UNIVERSITY OF BARI ALDO MORO

Computer Science Department

DOCTORAL PROGRAMME IN COMPUTER SCIENCE

XXVIII cycle

Scientific Disciplinary Sector INF/01

Design models and interaction paradigms to enable end users to create pervasive workspaces through service mashup

The Chair of the Doctoral Program: **Prof. Donato MALERBA**

Supervisor: **Prof. Maria F. COSTABILE**

Doctoral Dissertation of:

Giuseppe DESOLDA

April 2016

Table of contents

List of Figures

List of Tables

Abstract

The growing amount of resources available on Internet through Application Programming Interface (API) and the opportunities offered by Web 2.0 are pushing end users to evolve from passive information consumers into information producers, able to access such resources and manipulate them, in order to generate new content. This phenomenon demands for new interaction paradigms to enable people to access these contents, get them into personal interactive workspaces where people can integrate and compose them and also create new content to be possibly shared with other people. In this respect, solutions for service composition like the mashup tools play an important role as they let users integrate heterogeneous information that otherwise would be totally unrelated. In such ways, mashups generate new value.

Several mashup tools proposed so far, the so-called mashup makers, provide graphical notations for combining services. As compared to manual programming, such platforms alleviate the composition tasks, but they require an understanding of the integration logic (e.g., data flow, parameter coupling, composition operator programming). Studies with users show that mashup tools are still difficult to use by non-technical users.

The approach investigated in the research presented in this thesis is grounded on experiences reported in literature about new paradigms for mashup composition and on the lessons learnt on End-User Development (EUD), combining the advantages of both fields. According to the EUD vision, enabling non-technical users to create or modify the applications they are going to use requires abstractions and notations adequate for them. The entire work revolves around the development of a mashup platform, EFESTO, that allows the lightweight construction of integrated, situational workspaces pervasively accessible and sharable through a variety of devices such as desktop PCs, mobile devices and large interactive displays. EFESTO supports end users in performing mashups thanks to novel visual interaction paradigms. EFESTO was iteratively refined during the research and allowed the validation of the defined models and interaction paradigms.

Furthermore, mashup platforms are general-purpose and not adequate to the needs of specific application domains, making their use very difficult by end users. The key point of our research was the creation of a general platform for service mashup easily customizable to a specific domain, thus taking advantage of stakeholders' domain knowledge, in order to offer a composition process that makes sense for end users in that domain. In this direction, a first contribution of

this research is a design methodology based on a *meta-design model* and a novel "stratification" of the mashup platform into layers. Moreover, an innovative interaction paradigm is proposed that adopts visual notations defined after user studies based on interviews, focus groups, workshops with users, prototyping testing.

Another contribution is the definition of a polymorphic data source, a rich source of data built on top of the Linked Open Data cloud that can be dynamically modelled by the users depending on their specific informational needs. The added value of this data source is to overcome the limitation of the data sources available nowadays, which only describe a portion of a domain and often do not include many details, so that they do not provide all the information that end users might need.

A further contribution is the integration of the Transformative User eXperience principles in EFESTO. These principles support more elastic data composition and exploration tasks. Finally, a novel approach to data mashup on mobile devices is presented; it uses some cross-device mechanisms that allow end users to formulate queries and reconfigure the data flow between different mobile devices by physically rearranging them on a desk and/or performing cross-device touch gestures on their display.

Chapter 1. Introduction

1.1 Rationale

The problem of facilitating the access to Web services and APIs through visual user interfaces has been attracting the attention of several researchers in the last years. An ever increasing number of resources that provide content and functions in different formats through programmatic interfaces is available, while it is still difficult for laypeople, i.e., users without expertise in programming, to access and exploit the available content.

Recent research projects have been dealing with the problem of easing the creation of effective user interfaces on top of Web services and APIs (e.g., [55]). They focused on the notion of Web Service Graphical User Interfaces (WSGUIs) [79], i.e., on a set of mechanisms to enrich the Web service specifications with annotations that could make easier the definition of visual interfaces. The idea was to automatically generate the presentation layer, starting from enriched service descriptors. Unfortunately, the proposed solutions were only able to dynamically generate dialogs for input and output of structured parameters; this compromised their adoption, especially considering the emerging trend of providing rich user interfaces that go beyond the provision of forms and tables to query a service and interact with the query results respectively.

Service composition paradigms have been proposed since the 90's to create applications that integrate homogeneous resources. More recently, Web mashup methods have been proposed to create web applications by using heterogeneous web resources. Specifically, a mashup is the creation of a web application by integrating data and functions provided by different web services. The most common example of mashup is a web page created by mashing-up the Google Maps API with georeferenced data provided by another API, for example data on upcoming music events. The result of this mashup is an interactive map on which different markers representing the event location are shown.

Mashups support data exploration processes that go beyond one-time interactions and allow users to progressively seek for information. As studied in [86], for data retrieval and exploration tasks typically users invoke general-purpose search engines and/or specialized tools, and then use "their brain" (or suitable cognitive aids, e.g., annotations or clipboards) for taking note of results to be used next. Mashups solve (at least partially) these limitations, as they try to accommodate users' needs for data integration within personal, ad-hoc created workspaces.

Despite these advantages, some factors still prevent a wide use of tools to create mashups, especially by users who are not experts in programming. In fact,

while mashups have been identified as a useful means for application development by end users [34], so far the research on mashups has largely focused on defining integration technologies and standards, with limited attention on easing the mashup development process - in many cases mashup creation still involves the manual programming of service integration. Some user-centric studies also found that, although the most prominent platforms (e.g., Yahoo!Pipes) tried to simplify mashup development, they are still difficult to use by non-technical users, who encounter difficulties with the adopted composition languages [26, 65, 66]. Besides the complexity of the composition paradigm [9], the user interaction for exploring and manipulating retrieved data is still hard.

1.2 Thesis Contributions

The research described in this thesis has permitted the development of a mashup platform, EFESTO, which implements an interaction paradigm that enables users without skills in computer programming to extract content from heterogeneous services and integrate them into newly created applications accessible through different devices. The platform name is inspired by Efesto, a god of the Greek mythology, who realized magnificent magic arms for other Greek gods and heroes. Analogously, the EFESTO platform aims to put in the hands of the end users powerful tools to accomplish their tasks. The proposed approach is grounded on experiences reported in literature about new paradigms for mashup composition and on the lessons learnt on End-User Development (EUD), combining the advantages of both fields.

The performed work resulted in several new research contributions. As first contribution, two models have been proposed: the first one is *a meta-design model*, which guides the development of a general mashup platform that can be easily customized to a specific application domain [9]; the second one is a *UI component model* for modelling abstractions on which the novel composition paradigm adopted by EFESTO is based.

As a second contribution, the EFESTO mashup platform that supports non-technical end users in performing mashups has been developed and validated through field studies in the Cultural Heritage and Technology Enhanced Learning domains [5, 9, 36].

An important factor limiting the use of mashup platforms in real contexts is the difficulty in finding the information that people actually need. The data

sources available nowadays through APIs describe a portion of a domain and often do not include many details. It is sometimes possible to overcome this limitation by composing different data sources, but in some cases, when the end users' information need is more specific, no data source could provide it. Thus, a further novel contribution of this thesis is the definition a polymorphic data source built upon the Linked Open Data cloud, which overcomes this lack of information and better satisfies end users' information needs. This data source is called "polymorphic" because it is able to provide mutable information depending on the situational needs exposed by a given mashup under construction.

The field studies performed during this research revealed some strengths and weaknesses of our approaches and, in general, of mashup tools. One of the most important flaws relates to the limited manipulation of the information retrieved by the user-defined data sources. Transitions across different usage situations, which imply different functionality to be applied to information, should become possible without requiring users to switch between mashup tool and other applications. This means that rigid schemas for information provisioning and fruition, generally adopted by isolated, pre-packaged applications, have to be overcome by instrumenting systems with an intrinsic flexibility. As a possible solution, a framework where mashup composition paradigms are revisited and potentiated through the notion of Transformative User Experience (TUX) has been introduced and represents another contribution of this thesis. The goal is to overcome common application boundaries by enabling user interaction with information in terms of task objects (i.e., data elements, their visualization and specific functions used to perform a task) within dedicated task environments.

Lastly, considering that currently people are shifting from the use of PCs to a massive use of mobile devices, a set of interaction mechanisms, both spatial-aware and cross-device, to perform mashup in situational scenarios by means of mobile devices has been identified. Data sources can be flexibly combined by moving mobile devices around on the desk or by performing touch gestures to quickly express the situational information needs of a group of users. This is the most recent contribution of this thesis. The approach is still a work in progress, but the utilization study reported in the thesis shows the power of such interaction mechanisms.

1.3 Research Methods

The methodologies applied in this thesis are grounded in the User-Centered Design (UCD) model which emphasises the importance of taking end users into account during the iterative system design process. The ISO standard referring to UCD reports a spectrum of methods that designers can exploit to involve users [38]. Great importance is devoted to requirements analysis: designers have to study end users, the tasks they perform and the context in which they operate in order to better foresee how users might likely use the system under development. It is also fundamental to test the validity of designer assumptions with regard to user behavior in real settings tests involving actual users at each stage of the process (i.e., requirements, concepts, pre-production prototypes, mid-production and post-production prototypes), creating a circle of proof back to and confirming or modifying the original requirements. The main difference from other system design methodologies is that UCD tries to improve the system around how users can, want, or need to use the product, rather than forcing the users to change their behaviour to accommodate the system.

Driven by UCD, the most important activities iteratively performed in this research are:

- analysis of the state of the art and outline of lacks or not considered aspects;
- proposal of solutions to fill the lacks (e.g., models, composition paradigms, framework) based on user studies, such as elicitation studies, focus groups, interviews;
- implementation of the proposed solutions;
- validation of the proposed solutions, including models, methods and paradigms, through user studies, such as field studies and utilization studies.

1.4 Thesis Outline

The rest of this thesis is organised as follows:

- *Chapter 2* presents related work about mashups and mashup tools. It first describes the origins of mashup and then analyses the literature on mashups by providing a classification of mashups works according to various dimen-

sions. Then, it surveys mashup tools, also providing a classification along several dimensions.

- *Chapter 3* describes different models defined during this PhD research. The first one is a meta-design model that permits the customization of a mashup platform to a specific domain. This model underlies the creation of software infrastructures that support EUD activities and knowledge co-creation by the different stakeholders involved in system design. A more technical model to guide developers in building mashup tools is also defined. Finally, a framework in which composition paradigms are revisited and potentiated through the notion of Transformative User Experience is illustrated.
- *Chapter 4* illustrates the design and development of a mashup tool called EFESTO. First, a composition environment for EFESTO customization to a specific domain is described. Then, a user study to elicit the composition paradigm implemented in EFESTO is illustrated. Afterwards, a complete description of EFESTO and its architecture is provided. Finally, the EFESTO evaluation performed through two field studies and a utilization study is reported.
- *Chapter 5* illustrates a new polymorphic data source built upon the Linked Open Data cloud that is able to provide mutable information with respect to the composing data sources. First, the motivation to investigate this new data source is reported. Then, the algorithm to automatically annotate the 'traditional' data sources presented in a mashup tool, which is a mandatory step to create the polymorphic data source, is described. In the end, an evaluation of this algorithm and future work with the polymorphic data source are reported.
- *Chapter 6* presents a novel approach to mashup data-sources by means of cross-device mechanisms of mobile devices. Such mechanisms support groups of co-located people holding different devices to 'mashup' information that satisfies their situational needs. The elicitation study carried out to design the cross-device mechanisms is described. A preliminary evaluation carried out during a demo session at an international conference is reported.

Chapter 2. Related Work

2.1 Mashup origins

The World Wide Web consortium defines a web service as *a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols*[1]. This has become an emerging technology thanks to the possibility to develop automated interactions between distributed and heterogeneous applications. In early 2000, researchers started creating Web applications by integrating data and functions provided by different Web services. The web application resulting from service integration was called 'mashup'. This term was originally coined in the music where mashup indicates a song created by blending two or more songs, usually by overlaying the vocal track of one song seamlessly over the instrumental track of another.

Initially, a large number of web applications were built on the basis of Google Maps to plot geo-referenced data on the map. One of the ancestor of these web applications was *HousingMaps*, a site that overlaid Craigslist apartments and housing listings on a map, for some 30 US cities plus London [47] (see Figure 2.1). This site became popular because it highlighted the importance to browse real estate on a map; before that, real estate sites only showed lists of properties. It also promoted the idea that a website could be built by integrating parts of the Web. However, HousingMaps did not include only web services because it aggregated houses data coming from web services and maps "hacked" from Google Maps that, at that time, did not provide its APIs yet.

The first mashups were obtained by means of manual programming, but very soon it became clear the importance of allowing the enormous number of people who are not programmers to exploit the huge amount of services available to create mashups, without requiring to program. This new need pushed the researchers to investigate mashup tools, i.e. interactive systems that permit to access and compose services by exploiting, for instance, visual mechanisms such as drag&drop. For example, in some enterprises specific mashup tools were adopted to guide employees in composing web services. This composition is called *enterprise mashup* [51] because users create dashboards for aggregating and synchronizing widgets to access enterprise data and functions.

[1] https://www.w3.org/TR/2002/WD-wsa-reqs-20021011

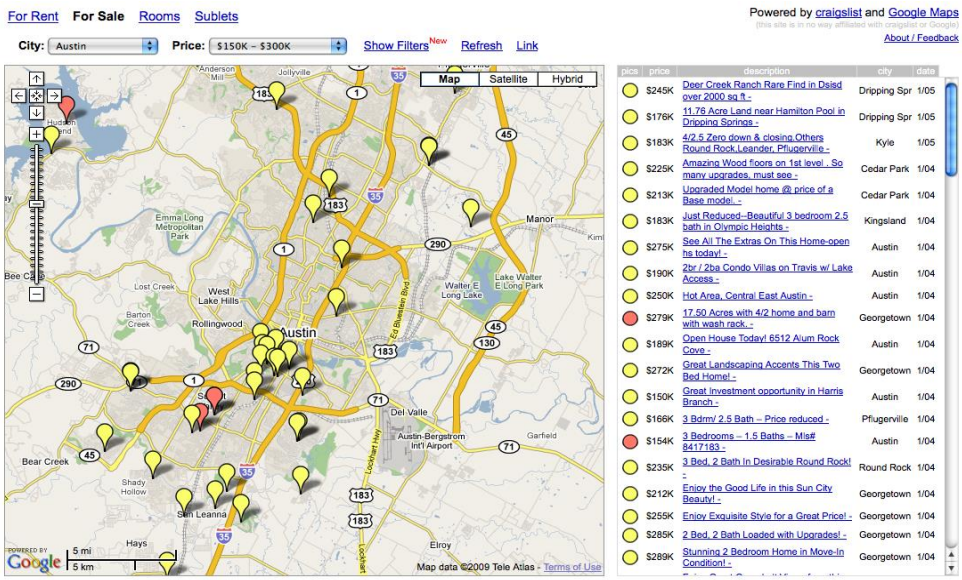

Figure 2.1: HousingMaps screenshot

One of the pioneers of general purpose mashup tools was Yahoo!Pipes [74], a visual editor that provided access to services and operators that could be combined into a canvas pane through drag and drop actions. In particular, services and operators were visualized as visual modules that users could link by means of 'pipes' (Figure 2.2). Moreover, Yahoo!Pipes provided a useful debugger for the users to inspect Pipe output at various stages in the Pipe. The composition results could be exported in other Web sites. Since August 30th 2015, Yahoo!Pipes has been dismissed.

