyprus, June 20-24, 2011*, volume 6757 of *LNCS*, pages 152–166. Springer, 2011.

[24] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega, Donato Barbagallo, and Chiara Francalanci. Dashmash: a mashup environment for end user development. In *Web Engineering*, pages 152–166. Springer, 2011.

[25] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in information visualization - using vision to think.* Academic Press, 1999.

[26] Nicole Carrier, Tom Deutsch, Chris Gruber, Mark Heid, and Lisa Lucadamo Jarrett. The business case for enterprise mashups. *IBM White Paper*, 2008.

[27] Peter H. Carstensen and Carsten Sørensen. From the social to the systematic mechanisms supporting coordination in design. *Computer Supported Cooperative Work*, 5(4):387–413, 1996.

[28] Fabio Casati. How end-user development will save composition technologies from their continuing failures. In *End-User Development*, pages 4–6. Springer, 2011.

[29] Fabio Casati, Florian Daniel, Antonella De Angeli, Muhammad Imran, Stefano Soi, Chritopher R. Wilkinson, and Maurizio Marchese. Developing mashup tools for end-users: On the importance of the application domain. *IJNGC*, 3(2), 2012.

[30] Stefano Ceri, Maristella Matera, Francesca Rizzo, and Vera Demaldé. Designing data-intensive web applications for content accessibility using web marts. *Commun. ACM*, 50(4):55–61, 2007.

[31] Prach Chaisatien and Takehiro Tokuda. A description-based composition method for mobile and tethered mashup applications. *J. Web Eng.*, 12(1&2):93–130, 2013.

[32] Olexiy Chudnovskyy, Christian Fischer, Martin Gaedke, and Stefan Pietschmann. Inter-widget communication by demonstration in user interface mashups. In *Web Engineering*, pages 502–505. Springer, 2013.

[33] F Daniel, M Matera, and M Weiss. Web mashups: leveraging user innovation. Technical report, Technical report, Politecnico di Milano, 2009.

[34] Florian Daniel, Fabio Casati, Boualem Benatallah, and Ming-Chien Shan. Hosted universal composition: Models, languages and infrastructure in mashart. In Alberto H. F. Laender, Silvana Castano, Umeshwar Dayal, Fabio Casati, and José Palazzo Moreira de Oliveira, editors, *ER*, volume 5829 of *Lecture Notes in Computer Science*, pages 428–443. Springer, 2009.

[35] Florian Daniel, Fabio Casati, Boualem Benatallah, and Ming-Chien Shan. Hosted universal composition: Models, languages and infrastructure in mashart. In *Conceptual Modeling-ER 2009*, pages 428–443. Springer, 2009.

[36] Florian Daniel, Agnes Koschmider, Tobias Nestler, Marcus Roy, and Abdallah Namoun. Toward process mashups: key ingredients and open research challenges. In *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups*, page 9. ACM, 2010.

[37] Florian Daniel, Maristella Matera, and Michael Weiss. Next in mashup development: User-created apps on the web. *IT Professional*, 13(5):22–29, 2011.

[38] Dapper. `http://open.dapper.net`.

[39] Gerhard Fischer. End-user development and meta-design: Foundations for cultures of participation. In *IS-EUD*, pages 3–14, 2009.

[40] Gerhard Fischer. Understanding, fostering, and supporting cultures of participation. *Interactions*, 18(3):42–53, 2011.

[41] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.

[42] Hendrik Gebhardt, Martin Gaedke, Florian Daniel, Stefano Soi, Fabio Casati, Carlos A Iglesias, and Scott Wilson. From mashups to telco mashups: a survey. *Internet Computing, IEEE*, 16(3):70–76, 2012.

[43] Antonietta Grasso and Gregorio Convertino. Collective intelligence in organizations: Tools and studies. *Comput. Supported Coop. Work*, 21(4-5):357–369, October 2012.

[44] Thomas R. G. Green and Marian Petre. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *J. Vis. Lang. Comput.*, 7(2):131–174, 1996.

[45] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Reusable awareness widgets for collaborative web applications - a non-invasive approach. In *ICWE*, pages 1–15, 2012.

[46] Kasper Hornbæk. Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Man-Machine Studies*, 64(2):79–102, 2006.

[47] http://www.w3.org/2008/webapps/wiki/PubStatus. Api specification documents. Technical report, W3C, January 2014.

[48] http://www.w3.org/TR/2013/WD-components-intro 20130606/. Introduction to web components. Technical report, W3C, June 2013.

[49] http://www.w3.org/TR/widgets/. Widget packaging and xml configuration. Technical report, W3C, November 2012.

[50] Thomas P Hughes. The evolution of large technological systems. *The social construction of technological systems: New directions in the sociology and history of technology*, pages 51–82, 1987.

[51] B. Iyer and T.H. Davenport. Reverse engineering google's innovation machine. *Harvard Busines Review*, 86(4):58–69, 2008.