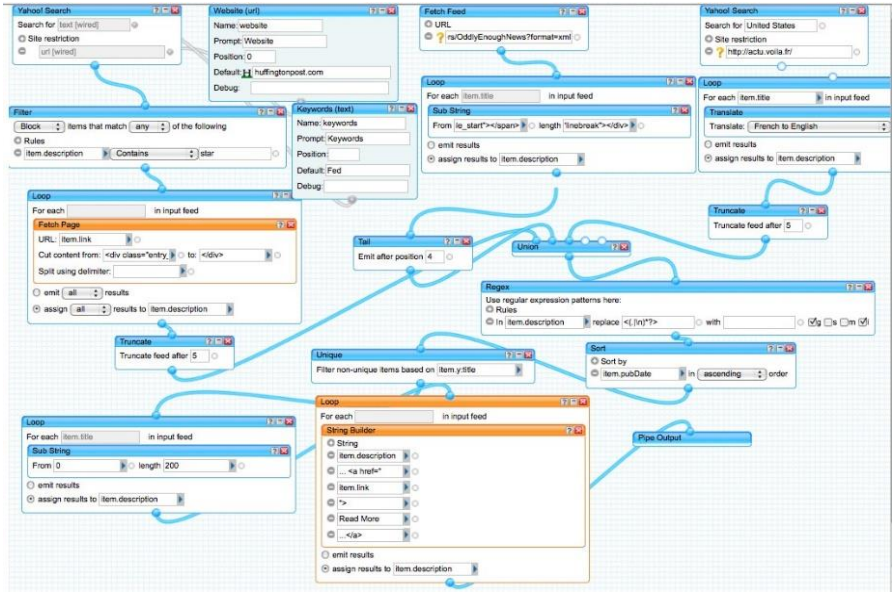

Figure 2.2: Yahoo!Pipes example where services and operators are shown as visual modules and linked by 'pipes'.

In the successive years, several mashup tools based their composition approaches on the Yahoo!Pipes paradigm. However, despite the wire paradigm seemed very promising for not-technical users, some factors limited a wider adoption of these mashup tools in real contexts. Some user-centric studies found that their composition paradigms are still difficult to master for non-technical users because the adopted composition languages do not fit their mental model [26, 65, 66]. This problem pushed researchers in investigating new mashup paradigms based, for example, on event-driven mechanisms [25] or on natural language processing [2].

2.2 Mashup and tools

A book that is a comprehensive reference for mashups, authored by Daniel and Matera, was published in 2014 [34]. The authors systematically cover the main concepts and techniques underlying mashup design and development, the synergies among the models involved at different levels of abstraction, and the way models materialize into composition paradigms and architectures of corresponding development tools. The book takes a balanced approach, combining a scientific perspective on the topic with an in-depth view on relevant technologies.

Some other publications also reported several mashups and mashup tool features. For example, a recent review of tools, languages and methodologies for mashup development is presented in [69]. In this paper the authors identify a set mashup and mashup tool features, taking into account other surveys and literature reviews. In [1] the design space of mashup tools has been proposed. The authors surveyed more than 60 articles on mashup tools, pointing out that only 22 tools are online. Based on these 22 tools, they proposed a model focused on the main perspectives occurring in the design of mashup tools.

In the following two sub-sections, we report a set of dimensions useful for illustrating important features of mashups (Section 2.2.1) and mashup tools (Section 2.2.2).

2.2.1 Mashup

In the book written by Daniel and Matera [34], a mashup is defined as *an application that integrates two or more mashup components at any of the application layers (data, application logic, presentation layer) possibly putting them into*

communication among each other. In particular, a mashup component is any piece of data, application logic and/or user interface that can be reused and that is accessible either locally or remotely. Moreover, mashup logic is the internal logic of operation of a mashup; it specifies the invocation of components, the control flow, the data flow, the data transformations, and the UI of the mashup.

This definition highlights three of the most important mashup aspects. First, the concept of components, i.e. the atomic parts that can be integrated to create a mashup. Second, the application layers in which mashup is performed, i.e. data, logic and presentation layers. Last, the type of mashup the user can perform like control flow, the data flow, the data transformations, and the UI of the mashup. In this book these aspects are presented in a mashup classification model.

In Table 2.1 the mashup dimensions we consider most significant are reported. The firsts five (labelled with *) are taken from the classification model presented in [34]. The dimension *Type of Resource* (labelled with **) is taken from [69]. The last two dimension were added, because they are relevant from our point of view.

Table 2.1: Mashup dimensions. The * indicates the dimensions derived from [34]; the ** indicates the dimension presented in [69]

	Dimensions	**Categories**
Mashup	Mashup Type *	*Data mashups – Logic mashups – User Interface mashups – Hybrid mashups*
	Component Types *	*Data components – Logic components – UI components*
	Runtime Location *	*Client-side only – Server-side only – Both client and server – Cloud*
	Integration Logic *	*UI-based integration – Orchestrated integration – Choreographed integration*
	Instantiation Lifecycle *	*Stateless- Short-living – Long-living*
	Type of Resources **	*Public - Private*
	Domain-Specific Languages for Mashups	*Enterprise Mashup Markup Language (EMML) – Web Services Description Language (WSDL)- Open Mashup Description Language (OMDL) – Mashup Component Description Language (MCDL) Custom grammar*
	Runtime Device	*Desktop – Mobile – Smart Object – Multi-device*

Mashup Type

Following the Media-View-Controller (MVC) software architecture, a mashup can be classified as *data mashup*, *logic mashup* and *UI mashup*, depending on which layers the mashup is spanned.

The **data mashup** refers to the access to different data sources (e.g. Web services, RSS feed) and their composition by means of operations like join, union, filter, sort. The result is a new integrated result set typically published as a new data source. An example is a data source that provides upcoming musical events as a result of a merge of datasets of different Web services like last.fm, SongKick and Eventful. A tool that supports data mashup is Damia, a lightweight enterprise data integration platform where users can create and catalogue high value data feeds for consumption by situational applications [3]. It consists of a browser tool that allows for the specification of data mashups as data flow graphs using a set of operators, a server with an execution engine and an APIs for searching, debugging, executing and managing mashups.

The **logic mashup** regards the integration of functionality published by logic components. An example of logic mashup is the integration of the New York Times RSS feed, which provides a list of recent news (data component), with the Google Translate API that takes these news and produces their version in another language (logic component). The translated results can be saved to a new data source by means of a data storage service. A tool that supports the logic mashup is Microsoft's Popfly: it allows users to create web pages, program snippets and mashups using the Microsoft Silverlight rich internet applications runtime and the set of online tools provided. It was discontinued on August 24, 2009 [57].

The **User Intarface (UI) mashup** consists in combining independent Web services with a native UI into a common UI. A typical example of UI mashup is a dashboard built by integrating different services shown in separated widgets. For example, in Netvibes [68] users can aggregate different widgets to visualize, for example, New York Times news, weather forecast information, a Gmail client, a YouTube search widget, a Facebook wall, a Google Calendar, and so on. The added value of this mashup is the possibility to centralize the functionalities and data that typically users access during their computer-supported activities.

Lastly, the **Hybrid mashups** involve multiple layers of the application architecture. A tool that supports the creation of hybrid mashups is PEUDOM [61]. With this tool, users can aggregate into their dashboard different widgets, each of them connected to a service. Moreover, a synchronization between two or more

services can be established according to an event-driven paradigm and drag&drop mechanisms. For example, last.fm and YouTube Web services, visualized in two separated widgets, can be synchronized in order to automatically retrieve YouTube videos each time a click occurs on a singer name in last.fm results.

With respect to this dimension, the mashup tool proposed in this thesis creates *hybrid mashups.* In fact, end users can aggregate different web services in their workspaces and can exploit data mashup operations to build complex data sources combining different web services.

Component Types

A mashup component is a reusable software module that can be involved in a mashup. Different technologies are used to build a component, for example SOAP and RESTful Web services, RSS Feeds, UI widgets, etc. The strong heterogeneity among the integrated technologies is one of the peculiarity of mashup that, differently by the traditional service composition where homogeneous resources are involved, deals with heterogeneous resources.

Similarly to the *Mashup Type* dimension, also the mashup components are classified into three different classes that span in the three application architecture levels, i.e. *data components, logic components* and *UI components.*

Data components provide read and write capabilities on remote or local data sources. They are typically RSS and Atom feeds, XML or JSON file, CVS file, crawled web data, micro-formats, and SOAP or RESTful services (when used as services to retrieve data). Data components can be static or dynamic. A data component is *static* if it can be invoked without any parameter, as in the case of RSS or Atom feeds. On the contrary, *dynamic* components can be invoked formulating parametric queries. These components are called dynamic since they provide results depending on the query parameter values. In both cases, a UI is needed to interact with them, i.e. to invoke them and insert parameters (in case of dynamic components) and to visualize the results.

Logic components expose business logic or functionalities. They are typically SOAP and RESTful services and JavaScript APIs or libraries. An example of logic component is the Gmail API[2] that allows to send an email by invoking a RESTful method parametrized with recipient email, subject and text, and sender

[2] https://developers.google.com/gmail/api/

parameters. Despite these components offer a powerful opportunity to create mashups, they just provide function and thus they have to be completed by programming a proper UI to interact with them to provide input or visualize the output.

The **UI Components** are services provided with a user interface. They are typically JavaScript UI libraries, Java portlets, code snippets and extracted UI components. In many cases, they also expose functions and data, as in the case of Google Maps API[3] where its integration in a web page provides not only the map, but also function to calculate routes between two points of interest or to search a place by typing an address.

With respect to this dimension, the mashup tool proposed in this thesis exploits *Data components* and *UI components*. The first ones are used to retrieve data from web services (e.g. Wikipedia API) while the second ones are used to visualize data, for example Google Maps API is used to plots geo-referenced data retrieved by compatible APIs, as, for example, the upcoming musical events provided by the last.fm API.

Runtime Location

This dimension is related to the environment in which a mashup is executed. In the original dimension identified in [34], the authors propose three locations: client, server, or client-server. In the last years, the spreading of the cloud technology also influenced the mashup, opening new opportunities to run the mashup in a more powerful location.

The **client-side mashup** regards the composition of mashup components without the use of remote resources. An example is the mashup of JavaScript libraries to visualize data extracted from data components that read CVS files. In this case, all the components, libraries and data, reside on the client. An advantage of this mashup is that no Internet connection is required to run the mashup. However, only mashup components that do not require internet can be used, limiting the number of services that can be integrated.

The **server-side mashup** is the integration of mashup components available on a remote server. For example, a server-side mashup could be the composition of the Twitter API (data component) with a service for sentiment analysis (logic

[3] https://developers.google.com/maps/

component). The first one takes in input a keyword and produces a list of tweets that contain the keyword. The second one takes the list of tweets and the keyword and produces an index on how much Twitter users like the topic associated to that keyword. This mashup is like a black-box (Twitter -> Sentiment Analysis) that can be invoked by providing a keyword to indicate a topic; the execution of this mashup produces an index on the topic indicated by the typed keyword. An advantage of this mashup is that it can be invoked from different clients since the resource reside on a server. However, an internet connection is required.

The **client-server mashup** is a fusion of the previous two runtime locations since mashups components reside on both client and server. In this case, the mashups can take advantage from the peculiarities of client and server.

A **cloud mashup** is an evolution of the client-server mashup. Cloud technology is opening new opportunities to dynamically balance the computing and/or storing resources on different servers. As in server-side mashup the composition resides on a remote machine, in the cloud mashup the composition, and in particular the components and their execution, are distributed on different cloud provider's servers. For example, in the last years, researchers and IT companies are implementing tools to access and compose smart things. Typically, the goal of these tools is the automation of processes carried out by the users (e.g., each time user come into his home the Wi-Fi router switch on). With these tools users can establish a large number of automation and each of them could require a continual communication with smart objects or web services, thus the scalability of the tool becomes a critical issue. To this aim, cloud technology allows the dynamical balance of the processes on the cloud.

With respect to this dimension, the mashup tool proposed in this thesis creates *server-side* mashups since the execution logic, data and UIs reside on a remote server. We opted for the server-side mashup to foster the mashup execution independently from specific hardware configurations and devices; in fact, the end users can execute their mashups on different devices such as smartphones, laptop PCs and large interactive displays [6].

Integration Logic

This dimension regards how mashup components communicate among them. In the classification reported in [34] the author identified three different types of logics: *UI-based*, *Orchestrated* and *Choreographed*.

The **UI-based integration** is a simple aggregation of UI components in the same user interface. One of the first tools that proposed this integration was iGoogle[4], a personal web portal launched by Google in May 2005 and dismissed on November 2013. This tool was a typical example of UI-based integration since users could aggregate into their dashboards different widgets that act without communicate among them.

The **Orchestrated integration logic** is a centralized composition logic that 'orchestrate' the execution of the components involved in the mashup. This logic can also act as a proxy to mediate communication among components.

The **Choreographed integration logic** refers to mashups where components are able to send and receive messages directly between them, without the need of a proxy like in the orchestrated integration logic. These types of mashups are characterized by a communication infrastructure as a message or event bus that is in charge to propagate an event raised by a sender component to a receiver component. An example of this logic is implemented in PEUDOM where users can aggregate widgets in UI-based integration logic, also defining a communication between them to create a 'choreography' of widgets [61].

With respect to this dimension, the mashup tool proposed in this thesis generates mashups with *UI based* and *Choreographed* integration logics. In fact, the mashups are an aggregation of widgets that included web services visualized by means of visual templates (*UI based*, see Section 3.3*).* Moreover, the task containers introduced in the mashup tool (see Sections 3.3 and 3.4) provide the possibility to "act" on the retrieved contents, for example to collect&save favorites, to compare items, to plot data items on a map, to inspect full content details, or to arrange items in a mind map to highlight relationships. To do this, the end users have to drag&drop content from a widget into a container: when an item is dropped into the container, the widget sends a message to the container with the raw data of the dragged content. These data are used by the container to visualize the content in a different fashion (*Choreographed*).

Instantiation Lifecycle

This dimension concerns how long a mashup runs in its location. In [34] the authors identified three types of lifecycle: *stateless*, *short-living* and *long-living*.

[4] http://www.igoogleportal.com/

A **stateless** mashup does not have an internal state during its execution. A typical example is the data mashup. If a user creates a new resource *Y* as the union of *n* data sources, only the composition schema of *Y* resides in the runtime location along the time, but the mashup runs only when it is invoked by the user, without persisting during the user session. Only the mashup results persist along the time. This is the reason why it is called stateless.

A **short-living** mashup lives just during the time of a user session. This is the case, for example, of a UI mashup. In fact, let us assume that a user is using Netvibes to define his dashboard [68]. Every time the user opens his dashboard, the components are instantiated, but when the user session is closed the components are deallocated.

A **long-living** mashup survives across different user sessions. This is a common state in the logic mashups that instantiate a process. For example, the mashup tool called IFTTT allows the creation of 'recipes' (an alias for mashup) to throw actions when events are caught. For instance, with a user-defined recipe, every time a YouTube video is liked on the user account, the video name is saved in a Google Drive sheet [48]. This recipe survives across the user sessions since it listens an event (a like on a YouTube video) to throw an action (save video in Google Drive) even if the user is not logged in IFTTT.

With respect to this dimension, the mashup tool proposed in this thesis generates *short-living mashup*. In fact, each user can save and execute his mashups during different sessions but the mashup components are deallocated each time the mashup tool is closed.

Type of resources

A mashup can be seen as an evolution of service composition where only homogeneous services could be composed. One of the peculiarity of mashup is the possibility to integrate heterogeneous resources in a single application. Two main types of resource, i.e. *public* and *private,* have been identified.

Public resources are components that publish data and function on the Web. The most of these resources are Web services available through SOAP or RESTful technology.

SOAP (Simple Object Access Protocol) was designed as an object-access protocol in 1998 by Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein for Microsoft, where Atkinson and Al-Ghosein were working at the time. It is a protocol specification for exchanging structured information in the imple-

mentation of web services in computer networks. It uses XML Information set for its message format, and relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission. Because of the verbose XML format, SOAP can be considerably slower than competing middleware technologies such as REST architectural style.

REST (REpresentational State Transfer) has emerged as a SOAP alternative to develop Web services. In fact, REST-based Web services provide important advantages with respect to SOAP. First, requests and responses are shorter than in SOAP, since it requires an enveloped XML-based format in every request and response. Second, the bandwidth required to transport requests and responses is lower than that of SOAP. Last, memory and processing resources required in order to process requests and responses are less compared with those of SOAP. Thanks to these advantages, today REST is the standard the facto to develop web APIs or web services, RSS, and Atom data services.

Another category is the **private resources**, i.e. the data and function used by single users or communities of users (e.g. companies). Usually, these data are stored in files like CSV, JSON, XML and HTML or in databases. A novel type of private resources are the smart things that today provide users with data and functions such as heartrate, body temperature, IP camera audio/video streaming, domotic sensors and actuators, and so on. Smart things data and functions can be used and integrated by means of their APIs.