[52] JackBePresto. `http://jackbe.com`.

[53] Till Janner, Robert Siebeck, Christoph Schroth, and Volker Hoyer. Patterns for enterprise mashups in b2b collaborations to foster lightweight composition and end user development. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 976–983. IEEE, 2009.

[54] Anant Jhingran. Enterprise information mashups: Integrating information, simply. In *VLDB*, pages 3–4, 2006.

[55] Robert Johansen. *Groupware: Computer support for business teams*. The Free Press, 1988.

[56] K. Khan and C. Locatis. Searching through cyberspace: The effects of link display and link density on information retrieval from hypertext on the world wide web. *Journal of the American Society for Information Science*, 49(2):176–182, 1998.

[57] Andrew Jensen Ko, Brad A. Myers, and Htet Htet Aung. Six learning barriers in end-user programming systems. In *VL/HCC*, pages 199–206. IEEE Computer Society, 2004.

[58] Reto Krummenacher, Barry Norton, Elena Simperl, and Carlos Pedrinaci. Soa4all: enabling web-scale service economies. In *Semantic Computing, 2009. ICSC'09. IEEE International Conference on*, pages 535–542. IEEE, 2009.

[59] Reto Krummenacher, Barry Norton, Elena Paslaru Bontas Simperl, and Carlos Pedrinaci. Soa4all: Enabling web-scale service economies. In *ICSC*, pages 535–542. IEEE Computer Society, 2009.

[60] Markus Latzina and Joerg Beringer. Transformative user experience: beyond packaged design. *Interactions*, 19(2):30–33, 2012.

[61] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[62] Henry Lieberman, Fabio Paternò, and Volker Wulf. *End user development*, volume 9. Springer, 2006.

[63] Henry Lieberman, Fabio PaternÚ, and Volker Wulf. *End User Development*, volume 9 of *Human-Computer Interaction Series*. Springer, 2004.

[64] Xuanzhe Liu, Gang Huang, and Hong Mei. Towards end user service composition. In *COMPSAC (1)*, pages 676–678. IEEE Computer Society, 2007.

[65] David Lizcano, Fernando Alonso, Javier Soriano, and Genoveva López. Supporting end-user development through a new composition model: An empirical study. *J. UCS*, 18(2):143–176, 2012.

[66] David Lizcano, Javier Soriano, Marcos Reyes, and Juan J Hierro. Ezweb/fast: reporting on a successful mashup-based solution for developing and deploying composite applications in the upcoming web of services. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, pages 15–24. ACM, 2008.

[67] Xiangfeng Luo, Marc Spaniol, Lizhe Wang, Qing Li, Wolfgang Nejdl, and Wu Zhang, editors. *Advances in Web-Based Learning - ICWL 2010 - 9th International Conference, Shanghai, China, December 8-10, 2010. Proceedings*, volume 6483 of *Lecture Notes in Computer Science*. Springer, 2010.

[68] Maristella Matera, Matteo Picozzi, Michele Pini, and Marco Tonazzo. Peudom: a mashup platform for the end user development of common information spaces. In *Web Engineering*, pages 494–497. Springer, 2013.

[69] Maristella Matera, Matteo Picozzi, Michele Pini, and Marco Tonazzo. Peudom: A mashup platform for the end user development of common information spaces. In Florian Daniel, Peter Dolog, and Qing Li, editors, *ICWE*, volume 7977 of *Lecture Notes in Computer Science*, pages 494–497. Springer, 2013.

[70] E. Michael Maximilien. Mobile mashups: Thoughts, directions, and challenges. In *ICSC*, pages 597–600. IEEE Computer Society, 2008.

[71] E. Michael Maximilien, Hernán Wilkinson, Nirmit Desai, and Stefan Tai. A domain-specific language for web apis and services mashups. In *ICSOC*, volume 4749 of *LNCS*, pages 13–26. Springer, 2007.

[72] Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors. *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*. ACM, 2012.

[73] Abdallah Namoun, Tobias Nestler, and Antonella De Angeli. Conceptual and usability issues in the composable web of software services. In Florian Daniel and Federico Michele Facca, editors, *ICWE Workshops*, volume 6385 of *Lecture Notes in Computer Science*, pages 396–407. Springer, 2010.

[74] Abdallah Namoun, Tobias Nestler, and Antonella De Angeli. Service composition for non-programmers: Prospects, problems, and design recommendations. In Antonio Brogi, Cesare Pautasso, and George Angelos Papadopoulos, editors, *ECOWS*, pages 123–130. IEEE Computer Society, 2010.

[75] Abdallah Namoun, Tobias Nestler, and Antonella De Angeli. Conceptual and usability issues in the composable web of software services. In *Current Trends in Web Engineering*, pages 396–407. Springer, 2010.