With respect to this dimension, the mashup tool proposed in this thesis produces mashups that include both *public* and *private* resources. In the first case, RESTful web services can be used in the mashup while in the second case CSV files and databases can be included.

Domain-Specific Languages for Mashups

As described in the previous sections, a mashup can be developed by involving different types of mashup components, integrating them following different logics, running it in different environments, and so on. This strong complexity motivated the proliferation of Domain-Specific Languages (DSLs) to describe mashups, for example EMML, Orc, YQL, OMDL and MCDL that are the ones of the most popular languages. In [34] the mashup programming language is not included in the mashup model but the authors deal with it presenting pro and cons of some languages. Since we consider the mashup programming language an im-

portant mashup aspect, in this thesis we included it as a mashup dimension. In the following, a description of some languages is reported. Further details about other languages are reported in [34, 69].

One of the most popular language is the **Enterprise Mashup Markup Language** (EMML), an XML-based language to describe the mashup components integration. In particular, with EMML it is possible to define which components are involved, how to invoke them and how to compose them as data or logic mashup. The EMML schema needs to be processed by an EMML engine that interprets EMML statements and creates the final mashup rendered in UIs or exported to other applications.

Similar to EMML, the Web Services Description Language (WSDL) is an XML-based interface definition language that is used for describing the functionality offered by a web service. The acronym is also used for any specific WSDL description of a web service (also referred to as a WSDL file), which provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns. It thus serves a purpose that corresponds roughly to that of a method signature in a programming language.

```
<!-- Loop for each country, retrieve plants for country and append to
     in-memory store, pause 5 seconds so timestamps are distinct -->
<foreach variable='location' items='$countries/countries/country' >
  <variable name="found" type="document" stream="true"/>
  <assign fromexpr="$location/text()" outputvariable="$searchFor"/>
  <directinvoke  method='GET' stream='true' outputvariable='plants'
    endpoint='http://mdc.jackbe.com/downloads/presto/data/mfgplants.csv' />

  <raql stream="true" outputvariable='found'>
    select Country, Name, Active_Production_Lines,
    Production_Lines_Under_Construction from plants
    where Country = '{$location}'
  </raql>
  <storeto cache='storeAppendPlants' key='#unique' variable='found' />
  <script type='text/javascript' outputvariable="result">
    <![CDATA[
      function pause(millis){
        var date = new Date();
        var curDate = null;

        do { curDate = new Date(); }
        while(curDate-date < millis);
      }
      pause(5000);
    ]]>
  </script>
</foreach>
```

Figure 2.3: EMML example to access to a CSV file and loop on its results in order to retrieve plants for each country and append to a memory-store each result, applying a 5 second pause between each item to obtain different timestamps [73].

Another popular language is **Yahoo Query Language** (YQL), an SQL-like language created by Yahoo! as part of their Developer Network that allows querying, filtering, and joining data across Web services. YQL propose SQL statements to retrieve data (SELECT), filter query results (WHERE), combine filter expressions (AND, OR), join tables, page or limit results, write (INSERT), update (UPDATE), and delete (DELETE) data. JavaScript can be included to be executed by the YQL Web service.

```
select *
from rss
where url in (
              select title
              from atom
              where
              url='http://example.org')
```

Figure 2.4: Example of YQL query to Aggregate and filter multiple RSS feeds

Another language is **Orc**, designed for programming process-oriented mashups. With respect to the previous languages, it requires user to have prior knowledge of functional programming language to write a mashup application, thus it is not very easy to learn. In fact, it is a Web scripting language to create Web service mashups, but it can be used as a general purpose programming language for concise encoding of concurrent and distributed applications.

```
include "search.inc"
each(results)
  <results<
    Prompt("Search for:") >term>
    ( Bing(term) | Google(term) )
```

Figure 2.5: This Orc script starts a Bing search and a Google search simultaneously and prints the first set of responses

The **Open Mashup Description Language** (OMDL) is a simple way to export mashups consisting of pages, layouts and widgets for use in other applications. Alternatively, OMDL can be used as part of a workflow for authoring mashups for users, starting with a conceptual design, adding the structure, then specific widgets and layouts, and then importing the description into a platform such as Apache Rave.

With respect to this dimension, a simplified version of the EMML language has been introduced. The reason is that the composition logic implemented in the EFESTO mashup engine (see Section 4.5.3) refers only to a small sub-set of the composition operators available in EMML.

Runtime Device

At the beginning of the research on mashups in early 2000, only desktop devices have been used to compose and run mashups. In the following, the spreading of mobile devices and smart objects opened new challenges in running mashups on these devices. Since this aspect affects the creation and use of a mashup, in this thesis the runtime device is considered as a dimension for a mashup.

In the last decade, **mobile devices** appeared and pushed researchers to investigate mechanisms to build and execute mashups on them, opening new challenges like the possibility to execute and distribute a mashup on a set of mobile devices. For example, a mashup runnable on a desktop PC, large interactive displays and tablets was developed during this thesis (see Section 4.6.1). Depending on each device, the mashup assumed different aspects and functionality to accommodate the device peculiarities and limitations.

In the recent years, a completely new types of devices are appearing, the so-called **smart devices**. These technologies are opening new opportunities to exploit mashup in a new fashion since smart things are typically available on the web through RESTful interfaces. Recent tools allow users to automatize processes that involve smart things. For example, with Node-RED [81] it is possible to register smart things through their REST interfaces and link them in a graph representation where each thing is a node and their connections are ‘wires’. For instance, a user can connect two personal wearable devices: a smart bracelet that tracks the heartbeat and a smart watch. Thanks to Node-RED, the user can establish a connection that produces a warning on the smartwatch every time the heartbeat overcome a threshold, useful to warn user during sport activities.

The last type of location is the **multi-device** setting, i.e. the possibilities to distribute the mashup on completely different devices, like the ones previously described. For example, the *SmartComposition* approach [54] enables the end users to easily create multi-screen mashups in terms of different widgets distributed and synchronized on different devices like PC, smartphone, smart TV. For example, a teacher can create a distributed mashup to present his lesson with a laptop connected to a projector and deliver additional information to participants’ mobile devices.

With respect to this dimension, the mashup tool proposed in this thesis produces mashups that can be executed on desktop PCs, tablets or a large interactive display (*multi-device*). In fact, thanks to the abstract representation of the mashup

in an EMML-like language, each device can instantiate the mashup in an ad-hoc manner.

2.2.2 Mashup tools

Mashup tools are interactive systems that assist users in developing mashups, i.e. web applications that reuse different resources such as web services. In the last ten years there has been a growing amount of mashup tools, characterized by different features. With the aim of enabling non-programmers to build mashups, one of the most important feature is the interaction paradigm offered to users.

The survey in [1] proposes a model of the mashup tool design space, built on the basis of the identified design issues. In this section, we describe the dimensions that, in our opinion, well characterize mashup tools. They are reported in Table 2.2, indicating with * the dimension derived from the design issues in [1].

Table 2.2: Mashup tool dimensions. The * indicates the dimensions derived from [1].

	Dimensions	Categories
Tool	Targeted end users *	*Non Programmers - Local Developers - Expert Programmers*
	Automation degree *	*Full Automatic - Semi-Automatic - Manual*
	Liveness Level *	*1 – 2 – 3 – 4*
	Interaction Paradigm*	*Editable Example – Form based – Programming by example – Spreadsheets – Visual DSL – Visual Language (Iconic) – Visual language (Wiring, Implicit control flow) - Visual language (Wiring, Explicit control flow) – WYSIWYG – Natural Language*
	License	*Open Source - Commercial*
	Runtime environment	*Desktop – Mobile – Cloud*
	Supported Resources	*RESTful – SOAP – smart things – file – database – CSV – excel – smart things*

Targeted end users

In terms of programming skills, the end users range from *non-programmers* to *experienced programmers*, with in the middle professional end-users without programming skills, but interested in computer and technology, also called *local de-*

velopers [67]. Typically, tools for experienced programmers are very powerful but less usable, on the contrary tools for non-programmers have simplified mechanisms that sacrificing the expressive power of the tools, as summarized in Figure 2.6.

Non-programmers are users without any skill in programming and represent the majority of web users. The tools they are interested in are the ones that don't require to learn/use programming language and technical mechanisms common for ICT experts and engineers (e.g. use of logical operators, complex process flows). Thus, non-programmers should be provided with tools that limit their involvement in the development process to small customizations of predefined mashup templates, or execution of parametrized mashups. An authoring tool for non-programmers has been described in [43]: it supports the development of adaptive user interfaces that reacts to contextual events related to users, devices, environments, and social relationships. In particular, non-programmers can define the context-dependent behavior by means of trigger / action rules.

Local developers are users with knowledge in ICT technology and software usage without having skills in computer programming. Typically, this target of users is willing to explore software and thus tools can provide composition functionality where mashups can be assembled from scratch by composing predefined components or by customizing and changing existing examples and templates. To do this, mashup tools for local developers have to provide a high level of abstraction that ideally hides all the underlying technical complexity of the mashup development. An example is the system presented in [32] where the author proposes a new perspective on the problem of data integration on the web, the so-called surface web. The idea is to consider web pages UI elements as interactive artefacts that enable the access to a set of operations that can be performed on the artefacts. For example, a user can integrate into his personal web page a list of videos gathered from YouTube and he can also append a list of Vimeo videos. This data integration can be improved by means of filtering and ordering mechanisms. These operations can be achieved, for example, by pointing and clicking elements (YouTube and Vimeo video lists), dragging and dropping them into a target page (e.g. personal Web page), choosing options (filtering and ordering).

The **programmers** are users with an adequate knowledge of programming languages. They are the only users that can compose a complex, rich of features, and powerful mashups by means of tools that provide also Web scripting languages for developing more complex and customized mashups.

With respect to this dimension, the mashup tool presented in this thesis is strongly oriented to *non-programmers* and *local developers*. In fact, the main goal of this thesis is to enable non-technical users to easily perform mashup, providing them with a composition paradigm that fits their mental model.

Automation degree

This dimension refers to how much the mashup creation can be supported by the tool on behalf of its users. For this reason, the author of [1] identified two categories: *semi-automation* and *full-automation*. A new category, *manual*, has been introduced to indicate tools without support in mashup creation.

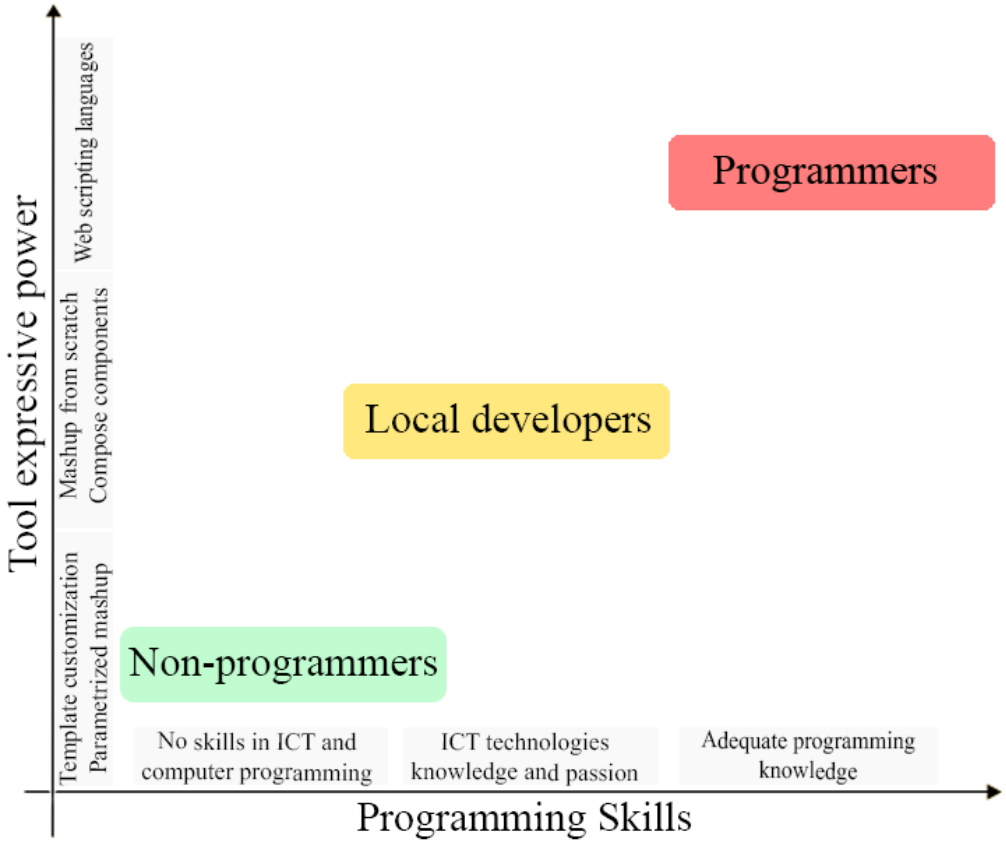

Figure 2.6: A representation of tool expressive power with respect to the required programming skills of users

Tools that support a **semi-automated** creation of mashup partially support users, providing low levels of guidance and assistance. A semi-automated tool requires users to have more skills, but guarantees many degrees of freedom in creating a mashup that satisfies user needs.

A **full automation** in mashup development reduces the direct involvement of users in the development process since users are strongly guided and assisted in

the process, with the role of supervisory of just to provide input or validate mashup results. These tools require a short learning curve and decrease the effort in mashup development. However, these facilities limit the possibility to create a mashup that fits all the user needs. An example of full-automated tool is Natural-Mash, a tool that allows users to express in natural language what services they want into their mashup and how to orchestrate them [2]. To ensure the accuracy of the expressed user queries, NaturalMash narrows the user in a controlled natural language (a subset of a natural language with a limited vocabulary and grammar).

The **manual** category refers to those tools that do not provide any support to the users during the mashup creation. For example, Yahoo! created a console[5] to formulate queries in a YQL language to perform data-mashup. In this tool, no help and assistance are given to the users because they have to formulate their queries following the YQL syntax. If the query is right expressed than the JSON or XML results is produced, otherwise a syntax error is shown.

With respect to this dimension, the mashup tool presented in this thesis supports a *full automation degree*. In fact, the tool composition paradigm is grounded on wizard procedures that guide the end users in creating a widget on top of a web service or in composing different web services by means of operations like join or union.

Liveness Level

The concept of liveness for visual languages presented in [80] was adopted in the mashup domain [1]. In particular, the authors of [1] used the four *liveness* levels to express the tools complexity.

The **level 1** is the *Flowchart as ancillary description*: in this case tools are used to compose a mashup as a non-runnable prototype that is not directly connected to any kind of runtime system. This prototype has just a user interface, but does not implement any functionality. If on one hand these tools don't require technical or programming skills, one the other hand an execution environment is necessary to execute the prototype. Microsoft Visio enables the creation of prototype mashups. The resulting prototypes can be completed with data and executed by Microsoft Excel [88].

[5] https://developer.yahoo.com/yql/console/

The **level 2** is the *executable flowchart:* tools in this category produce mashup design blueprint with sufficient details to give it an executable semantics. The consistency (logical, semantical or syntactical) of the produced mashups can be verified. However, the development of mashup with these tools requires skills in programming since users need to define low-level technical details and thus constraint their use only to programmers. For example, *Activiti*[6] is a lightweight workflow and Business Process Management (BPM) Platform characterized by features such as modeller, validation, and remote user collaboration.

The **level 3** is the *edit triggered updates*: in this case mashup tools are characterized by the development of mashups that can be easily deployed into operation. Users produce their mashups without spending too much effort in the manual deployment typically by using two environments: one for the mashup *editing* and another one for mashup *execution*. The deployment of mashup under development in the editing environment could be obtained, for example, by clicking a run button that produces a deployment in the execution environment. An example of a mashup tool with these features is *JackBe Presto*, a mashup tool characterized by a design environment to model the mashup and the runtime environment used for debugging and monitoring purposes.

The **level 4** is the *Stream-driven updates*: it is assigned to those tools that support live modification of the mashup code, while it is being executed, without differences between editing and execution. In this way, the mashup development is very fast and does not require particular programming skills. This approach was implemented in DashMash, a mashup tool that allows to create and synchronize web services by means of an event-driven paradigm [25] without distinction between editing and execution time.

With respect to this dimension, the mashup tool described in this thesis supports live modification of mashup since it blends into a single environment both the editing and the execution phases (*level 4 - Stream-driven updates*). In fact, the end users edit and run their mashups in the same environment, the tool exactly, without switching between two or more different environments. This mechanism is in line with our goal of proposing a mashup tool for non-technical end users.

[6] http://activiti.org/

Interaction Paradigm

One of the most important aspects that affects mashup tool adoption is the interaction paradigm to compose the mashups. Actually this dimension is called Interaction Technique in [1]. This is one of the most critical aspects that limited the wide adoption of mashup tools in the last years, since the interaction paradigm proposed by several tools was not suitable for non-technical people. In the following, the most adopted interaction paradigms are reported.

The **Domain Specific Language** class includes technical interaction techniques since it refers to script languages targeted to solve specific problems for specific domains. In fact, these languages are characterized by textual syntax, sometimes similar to existing programming languages. Obviously, since these languages are very similar to programming languages, they require users a strong knowledge and skills. An example is *Swashup*, a Web-based development environment for a textual Domain-Specific Language (DSL) based on the Ruby on Rails framework (RoR) [62].

A simpler but less powerful alternative is the class of **visual programming languages**, i.e. programming languages that use visual symbols, syntax, and semantics. In [1], the authors identified two sub-dimensions of visual programming languages: the **visual wiring languages** and the **iconic visual languages**. In the case of wiring languages, mashup tools visualize each mashup component or each mashup operation (e.g. filtering, sorting, merging) as a box that can be wired to other boxes. The mashup tools adopt in most cases the visual wiring mechanism since these are the most explicit thanks to the one-to-one relationship between the control flow and data from one activity to another and visual boxes wired to each other. Tools that implement **iconic visual languages** translate objects of mashup language in visual icons. In this way, if the icons are properly designed, users are facilitated in understanding how to compose a mashup.

The class of **WYSIWYG (What You See Is What You Get)** interaction mechanisms permit to create and modify a mashup on a graphical interface without any need to switch from an editor environment to an execution environment (similar to the Liveness Level 4). These tools are very useful and suitable for non-programmers since users have the mashup creation under control. However, these mechanisms sometime represent a constraint since users cannot access to advanced features like filtering and conversion that are typically hidden in the tool backend and thus non available to the users.

An alternative is the class of **Programming by Demonstration** interaction techniques that allow to program a computer by giving an example of a particular task. Typically, these interactions are very useful to reduce or remove the need to learn programming languages and therefore they are also adopted in the mashup tools' context. Whit these techniques, users can 'show' to the mashup tools how a mashup should be, the tools are then in charge to convert the given example in a runnable program, i.e. a mashup.

Another class of techniques, similar to the previous one, is the **Programming by Example Modification** that consists in allowing users to modify mashup instead of starting from scratch. If the tool provides an adequate set of examples, in most cases the customization of one of the available mashups requires a small effort by users.

An alternative class of interaction technique is the **Spreadsheets**, one of the most popular end-user programming approaches to store, manipulate and display data. Tools that implement spreadsheets are oriented to data mashup, but typically produce data visualization thus they cannot build a mashup with an own user interface.

The last example is the **form-based** interaction. Tools that adopt this interaction ask users to fill out forms to create an object or to edit an already existing one. Since the form filling is today a common practice in the Web for all kinds of users, mashup tools that implement this technique are easy to be used by a wide range of users. However, these tools cannot produce complex mashups.

With respect to this dimension, the mashup tool described in this thesis provides a WYSIWYG interaction mechanism to make more simple the mashup edits. In fact, during the wizard operations that assist the users in editing their mashup, all the web services details are always visible and under the control of the end users in a WYSIWYG fashion.

License

Several mashup tools are conceived as research projects published in public repository and/or available as runnable tools on a site. However, also commercial products appeared over the time, thus, regarding the license perspective, two types of tools can be identified: *open source* and *commercial*.

In the case of **open source** tools, the community of users is composed by projects contributors, i.e. programmers that participate in the tool development, and by end users, i.e. people that just use the tool. As common in many open

source projects, support and quality of mashup tools are sometime quite low since they born as research projects and there are no adequate funds and interests in maintain and update the projects along the time.

In the case of **commercial** tools, the development and update of the tools are performed by ICT companies that provide these tools for free or by paying. For example, in the case of Netvibes [68], the tool can be used for free to aggregate general web services or by paying from agencies and enterprises providing them advanced features like the possibilities to sell social dashboards to clients (agencies) or use personal data inside the dashboard (enterprises).

With respect to this dimension, the mashup tool described in this thesis is the results of an academic research and thus released as open source software.

Runtime environment

Similar to the device location dimension presented in the previous section, different devices can be used to create a mashup with a tool. The **desktop PCs** are the most common environments on which mashup tools run since they are equipped with wide screens that offer enough space to visualize mashup components.

However, in some cases, also **mobile devices** are used to create mashup. For example, the Atooma[7] app transforms a smartphone in a 'personal assistant' since the users can automate all the manual operations they usually perform with their phone, e.g. combine Wi-Fi, Mobile Data, Facebook, Twitter, Instagram, Gmail, and other services. In particular, with the Atooma app the users can simply create automations exploiting an event-action paradigm that allows to define rules following the syntax "IF something happens DO something else".

With respect to this dimension, the mashup tool described in this thesis runs on different environments that include tablets, desktop PCs and large interactive displays. The tool 'fits' the device on which it runs, optimizing the UI and functions depending on the hardware peculiarities and constraints (e.g. display size, interaction methods, etc.).

Supported resources

This dimension is related to the *type of resource* dimension identified in the mashup dimensions. In fact, in order to create a mashup with different services,

[7] https://www.atooma.com/

mashup tools have to support different type of services as the ones previously identified (e.g. RESTful and SOAP). More type of resources the tool is able to support, more flexible and powerful the tool is.

With respect to this dimension, the mashup tool described in this thesis implements a mashup engine that permits the mashup of different data sources such as RESTful web services, CSV files and databases. The modularity of this engine allows to easily integrate the tool with other types of data sources.

Chapter 3. Defined Models

3.1 Introduction

This chapter illustrates the models that clarifies and systematize the main conceptual contributions of this PhD research. In particular, the first contribution is a *meta-design model* that depicts the stratification of a mashup platform into layers, to foster its customization to specific domains and thus facilitate its adoption by non-technical people. The second model focuses on the components of a mashup tool and is specifies the main abstraction that drive the development of mashup tools. The last model refers to a framework for mashup composition paradigms revisited and potentiated through the notion of Transformative User Experience. The integration of these principles aims to overcome rigid schemas for information provisioning and fruition, generally adopted by isolated, pre-packaged mashup tools.

3.2 A Meta-Design Model to Customize a Mashup platform to a Specific Domain

The overall objective of the research presented in this thesis is to investigate models, methods and architectures to empower people, who are not software developers and have diverse needs, to create personal, interactive and pervasive workspace by integrating heterogeneous contents and artifacts.

The research is driven by two problems that limited the mashup spreading over the time: first, the complexity of composition paradigm, as demonstrated during studies with users (e.g. [65]) and second the high generality of the proposed mashup tools [26, 65]. With respect to the last aspect, the existing mashup platforms are usually general-purpose tools that are not adequate to the needs of specific application domains. Therefore, it is very important to identify methods and techniques to customize these platforms to specific domains in order to facilitate their adoption by non-technical people, as discussed in [26, 65]. For this reason, our research started with the definition of a meta-design model that permits the customization of our general mashup platform to a specific domain. Our model is inspired by meta-design principles presented in [7, 30, 41] and it is based on the Software Shaping Workshop (SSW) model described in [28, 29] .

3.2.1 Meta-Design principles

A meta-design process is characterized by two main phases. The first phase consists of creating the design environments that allow system stakeholders to

participate in the design (meta-design phase). Most of these stakeholders are non-technical people. Thus, they use software environments adequate to their skills and to the tasks they have to perform. The second phase consists of the design of the final applications, carried out as joint work by the various stakeholders, who collaborate through their design environments (design phase) [7, 30]. Thus, professional developers (software engineers) face new challenges, since they have to create software environments that can in turn empower non-technical people to shape the software they use, without obliging them to become programmers.

The proposed approach addresses all such lines of action, since it is principally aimed to empower people to create personalized interactive environments for information fruition. This is also in line with the so-called culture of participation, to which a lot of attention has recently been devoted [37, 40, 50]: it promotes a shift from consumer cultures, where produced artifacts are passively consumed, to participatory approaches that greatly exploit computational media to support collaboration and communication, providing users with the means to become co-creators of new ideas, knowledge and products that can satisfy their specific needs [71]. Indeed, we propose a redefinition (and also a seamless fusion) of roles that go beyond the conventional user-designer dichotomy, in a context where system design and system execution are interwoven to let users create, immediately execute and iteratively evolve their own applications.

3.2.2 The Meta-Design Model

Our approach for the mashup tool customization to a specific domain is contextualized within a meta-design approach, based on the Software Shaping Workshop (SSW) model [28, 29]. This model underlines the creation of software infrastructures that support EUD activities and knowledge co-creation by the different stakeholders involved in system design. All stakeholders of an interactive system, including end users, are “owners” of a part of the problem: software engineers know the technology, Human-Computer Interaction (HCI) experts know human factors, graphic designers know how to create an appealing graphical design, domain experts know the application domain and end users know their goals. Most of these stakeholders are non-professional developers. In order to contribute to system design by bringing their own expertise, all these figures need different software environments, specific to their culture and skills. The professional developers involved in traditional design actually become meta-designers, who create software environments, called Software Shaping Workshops (in short SSWs or

workshops, intended as a virtual laboratory whose users shape software), through which the other stakeholders, acting at some point as designers, contribute to shaping software artifacts. They create and modify elements (objects, functions, user interface widgets) of the system of interest and exchange the results of their activities to converge to a common design and to allow end users to adapt the software to fit their specific needs. In a similar way, various communities of stakeholders are involved in the different phases of the workspace life cycle.

With the adoption of this model for the mashup tools customization, three different design layers have been identified. At each layer, activities of meta-design are performed, or a mix of design and use activities, depending on the different stakeholders involved. Indeed, professional developers perform meta-design, since they create the software environments (SSWs) for all the other stakeholders involved in the design and implement and/or modify the software artifacts that require programming efforts (top level in Figure 3.1). In order to facilitate the mashup by end users, in our preliminary studies we soon realized that the mashup tools (bottom level in Figure 3.1) have to be customized to their needs. This introduces another layer of activities to be performed by other stakeholders (middle level in Figure 3.1). Professional developers and domain experts collaborated in meta-design activities to customize the general-purpose tools, by registering relevant resources, implementing adequate visual templates and packaging resources for the end users, e.g., the professional guides. Such collaboration is essential for a successful customization.

The customization to a new application domain is usually performed once; it is possibly updated to satisfy specific needs emerging later, e.g., to register or to combine further resources. Thus, the work of more stakeholders is required in order to create tools for non-technical users. This is true in various contexts, as discussed in [22, 42]. Some stakeholders perform meta-design activities (e.g., professional developers, service management experts), even if they are non-technical people (e.g. domain experts), since they create environments and tools that allow others to be de-signers; end users usually perform a mix of design and use activities.

With this model, starting from general-purpose mashup tools, we identified how some modelling abstractions, which guide the integration of contents within container visualizations, enable different stakeholders to package ad-hoc resources and to co-create mashup through intuitive visual paradigms. In particular, end users are facilitated in their composition activities by the availability in the platform of domain-specific resources and user interface templates, which guide their activ-

ities by providing basic visual elements that they can easily manipulate. Moreover, thanks to the adoption of platform independent modelling abstractions, the structure of the composed applications is specified in automatically generated schemas that can be deployed on multiple devices, thus promoting the pervasive fruition of the created mashup and also facilitating their sharing [4]. Such a separation of concerns, together with the possibility to extend the platform with ad hoc visual templates and the ease of packaging ad hoc content resources, facilitates the platform customization to specific application domains.

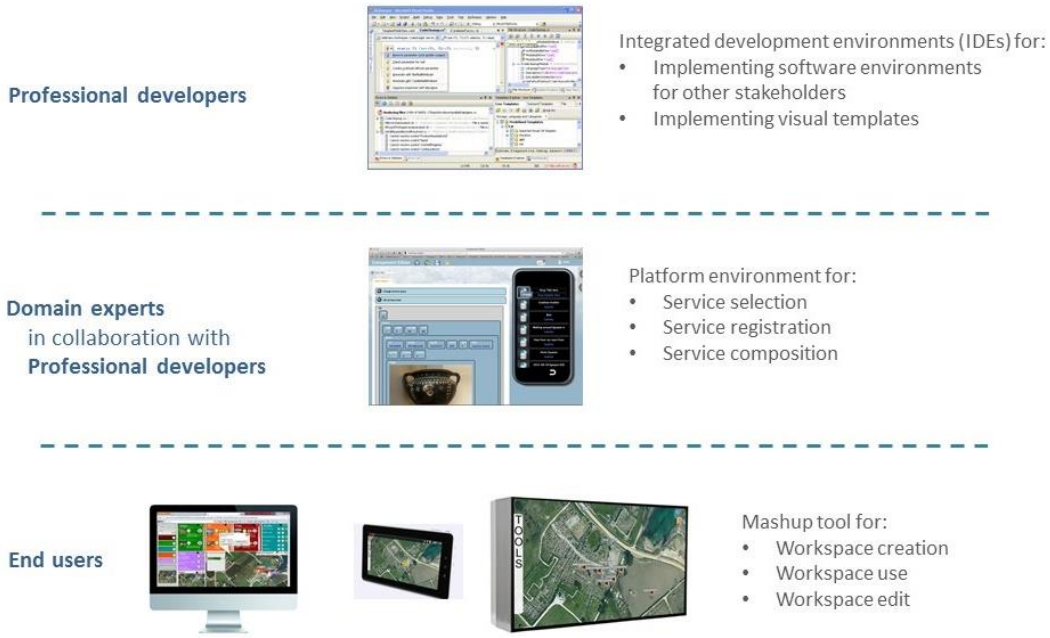

Figure 3.1: Meta-design approach to mashup creation. The bottom layer outlines the composition environments for the end users (deployed on different devices) and the middle layer the composition environments for domain experts: the top layer instead highlights the role

3.3 Model for UI Component Mashup

After the definition of the meta-design model, a mashup platform to validate our model during two field studies (Section 4.6) has been implemented. For the design and development of our platform, we took into account the model presented in [24], where the authors defined the modelling abstractions on which their composition paradigm is based. This model has been chosen as starting point for our model since it fosters the creation of mashup for non-programmers. During the research, our model has been iteratively refined by adding new components on the basis of our research directions, e.g. a different way to integrate service data by

means of more powerful data mashup operations as join and union (Section 4.3), the use of task containers that integrate the Transformative User eXperience principles (Section 3.4), and the polymorphic data sources based on Linked Open Data (0). The new model is depicted in Figure 3.2. In particular, with respect to the original model reported in [24], in the new model the components are:

- coloured white if they have the same meaning;
- filled with oblique lines if they have different meanings;
- coloured in grey if they are removed in our model;
- filled with vertical and horizontal lines if they are added in our model.