[76] Felix Naumann. *Quality-Driven Query Answering for Integrated Information Systems*, volume 2261 of *Lecture Notes in Computer Science*. Springer, 2002.

[77] Michael Nebeling, Stefania Leone, and Moira C Norrie. Crowdsourced web engineering and design. In *Web Engineering*, pages 31–45. Springer, 2012.

[78] Michael Nebeling, Stefania Leone, and Moira C. Norrie. Crowdsourced web engineering and design. In Marco Brambilla, Takehiro Tokuda, and Robert Tolksdorf, editors, *ICWE*, volume 7387 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2012.

[79] Tobias Nestler, Marius Feldmann, Gerald Hübsch, André Preußner, and Uwe Jugel. The servface builder-a wysiwyg approach for building service-based applications. In *Web Engineering*, pages 498–501. Springer, 2010.

[80] Sumaru Niida, Satoshi Uemura, and Hajime Nakamura. Mobile services. *Vehicular Technology Magazine, IEEE*, 5(3):61–67, 2010.

[81] Cesare Pautasso and Monica Frisoni. The mashup atelier. In George Feuerlicht and Winfried Lamersdorf, editors, *Service-Oriented Computing ñ ICSOC 2008 Workshops*, volume 5472 of *Lecture Notes in Computer Science*, pages 155–165. Springer Berlin Heidelberg, 2009.

[82] Matteo Picozzi, Marta Rodolfi, Cinzia Cappiello, and Maristella Matera. Quality-based recommendations for mashup composition. In *Proc. of ComposableWeb 2010, in print*, 2010.

[83] Matteo Picozzi, Nervo Verdezoto, Matti Pouke, Jarkko Vatjus-Anttila, and Aaron J. Quigley. Traffic visualization - applying information visualization techniques to enhance traffic planning. In Sabine Coquillart, Carlos Andújar, Robert S. Laramee, Andreas Kerren, and José Braz, editors, *GRAPP/IVAPP*, pages 554–557. SciTePress, 2013.

[84] José Matías Rivero, Sebastian Heil, Julián Grigera, Martin Gaedke, and Gustavo Rossi. Mockapi: an agile approach supporting api-first web application development. In *Web Engineering*, pages 7–21. Springer, 2013.

[85] Agnes Ro, Lily Shu-Yi Xia, Hye-Young Paik, and Chea Hyon Chon. Bill organiser portal: A case study on end-user composition. In *WISE Workshops*, volume 5176 of *LNCS*, pages 152–161. Springer, 2008.

[86] David E. Simmen, Mehmet Altinel, Volker Markl, Sriram Padmanabhan, and Ashutosh Singh. Damia: data mashups for intranet applications. In Jason Tsong-Li Wang, editor, *SIGMOD Conference*, pages 1171–1182. ACM, 2008.

[87] David E Simmen, Mehmet Altinel, Volker Markl, Sriram Padmanabhan, and Ashutosh Singh. Damia: data mashups for intranet applications. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1171–1182. ACM, 2008.

[88] John B Smith. *Collective intelligence in computer-based collaboration*. Psychology Press, 1994.

[89] Josef Spillner, Marius Feldmann, Iris Braun, Thomas Springer, and Alexander Schill. Ad-hoc usage of web services with dynvoker. In *ServiceWave*, volume 5377 of *LNCS*, pages 208–219. Springer, 2008.

[90] Josef Spillner, Marius Feldmann, Iris Braun, Thomas Springer, and Alexander Schill. Ad-hoc usage of web services with dynvoker. In Petri Mähönen, Klaus Pohl, and Thierry Priol, editors, *Towards a Service-Based Internet*, volume 5377 of *Lecture Notes in Computer Science*, pages 208–219. Springer Berlin Heidelberg, 2008.

[91] A. G. Sutcliffe, M. Ennis, and S. J. Watkinson. Empirical studies of end-user information searching. *J. Am. Soc. Inf. Sci.*, 51(13):1211–1231, 2000.

[92] E. von Hippel. *Democratizing Innovation*. MIT Press, 2005.

[93] Eric Von Hippel. Democratizing innovation: The evolving phenomenon of user innovation. *Journal für Betriebswirtschaft*, 55(1):63–78, 2005.

[94] Jen-Her Wu, Yung-Cheng Chen, and Li-Min Lin. Empirical evaluation of the revised end user computing acceptance model. *Computers in Human Behavior*, 23(1):162 – 174, 2007.

[95] Ke Xu, Xiaoqi Zhang, Meina Song, and Junde Song. Mobile mashup: Architecture, challenges and suggestions. In *Management and Service Science, 2009. MASS '09. International Conference on*, pages 1 –4. IEEE Computer Society, sept. 2009.

[96] Jin Yu, Boualem Benatallah, Régis Saint-Paul, Fabio Casati, Florian Daniel, and Maristella Matera. A framework for rapid integration of presentation components. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 923–932. ACM, 2007.