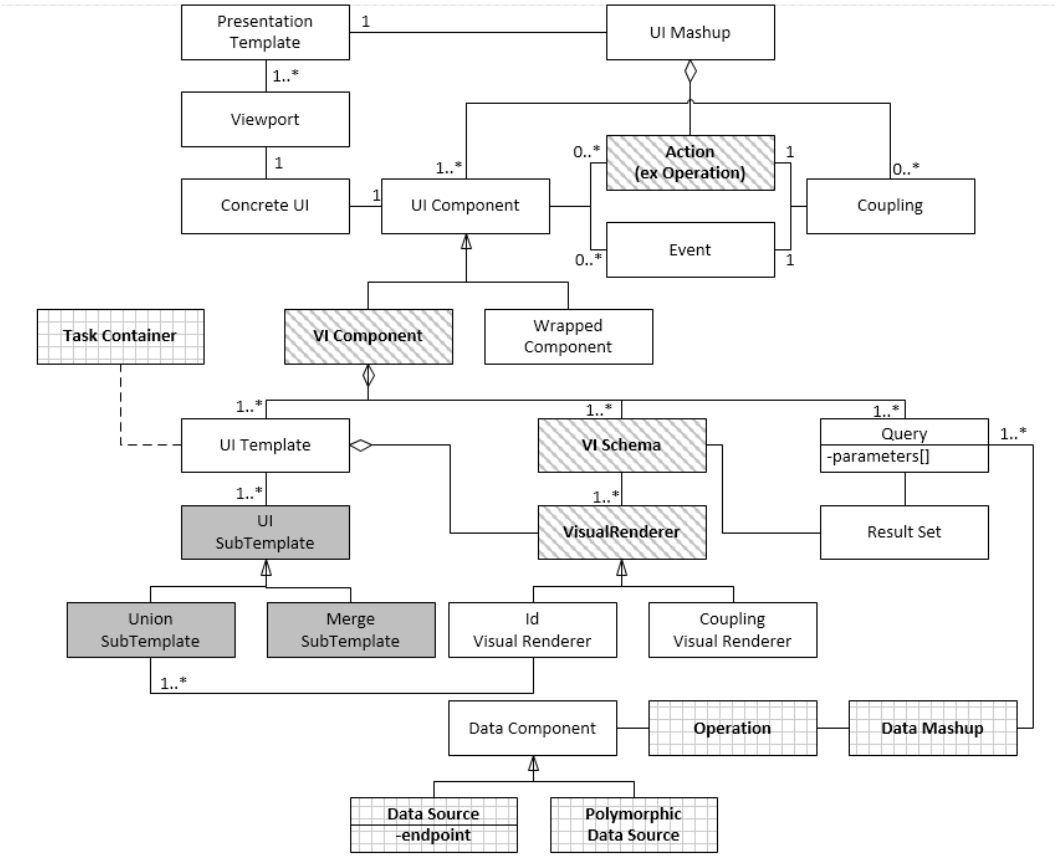

Figure 3.2: A model of the components of our mashup

In the following, a formal definition of each component is reported. The definitions about the white components are the same of [24] and are reported to simplify the reading and model understating.

Definition 1. ***UI Component.*** A UI Component *uic* is a self-contained software module that is bound to one or more services. It also provides data and/or functionality and it is equipped with its own user interface (UI) (its concrete view).

Figure 3.3: Example of UI Component that show musical events on Google Maps

A UI Component also exposes an event-driven logic characterized by a set E of events that can be generated by the user interaction with its concrete view, and a set A of actions 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 component'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 APIs protocols.

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

$$UI\_Mashup =< UIC, C, PT>,$$

where UIC is the set of UI Components involved in the mashup, C is the set of *components' couplings* that determine the synchronized behaviour of components within the mashup and PT is the presentation template adopted to organize the UI of the mashup. Component couplings are defined based on an event-driven,

publish-subscribe integration logic [25, 90]. Couplings are channels for inter-component communication, based on which the occurrence of a published event causes the execution of a subscribed action, 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

$$c=< uic_s\ (<output\_parameters>),\ uic_t\ (<input\_parameters>)>$$

representing the subscription of an *action* of the target UI Component, uic_s, to an *event* raised by the source UI Component, uic_t, and more specifically to the output parameters the event might transport.

Definition 4. Presentation Template. Given a set of UI components in a *UI_Mashup*, a presentation template PT is a set of abstracted UI representations that describe the visual organization of the UI components in the user interface of different devices. For example, in Section 3.2.2 we specified that, thanks to the adoption of device independent modelling abstractions, the structure of the composed applications is specified in automatically generated schemas that can be deployed on multiple devices. The translation from the abstract mashup representation to the concrete UI is performed thanks to a specific *pt* that is different with respect the device that executes the mashup.

a b c

Figure 3.4: An example of different presentation templates used to execute the same mashup on different devices. In *a* the mashup runs on a desktop device. In *b* the mashup runs on a large interactive display. In *c* the mashup runs on a tablet.

Definition 5. Data Component. A Data Component is an abstract representation of a resource that can be used to retrieve data. In particular, dc is a tuple

$$dc=<t, I, O>$$

where *t* indicates the type of resource, for example *Data Source* and *Polymorphic Data Source* in our model. *I* indicates the set of input to query the resources. *O* indicates the set of output features, for example, the instance attributes.

Definition 5.a Selection. Given a data component *dc*, a selection is a unary operator defined as:

$$\sigma_C\ (dc) = \{r \in dc \mid result\ r\ satisfies\ condition\ C\}.$$

where r is a result obtained by querying the data component dc and C is a condition used to query dc.

Definition 5.b Join. Given a couple of data components $dc_i = <ep_i, q_i, A>$ and $dc_j = <ep_j, q_j, B>$, a *Join* is a binary operator defined as:

$$dc_i \mid >< \mid_{ai} dc_j = \{(a_1, \ldots, a_n, \sigma_C\ (dc_j)) \mid C: q_j = a_i\ \}$$

Definition 5.c Union. Given a couple of data components $dc_i = <ep_i, q_i, A>$ and $dc_j = <ep_j, q_j, B>$, a *Union* is a binary operator defined as:

$$dc_i\ U\ dc_j = \{\ x \mid x \in dc_i\ or\ x \in dc_j\ \}$$

Definition 6. Data Mashup. A Data Mashup is a data integration performed between different data components. It is a pair

$$dm = <DC, O>$$

where DC represents the set of data components involved in the composition; O is the set of operations (e.g. join and union) performed between data components in DC. This component represents one of the first improvements performed to the original model presented in [24] where data mashup was conceived just as a visual aggregation of different data sources by means of union and merge sub-templates. In that case, the data mashup could not be reused with other templates since the users need to perform another mashup. In our model, the data mashup follows the traditional vision of data mashup where the result is a new integrated result set published as a new data source. This new data source can be used in the platform as a new source that can be visualized by using visual templates.

Definition 7. Data Source. A Data Source is a triple

$$ds = <ep, Q, A>$$

where ep represents the service *endpoint*, e.g., the URI of a RESTful service, and Q represents the set of *pre-defined parametric queries Q* over the data services. Moreover, ds is characterized by a set $A=\{A_1 \ldots A_n\}$ of attributes that characterizes each query result.

To enable the invocation of a data source at runtime, its endpoint and the queries are specified in a descriptor created when the component is registered into the platform. The registration is supported by a form-based user interface where the user specifies the needed configuration data; configuration files are then generated. For example, the Last.fm data component used in our reference scenario is defined on top of the Last.fm RESTful API. Its registered endpoint is the URI http://ws.audioscrobbler.com/2.0/ where the API is published. Q includes some parametric queries, for example, the one based on an operation to retrieve music events located in a specified city. Queries are expressed as HTTP GET requests, for example, "?method=geo.getevents&location=location_str&api_key=lastfm_api_key." At runtime, requests are instantiated by considering actual parameter values specified during the composition or also during the component execution, for example, the value "Chicago" for the location_str parameter and the API key 0697XXX.

Definition 8. UI Template. A *UI Template* can be characterized as the triple

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

where:

- *type* is the template class (e.g., list, map, chart) selected by the user;

-*VR* is a set of *visual renderers,* vr_k, i.e., elements that provide visual placeholders for single data attribute A in of dc. 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 ofevents.

Definition 9. VI Schema. A Visual Integration Schema is 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. A VI Schema can be formalized a tuple:

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

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,

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

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.

The VI schemas need to be adequately codified in order to be deployed in the mashup tools. In literature there are different languages that support this problem. One of the most popular language is the *Enterprise Mashup Markup Language* (EMML), an XML markup language for creating enterprise mashups that consume and integrate data from variety of sources, often performing logical or mathematical operations. In the platform presented in this thesis, an XML based language inspired by EMML has been introduced (see Section 2.2.1). The VI schema has been divided in 4 descriptors: 1) to described the data mashup (Figure 3.5); 2) to describe the data sources in terms of inputs and outputs (Figure 3.6); 3) the UI template descriptors (Figure 3.8); 4) the mapping between the data mashup and UI template (Figure 3.7).

In Figure 3.5 it is reported an example of our XML language. This file reports a UI component, in particular a data mashup consisting in a union between two services (YouTube and Vimeo) and a join of the unified services with a third service (Wikipedia). In the XML file, the tag *unions* has two children, *services* and *shared*. The *services* tag summarizes the unified services. Each service is reported in a *service* tag. In particular, the *service* tag has the attribute *name* that indicates the name of the data source. This value is used by the mashup tool to retrieve the source details to perform the query. The *shared* tag describes the alignment of the attributes of the unified data sources. For example, it has two children called *shared_attribute*, each of them with two children, *attribute*, that represent the service attributes that are mapped in a *UI template.*

```
<composition join = 'true' union ='false'>
    <unions>
        <services>
            <service name="youtube">
                <attribute name="Title">title.t</attribute>
                <attribute name="Link">link[0].href</attribute>
                <attribute name="Author">author[0].name.t</attribute>
            </service>
            <service name="vimeo">
                <attributes>
                    <attribute name="title">title</attribute>
                    <attribute name="id">id</attribute>
                    <attribute name="owner">path</attribute>
                </attributes>
            </service>
        </services>
        <shared>
            <shared_attribute name="Title">
                <attribute from_service="youtube">title.t</attribute>
                <attribute from_service="vimeo">title</attribute>
            </shared_attribute>
            <shared_attribute name="Link">
                <attribute from_service="youtube">link[0].href</attribute>
                <attribute from_service="vimeo">id</attribute>
            </shared_attribute>
        </shared>
    </unions>
    <joins>
        <join type='composition'>
            <service name = "wikipedia"></service>
            <input>Title</input>
            <extendedAttributes>
                <attribute name="Title">title</attribute>
                <attribute name="URL">fullurl</attribute>
            </extendedAttributes>
        </join>
    </joins>
</composition>
```

Figure 3.5: An example of UI component descriptor codified with our XML language

Each service reported in the *service* tag of our VI schema is detailed in a separate service descriptor XML file. In Figure 3.6 the YouTube service descriptor is reported: inside the root tag called *service*, there are the tags *source*, *inputs*, *params*, *attributes* and *flags*. The first three nodes represent all the information useful to query a data source. The fourth node, *attributes*, described the attributes that detail each result. The last node, *flag*, is introduced to solve the heterogeneity problem of the data sources. In fact, the data sources typically send the results by formatting them in a JSON file but the list of results if formatted in different ways (e.g. inside a json array).

```
<?xml version="1.0" encoding="UTF-8"?>
<service name="youtube" type="JSON" Auth="none">
    <!-- Fonti per il reperimento dei dati del servizio (estensione in ampiezza)-->
        <source name="youtube" url="http://gdata.youtube.com/feeds/api/videos?"
        type="GET">http://gdata.youtube.com/feeds/api/videos?</source>
        <inputs>
            <input name="q"></input>
        </inputs>
        <params>
            <param name="key">AI39si5VLnjvFjVL0zXIzJwl1i1GcTRdfEIxfE76u2ssYw</param>
            <param name="alt">json</param>
        </params>
        <separator>&amp;</separator>

        <attributes>
            <attribute name="title" path="title.t">Vasco Rossi in concerto</attribute>
            <attribute name="link" path="link[0].href">www.google.it</attribute>
            <attribute name="author" path="author[0].name.t">Il vaschista</attribute>
        </attributes>

        <flags>
            <flag name="is_array">false</flag>
            <flag name="array_name">$.feed.entry[*]</flag>
        </flags>
</service>
```

Figure 3.6: An example of service descriptor codified with our XML language

Another XML descriptor introduced in our model regards the UI Template. In Figure 3.7 the *list* UI Template has been reported. It is characterized by a set of sub-UI templates (different types of lists). In particular, the root node, *template*, has an attribute name that indicates the template name. The root has a set of children that describe different alternatives to visualize the UI template.

```
<template type="list">
    <sub_template name="list_A">
        <column width="30%">
            <component name="0"
                       height="20%"/>

            <component name="1"
                       height="80%"/>
        </column>

        <column width="70%">
            <component name="2"
                       height="100%"/>
        </column>
    </sub_template>

    <sub_template name="list_B">
        <column width="100%">
            <component name="0"
                       height="20%"/>

            <component name="1"
                       height="80%"/>
        </column>
    </sub_template>
</template>
```

Figure 3.7: An example of mapping descriptor codified with our XML language

The UI template descriptor is linked with the VI schema through the XML mapping descriptor. An example of mapping is reported in Figure 3.8. In this descriptor, the root node, *mappings*, has two attributes: *templatetype* and *templatename*. The first one recall the name of a UI Template (e.g. *list*), the second one the name of its sub-template (*list_A*).

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings templatetype="list" templatename="list_A">
  <mapping vr="0" attribute="Title" path=".Title" />
  <mapping vr="1" attribute="Author" path=".Author" />
  <mapping vr="2" attribute="Video" path=".Video" />
</mappings>
```

Figure 3.8: An example of UI template descriptor codified with our XML language

Definition 10. ***VI Component.*** *Visual Integration Components* are created by the users mapping visually result sets extracted by one or more data components to UI templates. A VI component can be defined as the tuple

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

where *vis* is a Visual Integration Schema 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 for UI synchronization. E ⊆ templEUItemplate, that is, E is derived from the events associated with the UI template (see Definition 8). For example, for the My events component, E includes the selection of the list items in the merge subtemplate. O ⊆ selQ, that is, O is derived from the set of queries exposed by the involved components. In the My events component, an operation that updates its status is, for example, the one that queries the underlying data sources searching for events based on a specified city name.

Definition 11. RDF Graph. A novel contribution of this thesis is the possibilities to use the Linked Open Data cloud as new data source. In 2009, Tim Berners-Lee defined Linked Data as "a set of best practices for publishing and connecting structured data on the Web" [18]. The goal of the Linked Data project is to publish data in a way readable by a human and by an automatic agent. The idea is to use the HTTP URIs to denote *things* (e.g., a person, a city, an organization) and to provide useful information about them by exploiting RDF[8] standard. In particular, with RDF each thing is coded by a set of triplets in the form

$$t = <Subject, Predicate, Object>$$

[8] http://www.w3.org/RDF/

The subject (S) indicates the thing and it is an URI. The object (O) can be a literal or another thing identified by a URI. The predicate (P) indicates what kind of relation exists between subject and object, e.g., a name of a person (in the case of a literal), or a friendship with another person (in the case of another resource). The predicates are identified by a URI and come from vocabularies used to represent information about a domain. The LOD are Linked Data distributed under an open license that allows its reuse for free. At the time of this thesis, there are more than 1000 KB available in the LOD cloud. An example of a RDF graph is reported in Figure 3.9.

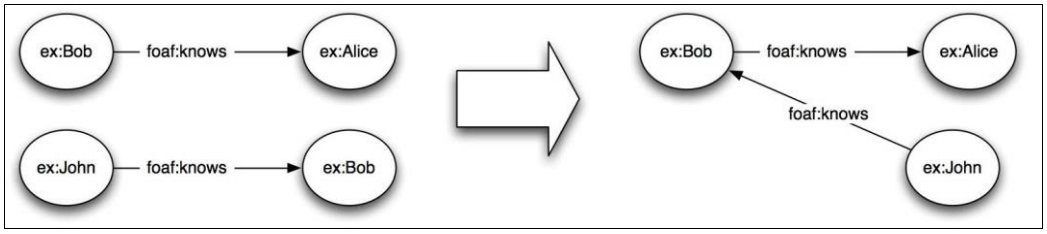

Figure 3.9: A graph representation of a RDF triplets.

The representation in RDF language of the graph in Figure 3.9 is:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:ex="http://example.org/"
    xmlns:foaf="http://xmlns.com/foaf/0.1//"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about="http://example.org/John">
        <foaf:knows>
            <rdf:Description
rdf:about="http://example.org/Bob">
                <foaf:knows
rdf:resource="http://example.org/Alice" />
            </rdf:Description>
        </foaf:knows>
    </rdf:Description>
</rdf:RDF>
```

Nowadays, one of the widest knowledge base (KB in the following) in the LOD cloud is DBpedia (the RDF version of Wikipedia). The DBpedia English version describes 4.58 million things, out of which 4.23 million are classified in its ontology. In the DBpedia ontology, there are 685 classes. Thanks to the availability of this huge amount of information and their semantic structured in the

DBpedia ontology, DBpedia has been chosen as starting point to create the data sources based on LOD described in Chapter 5.

DBpedia, as well as all the knowledge bases in the LOD cloud, is an RDF graph *T* that can be defined as:

$$T = \{ t \mid t \in S \times P \times O \}$$

Definition 12. Polymorphic data source. A polymorphic data source is a novel type of source built on top of a knowledge base in the LOD cloud (DBpedia in our case). In particular, given a set of ontologies *ONT* and a knowledge base represented as RDF graph T, $\forall$ s $\in$ S instance of a class C of *ont* $\in$ ONT, a polymorphic data source is defined as:

$$PDS = \{ t \in T \mid \exists s_t \text{ instance of } C\}$$

where C is a class of an ontology ont $\in$ ONT. In other words, a PDS is like a selection of triplets of a LOD KB whose subjects are labelled with a specific class C of an ontology ONT. For example, starting from DBpedia, it is possible to build a source where all the instances of a musical artist are available. As explained in 0, it is possible to build this PDS from scratch or extending a data source by means of join operator.

Definition 13. Task Container. Task containers are visual elements whose role is to supply task-related functions for manipulation and transformation of task objects (data items retrieved from a source) along user-defined task flows. A task container can be defined as a couple:

$$tc = \langle TF, TV \rangle$$

where TF is the set of *task functions* for manipulation and transformation of task objects along user-defined task flows while TV is the set of UI templates to visualize the tasks objects. This aspect has been described in more detail in Section 3.4 where a framework for TUX principles integration in a mashup tool has been illustrated.

3.4 A Framework for Actionable Mashups

The last model reported in this chapter is a framework for mashup composition paradigms re-visited and potentiated through the notion of Transformative User Experience (TUX). In fact, the field studies performed during this research (Section 4.6) revealed some weaknesses of our approaches and, in general, of mashup

tools [5], related to the manipulation of the information retrieved by the user-defined data sources. The integration of TUX principles aims to overcome rigid schemas for information provisioning and fruition, generally adopted by isolated, pre-packaged mashup tools.

This section describes how to extend the coverage of mashups by augmenting information exploration, generally operated on top of mashup data sets, towards more active prosumption (i.e., genuinely merging "production" and "consumption") and sense making. The important feature we focus on is to support the accomplishment of sophisticated sense making tasks on the visualized information thanks to additional manipulations driven by task semantics. In other words, we aim to enable a kind of active sense making, in which the presented information can not only be viewed differently and in meaningful ways towards the gaining of insights, but moreover transformed effectively towards the actual accomplishment of task goals. In this regard, the visualizations of data retrieved from data sources, that in a mashup environment can occur by means of UI templates, are enriched by augmenting the UI templates with the notion of TUX task containers, i.e., elements whose role is to supply task-related functions for manipulation and transformation of task objects along user-defined task flows [16]. As a consequence, through task containers and their particular task semantics, users are empowered to interact with the displayed information in a contextual manner, thus raising information in mashups to the level of task objects the user can act upon.

As represented in Figure 3.10, system objects (i.e., data items), resulting from the mashup, and their visualizations within UI templates (UI objects) can be promoted to the role of task objects that in turn can be endowed with and treated according to the various task functions offered by the containers in which they are cast. Task objects - not simply data items or their representation in UI templates - become the very objects of user interaction, with the result that the users are not only allowed to consume the information displayed by the mashup, but they are also enabled to manipulate and transform it, i.e., to prosume it, in accordance with the tasks they intend to perform. In principle, mashups – without considering TUX principles – can be equipped with some functionality that has task-semantic character, exceeding the mere modification of data visualizations. Yet, in such cases the task semantics would reside in the application implicitly and in a rather hard-wired fashion. For example, a component for the visualization of products could be enriched with a functionality to send emails to vendors. However, this would be a hard-coded function, which the users could not adapt flexibly into their spontaneously defined task flow. According to TUX, it would be instead

possible to apply the communication capability to other object types, for example to submit inquiries on the products to consumer forums.

Figure 3.10 illustrates the organization of a framework supporting this new task-centric perspective on the organization of an EUD system based on the mashup paradigm. Modules supporting mashup composition and execution are integrated with modules for the manipulation of task objects according to TUX principles. Typical mashup modules are exploited to create the base of UI objects to be then manipulated as task objects. Within the mashup engine, the data access module extracts data from the services on which the system relies on (by means of the mashup components [34]). The integration module interprets user composition actions performed at the UI level and creates an execution model determining how system objects have to be integrated. The results, i.e., the integrated system objects, are rendered as UI objects within UI templates. Such UI objects provide the actionable information on which task functions can be applied. In this sense, UI objects are promoted to the level of task objects by virtue of the functionality provided by the task-semantic layer. In Figure 3.10, the dotted-dashed line connecting a UI template (used for rendering various views of UI objects) and a task container (hosting task objects) makes this promotion explicit.

The task-semantic layer then provides for the identification of the current task contexts, based on the interpretation of user actions as they manipulate task objects by applying container-specific task functions; it also supports the casting of task objects within and across various task containers. At the UI level, a task container “wraps” mashup UI templates, so that the user can act on the displayed UI objects by means of the task-related manipulations. This results in treating UI objects as task objects by virtue of their interpretation through the context, which is defined and provided by each task container. Different UI templates within a task container can be used for providing different views of the same task objects without changing however the semantics of the objects as implied by the task container. Changes of views would in fact still be in line or even supportive of the particular task semantics.

It is worth noticing that, in order to associate different task semantics to data extracted from heterogeneous resources, it is important to maintain continually the relation of the elements representing the task objects to their original context. According to the framework shown in Figure 3.10, establishing and maintaining the identity of task objects (as data returned by a given resource) is supported by the task context engine, in particular by its identity management component.

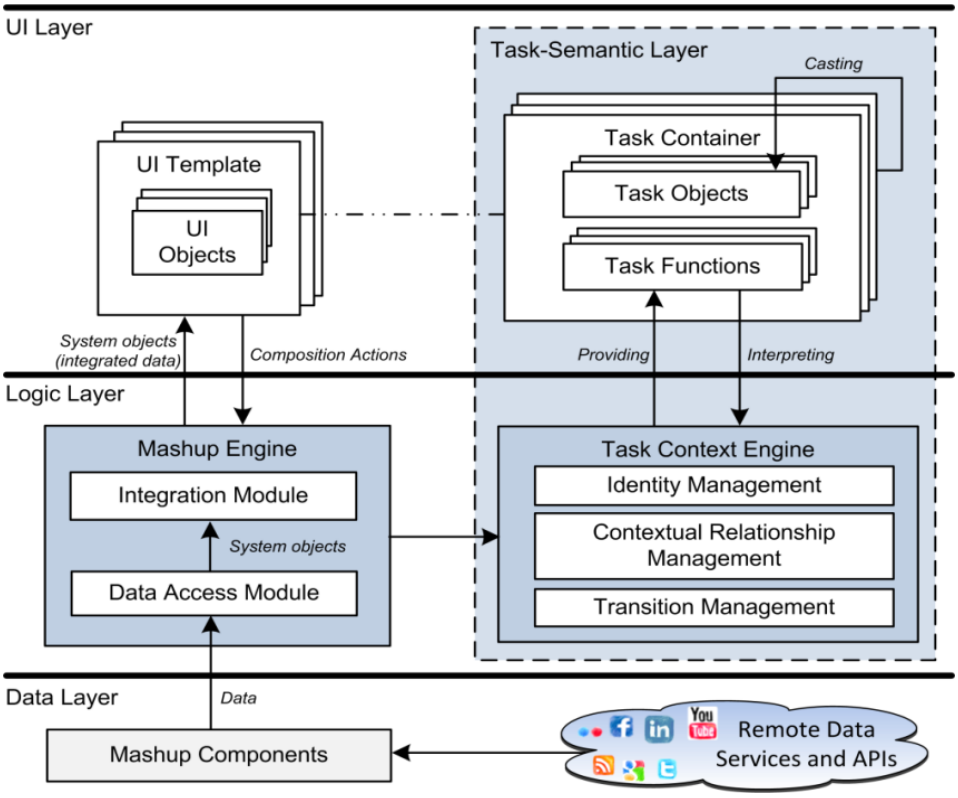

Figure 3.10: Overall organization of the framework supporting the interaction with mashups enhanced according to TUX principles

Another challenge is to deal with the need of users to endow objects with meanings that depend on the task they choose to accomplish. From the system perspective, a contextual relationship management module (see Figure 3.10) allows task objects to be augmented by users with subjective meanings and functions that relate to the task semantics of the selected containers where the interaction with the objects takes place. More specifically, this is handled by the generic function of casting, which implies that task objects are exposed to the aforementioned container-specific task semantics.

In any concrete scenario, users may interact with task objects in a sequence which spans multiple containers, along spontaneously defined trajectories that however have to keep track of the sequence of the various task semantics a given set of task objects was subjected to. The overall process can thus be considered a kind of “sequential casting” that in the framework of Figure 3.10 is managed by the transition management module.

Chapter 4. The EFESTO Platform

4.1 Introduction and Motivation

Mashup technology claim to be oriented to end users revealed unrealistic because the adopted composition languages are not actually suitable for end users [65, 84]. Recent studies found that, although some prominent platforms (e.g. Yahoo!Pipes) simplify the mashup development, they are still difficult to use by non-technical users [26, 65, 66, 91]. This might be due to the fact that the research on mashups has primarily focused on technologies and standards, with little attention on easing the mashup creation process, which often involves the manual programming of service integration. Mashup makers provide graphical user interfaces for combining mashup services [33, 39, 87]. With respect to manual programming, such tools make easier mashup composition tasks. However, to some extent they still require an understanding of the integration logic (e.g. data flow, parameter coupling, etc.).

With the intent of overcoming the limitations identified in literature, we created EFESTO (EFesto End uSer composition plaTfOrm), a platform for the End-User Development of mashups. Efesto was a god of the Greek mythology, who realized magnificent magic arms for other Greek gods and heroes. Analogously, the EFESTO platform aims to put in the hands of the end users powerful tools to accomplish their tasks. Our platform, in fact, is characterized by a paradigm for the exploration and composition of heterogeneous data sources that tries to accommodate the end-user mental model for a lightweight data integration within workspaces. As described in Section 4.3, the paradigm was designed taking into account the results of some elicitation studies performed to identify the service composition end-user mental model. It was also validated during two field studies in specific application domains, namely Cultural Heritage [9] (Section 4.6.1) and Technology Enhanced Learning [5] (Section 4.6.2) and, finally, in a utilization study (Section 4.6.3). These studies also highlighted new (unexpected) requirements, most of them implemented in the current version of EFESTO.

4.2 A Customization Environment for EFESTO

According to the meta-design model described in Section 3.2, an environment for customizing EFESTO to a specific domain has been developed. This customization is performed by means of different activities like services (also called *Data Component* in the model depicted in Section 3.3) registration, service attributes

renaming, service attributes selection, development of visual templates and/or task containers.

Service registration is performed by specifying basic properties to invoke the service. For example, in case of RESTful services, at registration time the service URI and the value of some search keys need to be specified. Service registration is facilitated by visual forms that guide the user to insert the data needed, so that, even if it is usually performed by technology experts, domain experts or end users themselves could do it with some guidance or after some training. The service registration produces a service descriptor (see Section 4.5).

During service registration, the customization environment permits also to rename the data attributes, which very often have not self-explanatory names. Moreover, since some of the many service attributes are useless for the user needs in a certain domain, only a sub-set of attributes are made available.

Another important activity in the customization environment is the development of visual templates (also called UI Templates in the model described in Section 3.3). Thanks to the visual templates, in our mashup tool the user can visualize the service raw data and integrate data gathered from different data components, by means of two integration operations: join and union. Further details about these integrations are described in Sections 4.4 and 4.5, where the EFESTO implementation and architecture are described.

In the customization environment containers to support sense-making tasks with the mashup tool can be developed. As described in Section 3.4, containers provide functions to process data; such functions are strictly related to the current context, as informed by the task actually performed by the users.

4.3 Design of a new composition paradigm for EFESTO: an elicitation study

Very few studies are reported in literature on understanding end-user requirements for accessing and integrating different services. The most significant study is reported in [84]: three separate focus groups were organized with the purpose of comparing three design alternatives of service-based systems enabling end-user composition. For the design of EFESTO, in our study we approached the problem differently. We purposely did not ask end users to assess pre-defined composition paradigms for not introducing biases. Rather, we tried to elicit through discussion their mental model and attitude on accessing and composing services. Through in-

terviews and a focus group, participants discussed on how to compose data and services. The identified requirements were then used in three design workshops to model a composition process and to create some preliminary prototypes. In the following sub-sections, we will illustrate the study, outline the results and shortly describe the preliminary prototypes.

4.3.1 Method

The study was based on interviews and focus groups, in which participants discussed about composing data and services. It was decided that no prototype should be shown, in order to avoid influencing participants. An HCI researcher conducting the interview or the focus group would make an introduction to explain what is a service and the motivations to compose services and/or data coming from different sources. Starting from the data mashup operations we wanted to accomplish through EFESTO, the discussion was focused on three main tasks: 1) *union* of data items taken from different sources, 2) *join* of data items returned by a source with data items of other sources, 3) *visualization change* of data items.

Questions for both interviews and focus group were defined in order to collect information from users about these three important tasks. Questions related to tasks 1 and 2 were formulated in two steps: a) illustrating, through a simple scenario that users could easily understand, how data from different sources are actually combined using a web browser, and b) asking the question on how the interviewee would like to use an application allowing him/her to perform such activities. For example, to explain the *join* task, the following scenario was illustrated (interviewees said that they knew SongKick, thus this service was used in the scenario): “A user (who is keen on music) daily uses SongKick to know the concerts that are held in his/her geographic area. From the browser, he/she accesses SongKick and looks at the results. The user does not know an artist who is giving a concert next week, thus he/she opens another website or service, like Wikipedia, to search for the artist discography”. A new Wikipedia search has to be done for every artist. Then, it is shown how this is currently performed and, eventually, it is asked: “How would you like to interact with an application that allows you to combine data from more than one source at the same time, in order to avoid jumping among different sites?”. Finally, in order to investigate the task 3, i.e. visualization change, the interviewer/moderator shows the interface of a popular web service (YouTube, SongKick, etc.) and askes the participant if he/she would like

to see the obtained results with a different visualization and how this task should be performed.

Nine single interviews and a moderated focus group were conducted. We took notes of users' comments, opinions and discussions and collected all the sketches created during the study. Based on the collected information, at the end of users' requirement gathering the interaction process has been identified during a design workshop. In two successive design workshops, user interface prototypes of the composition platform were created and refined.

4.3.2 Participants and procedure

9 people (5 female) aged 19-60 years old were interviewed. The interviewees had different cultural background. They included: a high-school student, two university students in different disciplines, a telecommunication technician, a dance teacher, two lawyers, an archaeologist, a business consultant.

The focus group involved 5 people (1 female) aged 22-25 years old; three of them were graduated in Economics, two were students in Engineering. All participants had low-medium knowledge of computer use. All were familiar with Internet but they had no experience in programming.

The interviews and focus group followed the same procedure. Initially, a 10-minute presentation to introduce the interview goal has been given: the platform scope and the meaning of a web service were illustrated by showing examples using the Web. Then, the discussion started by asking the participants questions related to the three tasks illustrated before. Participants answered by providing comments and, in the meanwhile, sketching interface elements and layout, showing how each one would like to carry out each task, e.g. through drag and drop actions, clicking on the mouse right button, selecting an icon, etc.

4.3.3 Results and discussion

The participants commented about platform benefits in their own daily life with a lot of enthusiasm. For example, a dance teacher said she frequently prepares choreographies and she happens to look for videos on YouTube and Vimeo. She clearly said that the platform would simplify the search, allowing her not to jump from a web page to another.

All participants said that they would like to use very simple and familiar interaction mechanisms, e.g., clicks of buttons and icons, menu item selection, drag-and-drop. Indeed, they specified these mechanisms in their sketches. They all indicated that the results coming from each source would be shown in a resource box in their own workspace.

On the basis of the participants' answers and sketches, two proposals actually emerged for the union of data sets taken from different sources. 10 out of 14 participants said that they would like to perform the union through a button, labelled with "+" or "Add" and included in the title bar of the resource box to be extended. This button opens a popup window proposing a wizard procedure that guides the user through the steps for combining two sources. The 4 remaining participants proposed that the platform interface could provide a search box in the title bar of the source to be extended, in which the user types a keyword for the category of content the new source should provide, e.g., video, audio, text. A drop-down menu is visualized listing the name and, possibly, an icon of the retrieved sources, e.g., YouTube and others. The user drags the source of interest from the ones listed and drops it into the box of the source to be combined.

Three alternatives were identified from participants' answers and sketches for the join of two or more data sources. The user started from a resource box showing data items coming from a specific source, e.g., Facebook. The first alternative, proposed by 6 participants, was to provide a button near each field of a data item, whose click activates another wizard procedure. 4 out of 14 participants proposed an alternative, in which the user positions the mouse cursor on the data item he/she wants to extend. The user clicks on the mouse right button to open a context menu showing a list of target sources from which he/she selects the source of interest. The third alternative was suggested by 4 participants: the platform shows a search box to search for the new source to be joined (a mechanism similar to the one used for the union operation). The user then drops one selected source in the field of a specific data item.

Finally, all participants expressed the desire to select the visualization of the returned items through a button opening a popup window where alternative visualizations are shown, so that the user can select the preferred one.

The study results showed that users like to be guided step by step in the composition of data sources. Participants indeed suggested a wizard procedure for performing both union and join of data. This is in line with the study reported in [84], where participants preferred the system-assisted composition approach over

the other two, which were based on the definition of control flow and data flow respectively. The wizard, indeed, hides complex technical details and guides the users in selecting data sources and in combining or linking data items. On the other hand, mechanisms strictly based on parameter coupling, as it is for data and control flow approaches, are the "programmer's way of building software artifacts", which does not necessarily match the end users' mental models ([65, 91]).

At the end of user-requirement gathering, during a first design workshop the team identified the interaction process. In the two successive design workshops, prototypes of the enabling user interface were created and refined, also considering the results of both heuristic evaluations and tests with two end users. These prototypes are described in the next section.

4.4 Interacting with EFESTO: an example

Our mashup tool is the result of an iterative process of design and evaluation phases grounded on the meta-design model presented in Section 3.2 and started with the design study for eliciting the composition paradigm (described in previous section), and then refined with the integration of TUX principles (see Section 3.4) and the polymorphic data source (see Chapter 5.) to satisfy the requirements emerged during our field studies (see Section 4.6).

The EFESTO user interface has been implemented by using Primefaces, an open source User Interface (UI) component library for JavaServer Faces (JSF) based applications. It is deployed on a remote Apache Web server and a Mysql database is used for the user account management.

With respect to the classification framework reported in Section 2.2.2, EFESTO is a WYSIWYG mashup tool since it blends the design and execution phases into the same environment. For this reason, we called *workspace* the space in which the user creates and use the mashup.

To illustrate the main features of EFESTO, a usage scenario is now introduced. Let us consider an end user, Michael, who is going to organize his summer holidays. Michael has not yet decided where to go between London and Madrid but, regardless the destination, he would like to attend a concert during his holidays. For this reason, Michael uses EFESTO to retrieve and integrate various information (i.e., to create mashups) about music events. Michael starts looking for pertinent services among those registered in the platform. A wizard procedure guides him to make a selection from a popup window where services are classi-

fied by category (e.g., videos, photos, music, social). Michael selects SongKick, a service that provides information on music events given an artist name. He also selects a map UI template for displaying the retrieved information. The aim of Michael's activities in the EFESTO workspace is indeed to create some widgets, called UI components in our model illustrated in Section 3.3, that visually render, in a chosen format, data extracted from selected data sources. As SongKick data are geo-localized, Michael decides to visualize the retrieved data on a map.

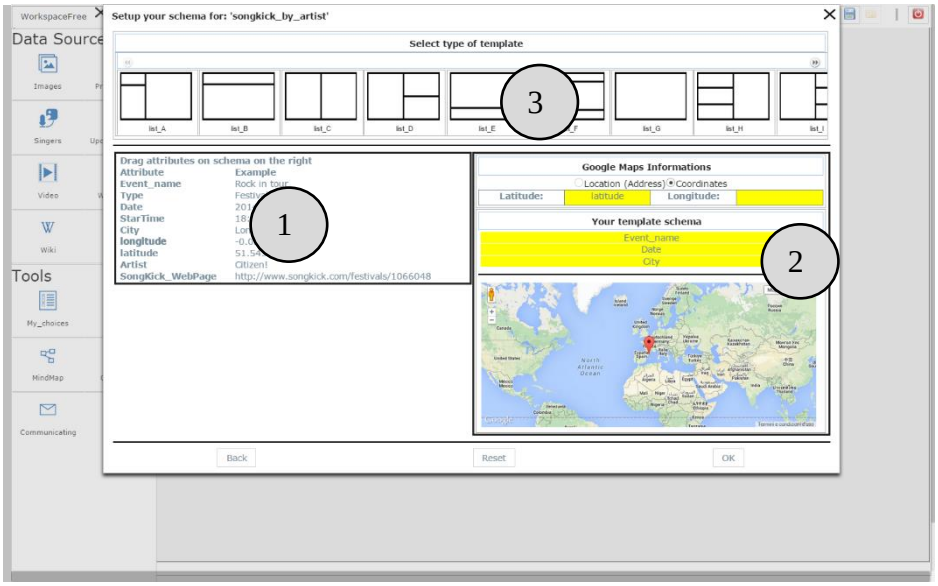

Figure 4.1: Mapping between the SongKick data attributes and UI template fields

As shown in Figure 4.1 (circle #1), the SongKick data attributes are visualized in a panel on the left. To make the attributes understandable by the user, the system also shows some example values. First, Michael drags & drops the latitude and longitude SongKick attributes into the related fields in the map UI template (Figure 4.1, circle #2). Then he chooses a table UI template with three items in column (Figure 4.1, circle #3) for visualizing, when required, some additional details about a musical event. He selects and drops the desired attributes in the fields of the table template (highlighted in yellow in Figure 4.1, circle #2).

After performing the mapping phase, Michael saves the mashup. Figure 4.2 reports an example of the created mashup, which is immediately executed in the Web browser. By typing "Vasco Rossi" in the search box, the forthcoming events of this singer are visualized as pins on the map.

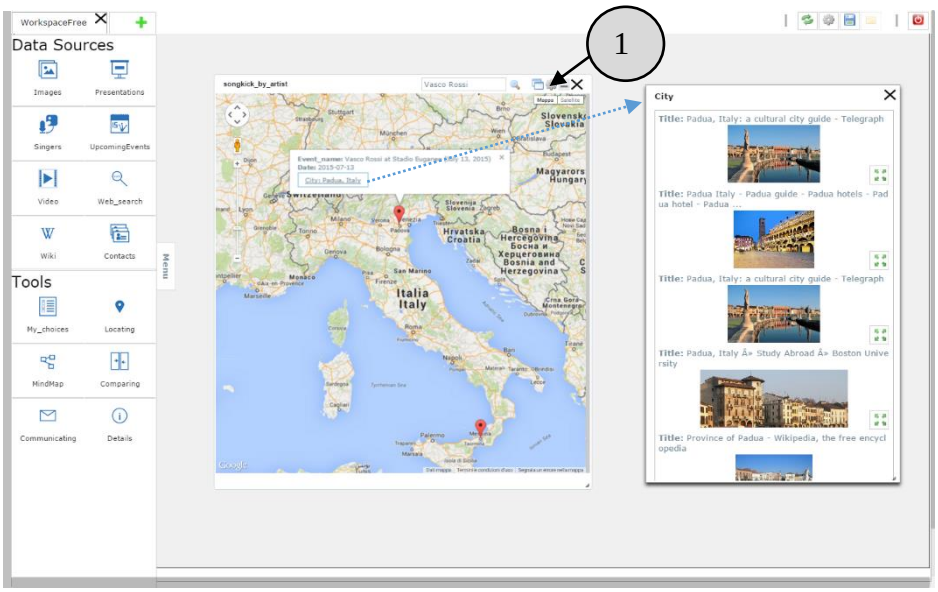

Figure 4.2: SongKick data source visualized as a map and joined with Google Images to show city pictures related to each SongKick event

Michael can also integrate data coming from different services through union and join operations (also called merge in other mashup tools [24]) that he visually expresses through drag&drop actions operated on the running mashup. For example, to enrich the dataset of events retrieved by SongKick, Michael integrates SongKick with Last.fm, thus exploiting the union operation. In particular, he acts directly on the SongKick UI component previously created by clicking on the gearwheel icon in the toolbar (pointed by the circle #1 in Figure 4.2) and choosing the "Add results from new source" menu item. A wizard procedure now guides Michael in choosing a new service and in performing a new mapping between the Last.fm attributes and the UI template already used when SongKick was created. The newly created dataset is shown in the same fashion as reported in Figure 4.2 but now, when queried with an artist name, the widget visualizes results gathered both from the SongKick and Last.fm services.

Another data integration operation available in EFESTO is the join of different datasets. For example, since SongKick does not provide images of the location where concerts are held, Michael joins the SongKick city attribute with Google Images; the city name now becomes the keyword for extracting from Google Images a sequence of related pictures. To perform this operation, Michael clicks on the component gearwheel icon and choses the "Extend results with details" menu item. A new wizard procedure guides him while choosing the service attribute to be extended (City in this example), the new data source (Google Images) and how to visualize the Google Images results. From now on, as shown in the right-hand

side of Figure 4.2, when clicking on the city name in the map info window, another pop-up visualizes the Google Images pictures related to the selected city.

Another operation available in EFESTO is the change of visualization for a given UI component. Michael, in fact, during the interaction with SongKick, decides to switch from the map UI template to the list UI template (see the result in Figure 4.3, circle #1). To perform this action, he clicks on the gearwheel icon in the SongKick toolbar and choses the *Change visualization* menu item. A visual procedure allows Michael to choose a UI template (a list in this case), and drag&drop the SongKick attributes onto the UI template, as already performed during the SongKick creation.

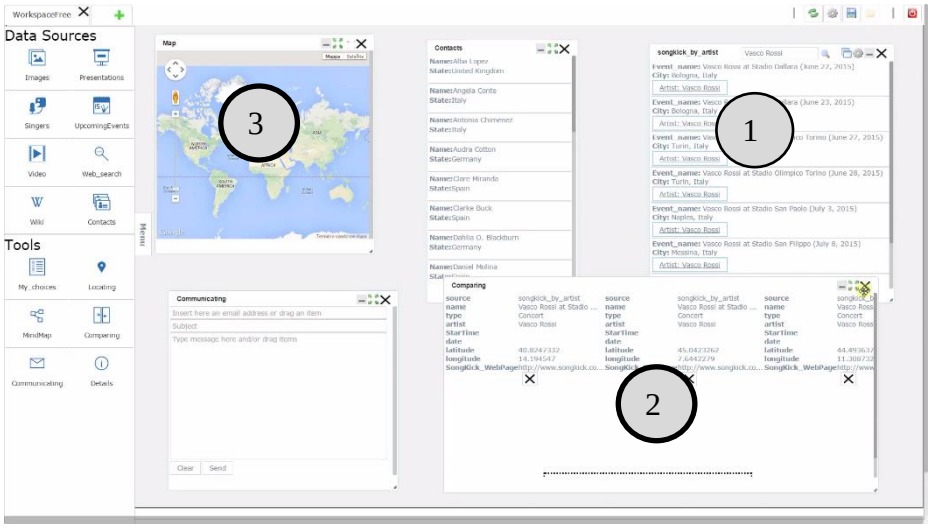

Figure 4.3: Use of some tools available in EFESTO to manipulate SongKick data

Until now, Michael has aggregated and composed information according to a paradigm that is similar for some aspects to the ones provided by other mashup platforms [34]. Our field studies (see Section 4.6), however, revealed that mashups generally lack data manipulation functions that can be instead useful to support common tasks [5, 9] and can empower the users to play a more active role than just consuming the finally visualized information. EFESTO was thus extended with a set of tools that, by exploiting functions local to the platform or exposed by remote APIs, provide the possibility to “act” on the extracted contents, for example to collect&save favorites, to compare items, to plot data items on a map, to inspect full content details, or to arrange items in a mind map to highlight relationships [12]. Coming back to our scenario, as shown in Figure 4.3, Michael adds some tools into his workspace, each of them devoted to a particular task. For example, each time Michael drags a SongKick event into the Map tool (Figure 4.3, circle #3), this item is automatically ‘translated’ as pin on the map. Another ex-

ample is the Comparing tool (Figure 4.3, circle #2) that assists the user in comparing items retrieved by one or more services (SongKick events in Figure 4.3). In general, item transitions across different tools determine different organizations and visualizations of data and progressively enable different functions (see Section 3.4).

4.5 EFESTO architecture

Figure 4.4 illustrates the overall organization of EFESTO. The platform supports the composition of heterogeneous components (data, UI and logic components) by means of an orchestration logic that enables extracting and integrating data and operations provided by different data components, mainly to create the so-called UI components. A UI synchronization logic also allows one to synchronize at the presentation layer the behavior of different UI components. This synchronization is based on an event-driven paradigm that couples events generated by source components to operation enacted in target components. The platform thus generates hybrid mashups that integrate data and orchestrate functions, and provides structured and coordinated visualizations of the integrated data set and functions.

EFESTO is strongly characterized by its interaction layer and, in particular, by its visual language that allows the users to create “live” mashups without writing a line of code. The adoption of a visual notation and the liveness of the mashups under construction demand for the definition of an execution logic that is distributed between the platform front-end and back-end and is in charge of interpreting the user composition actions and putting them in action immediately.

Another relevant feature is the capability of generating models (*Workspace Descriptors* and *UI Component Descriptors*), in a model-driven engineering (MDE) fashion. Models, expressed according to a Domain-Specific Language [9, 24], specify the user composition choices and drive the instantiation of the mashup running code. The MDE paradigm thus enables the deployment of a same mashup on multiple devices, as native execution engines can interpret the same generated models on different target devices. In order to support this execution paradigm, service descriptors are also needed to provide an adequate abstraction layer for invoking and querying services. The rest of this section will illustrate the mechanisms through which different modules, distributed along different layers, interoperate to give life to the EFESTO composition and execution paradigm.

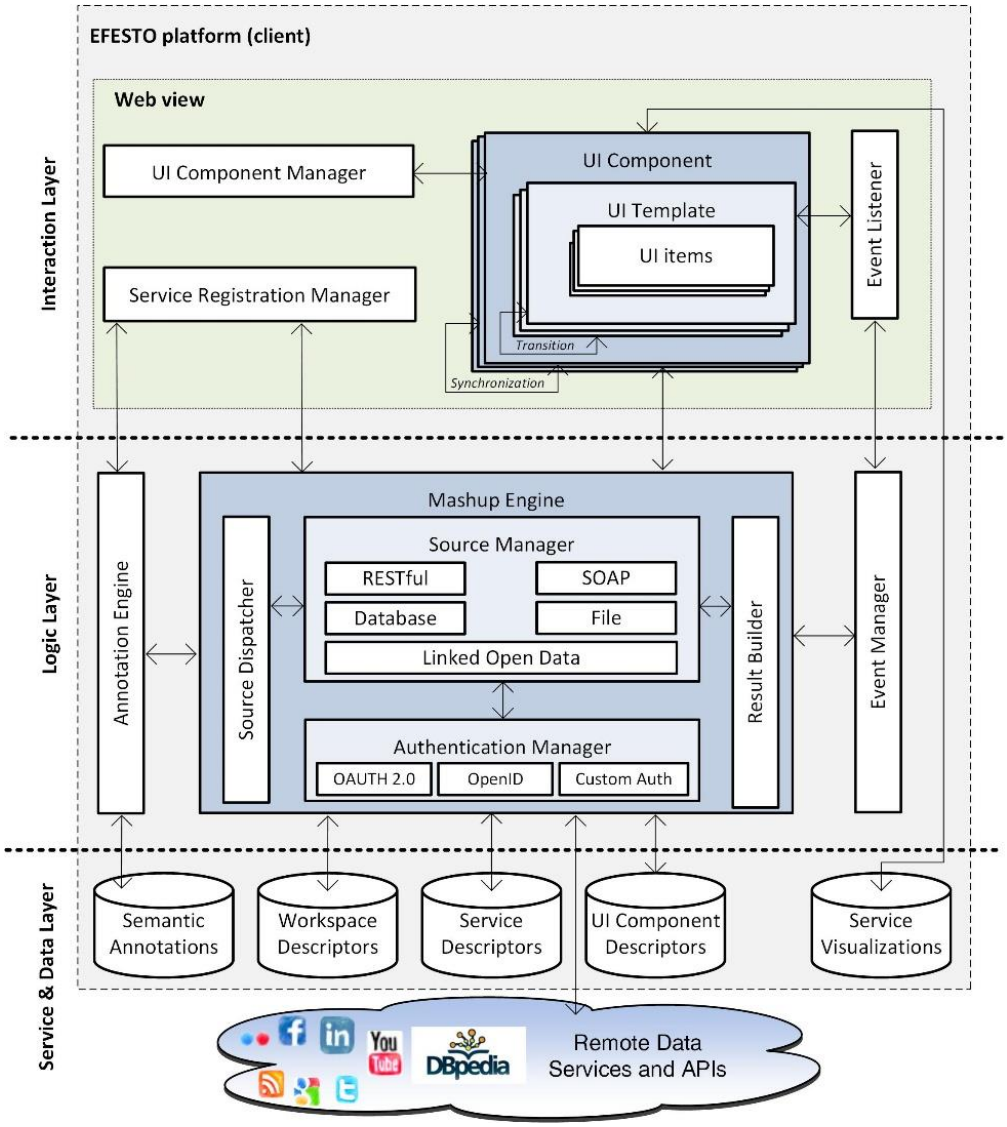

Figure 4.4: The EFESTO Three-layers architecture

4.5.1 Interaction Layer

In EFESTO, the *Interaction Layer* provides a kind of key metaphor determining the mashup logic and the overall system behaviour. Operations for mashup composition are indeed expressed by the users through direct manipulation actions on UI elements in charge of rendering data. According to a "programming-by example" paradigm, user actions operated on sample data items extracted from data sources are interpreted as models of queries to be executed on entire data sets and of the orchestration logic to be applied on the involved services. For instance, users connect some UI elements that display items retrieved from two different data sources to express a data flow for merging the two sources; or they move into an existing UI component some data attributes taken from a different service to de-

fine a union with this service. In other words, while acting directly on sample data objects, users program service composition to obtain new data sets, functions and visualizations.

This paradigm that, as demonstrated in some user studies [10, 65], is an essential prerequisite to foster EUD of mashups, is made possible by some front-end modules. As represented in Figure 4.4, the *Interaction Layer* consists of a Web application that represents a view on the model governing the logic for mashup composition and execution. A Web mashup in EFESTO is a set of UI components, each one providing a view on one or more data sources. The construction of such data views and their visualizations are managed by the *UI Component Manager*, a front-end module that instantiates each UI component based on the data sets built by the mashup engine. The logic of the UI Component Manager is determined by *UI Templates*. UI Templates are cornerstone elements in EFESTO, both for the way the users perceive the mechanisms for building UI components, and for the data integration logic behind the construction of the components data sets. Indeed, on the one hand, UI Templates provide the users with a schematic representation of how data extracted from services will be organized (i.e., aggregated and visualized) within each single UI component [9, 24]. On the other hand, at the Logic Layer UI templates then provide *data integration schemas*, as they determine how the mashup engine has to query the involved data sources and integrate the resulting data. Indeed, by associating selected service attributes to UI template elements, the composer defines a projection of the only attributes of interest. In addition, if the attributes associated to a single UI template element are selected from multiple services, then the structure of the UI template determines a *global integration schema* mapping the attributes of single services into an integrated data set. These actions captured at the interaction level are then translated into the specification, within a UI component descriptor, of service queries and data fusion procedures used by the mashup engine to build the integrated data sets [24].

As represented in Figure 4.4, each UI component displays a set of *UI items*, i.e., data elements rendered according to the layout provided by the UI template. UI items are the atomic elements composition actions can be applied to. Starting from a UI item, the users can expand the mashup data set by defining data integration operations (union and join) with data sets of additional services. The selection of a UI item can provide an entry point for the exploration in the LOD. The user can also achieve coordinated visualizations of the UI Components by *synchronizing* the event of selecting a UI item in a component with the activation of opera-

tions that can change the status of other components (e.g., to achieve a different data set filtering or a new visualization).

Given a UI component, *transitions* among different UI templates are possible to achieve different data organizations (e.g., from a table highlighting detailed properties of each single data instance to a mind map highlighting the relationship among different instances) and visualizations (e.g., from a list of addresses to a map based representation of the same data). Transitions, however, imply the need of modelling the structure of the data items originally extracted from data sources, to be able to trace and identify the transformations needed when moving the items across different visualizations. For this reason, each service, when registered, is associated with a set of possible *service visualizations*, i.e., the specification of UI templates families (i.e., lists, maps, charts, graphs) that can be properly used to render the service data. The mapping between the service data attributes and specific UI items in charge of attributes rendering is also defined.

The live programming paradigm, which allows the users to see immediately the effect of their actions on the mashup under construction, is achieved by means of *Event Listeners* that are able to catch the events generated by the user actions (e.g., the drag of a service attribute to a field of a UI template) and send them to an *Event Manager*. This module of the Mashup Engine, located in the Logic Layer, is in charge of translating events into the proper invocation of services whose effect is the refresh of the status of the mashup and of its UI components, depending on the captured events.

4.5.2 Logic Layer

The Logic Layer provides modules and mechanisms that translate the user composition actions operated at the Interaction Layer into the mashup executing logic. In this Section, different modules are described supposing that they are deployed separately from the Interaction Layer modules, i.e., on a back-end server. However, the Logic Layer can be distributed between the client and the server or, at the other extreme, located only at the client-side if the execution context requires a single-user, lightweight deployment. Server-side execution offers the advantage of managing a long lasting instantiation logic with the additional possibility of supporting multi-user mashups, collaborative composition paradigms, and the distributed execution of interactive workspaces.

The Mashup Engine

The *Mashup Engine* is invoked by the UI Component Manager each time an event, requiring the retrieval of new data or the invocation of service operations, is generated at the interaction layer. For instance, when the user specifies a search key to filter a component data set, the typed key and the component identifier are passed to the Mashup Engine. The Mashup Engine retrieves from a dedicated repository the XML-based *UI Component Descriptor*, and inspects it to identify all the services used in the mashup. Figure 4.5 illustrates an example of UI component descriptor where SongKick is joined with YouTube. Based on this specification, the Mashup Engine retrieves from the Service Descriptor repository all the XML descriptors associated with the services involved in the mashup (SongKick and YouTube in Figure 4.5). Each service descriptor is sent to the *Source Dispatcher* that, depending on the specified service type, invokes specific adapters to retrieve the data. In fact, our platform can manage different types of data sources, like RESTful and SOAP services, databases, files (e.g., csv, excel) and Linked Open Data. If a new type of data source needs to be registered in the platform, a new adapter has to be developed. Depending on the nature of the data source, the Source Dispatcher instantiates an adapter available in the *Source Manager* package that implements the logic for querying the specific type of data source. Moreover, if a data source demands for an authentication, the *Authentication Manager* provides for different classes implementing different types of authentication, like *OAUTH 2.0*, *OpenID* and *Custom Authentications*.

```
<?xml version="1.0" encoding="UTF-8"?>
<composition join="true" union="false">
  <base_service name="songkick_by_artist" hyperlink="false">
    <attribute name="Event_name" path="displayName">displayName</attribute>
    <attribute name="Type" path="type">type</attribute>
    <attribute name="Date" path="start.date">start.date</attribute>
    <attribute name="StarTime" path="start.time">start.time</attribute>
    <attribute name="City" path="location.city">location.city</attribute>
    <attribute name="longitude" path="location.lng">location.lng</attribute>
    <attribute name="latitude" path="location.lat">location.lat</attribute>
    <attribute name="Artist" path="performance[0].artist.displayName">performance[0].artist.displayName</attribute>
    <attribute name="SongKick_WebPage" path="uri">uri</attribute>
  </base_service>
  <unions>
  </unions>
  <joins>
    <join type="composition">
      <service name="youtube" />
      <input>Artist</input>
      <extendedAttributes>
        <attribute name="Title" path="snippet.title">snippet.title</attribute>
        <attribute name="Video" path="id.videoId">id.videoId</attribute>
      </extendedAttributes>
    </join>
  </joins>
</composition>
```

Figure 4.5: XML *UI Component* descriptor: the SongKick service is joined with YouTube through the Artist attribute

After querying each service as modelled in the *UI Component Descriptor*, the *Result Builder* creates the final data set, codified in JSON, and sends it back to the UI component manager. Figure 4.6 represents an example of JSON array produced by querying the mashup shown in Figure 4.5. Finally, the UI Component Manager builds the UI view to render the JSON data according to the layout of the component UI template.

The Event Manager

Another important module in the Logic Layer is the *Event Manger*. It is in charge of translating any composition action into proper descriptors, and to enact immediately service invocations to achieve the corresponding behaviour in the mashup under construction. When the users operate on a mashup the visual actions are caught by the *Event Listener* at the Interaction Layer and sent to the Event Manger. For example, at the beginning of our reference scenario, Michael creates the SongKick UI component by means of a wizard procedure that guides him to choose the data source (SongKick) and the UI template (Map), and to associate through drag&drop actions the SongKick attributes to the UI template fields. When Michael saves the SongKick mashup, two descriptors are created. The first one is similar to the one reported in Figure 4.5 (except for the <*joins*> tag that does not have any children when SongKick is created). When users expand the data source by joining and unifying it with other sources, the <*joins*> and <*unions*> tags are enriched with specific children.

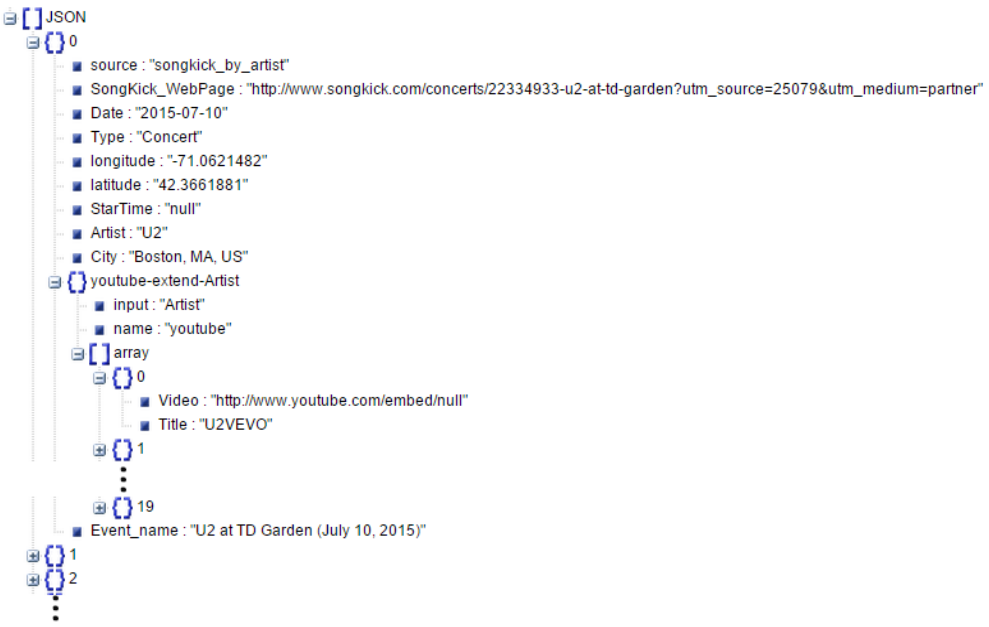

Figure 4.6: JSON array produced by the Mashup Engine invoked on the *UI Component* descriptor shown in Figure 4.5 with the *"U2"* query

The second XML file then defines the mapping between the data attributes included in the mashup (as described in the first descriptor) and the chosen UI template (whose structure is in turn described in an XML file stored in the Service Visualizations repository).

The Annotation Engine and the Polymorphic Data Source

During our field studies, we noticed that very often, during the process of exploring information, end users were forced to leave the platform to perform their tasks through traditional search engines. To overcome this limitation and better satisfy the end users' information needs, a new polymorphic data source built upon the LOD cloud, and in particular exploiting the DBpedia knowledge base has been introduced.

In order to create the polymorphic data source, a mapping step is required between all the data sources registered in the platform and the DBpedia ontology classes. The main goal of this mapping is to annotate the attributes of each service by using a DBpedia class that is semantically similar to the attribute. In fact, each time the EFESTO administrator registers a new service through the administration panel, the *Service Registration Manager* (a module of the Web front-end) asks the administrator to type some example queries (at most a dozen) to automatically annotate the service attributes. The service descriptor, together with the provided example queries, is sent to the *Annotation Engine* that automatically generates the service attribute annotations [35], which are then stored in the *Semantic Annotation* repository. Further details about this data source, and in particular about motivation and algorithm, are described in Chapter 5. In this section, only the integration of this data source in EFESTO architecture is described.

Now let us come back to the Michael scenario and suppose that he wants to join the SongKick Artist attribute with DBpedia. After he decides to use DBpedia as extension data source, the Event Manager triggers the retrieval, by the source manager, of the XML file with the annotations associated with SongKick. The class used to annotate the artist attribute (*MusicalArtist* class) is then extracted from the DBpedia ontology. Afterwards, the wizard procedure shows to Michael all the MusicalArtist properties as attributes that he can choose to build the joined data source (see the highlighted box in Figure 4.7). After the drag&drop of a subset of properties into the UI template fields, Michael saves SongKick. Now on, the event of clicking on a specific artist name in the SongKick results triggers in EFESTO the retrieval of a specific instance of the DBpedia knowledge base and its visualization in the chosen UI template, according to the mapping previously

performed by the user. To better understand what happens behind the scene when an artist is clicked, let us suppose that Michael clicks on the *U2* label. First of all, the Mashup Engine, and in particular the *Linked Open Data* module, queries DBpedia with a SPARQL query like:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
select ?p ?o
where {dbpedia:U2 ?p ?o}
```

The query result is a DBpedia instance characterized by a set of properties, some of which have to be mapped in the chosen UI template (e.g., *genre*, *starting year of activity*, and *artist photo*). Sometimes, it could happen that the retrieved instance does not have a value for a specific property; in this case, this value is skipped in the UI template. Furthermore, when the Mashup Engine queries DBpedia, it could happen that different instances are associated with the same label. For example, if the previous query includes the *Ligabue* search key instead of *U2*, five instances are retrieved: Antonio Ligabue, an Italian painter; Giancarlo Ligabue, an Italian palaeontologist; Ilva Ligabue, an Italian operatic soprano; Ligabue, a TV drama; Luciano Ligabue, the Italian singer (our target). To identify the right instance (Luciano Ligabue), the system checks which one is a sub-class of the class used to annotate the artist attribute, namely *MusicalArtist* in the Michael's scenario. This example highlights the dual role of service attribute annotations, which are used *i)* during the mapping phase, to show the DBpedia class properties that the users can move into the UI Template fields (Figure 4.7) and *ii)* during the execution of a SPARQL query, to disambiguate multiple retrieved instances.

4.5.3 Service&Data Layer

Through the *Service&Data* layer, the EFESTO Web server exposes repositories of XML-based descriptors that enable the invocation of services to extract data.

The *Service Descriptors* provide abstract specifications on how to query each data source registered in the platform and how to read its results. The *Workspace Descriptors* then contain representations of the workspaces created by each user. For each workspace, a descriptor specifies the included UI components and possible UI synchronizations defined among them. The UI Component Descriptors then specify the services included into the components, the user-defined

queries to integrate the services data sets (see Figure 4.5), and the specification of the component UI template.

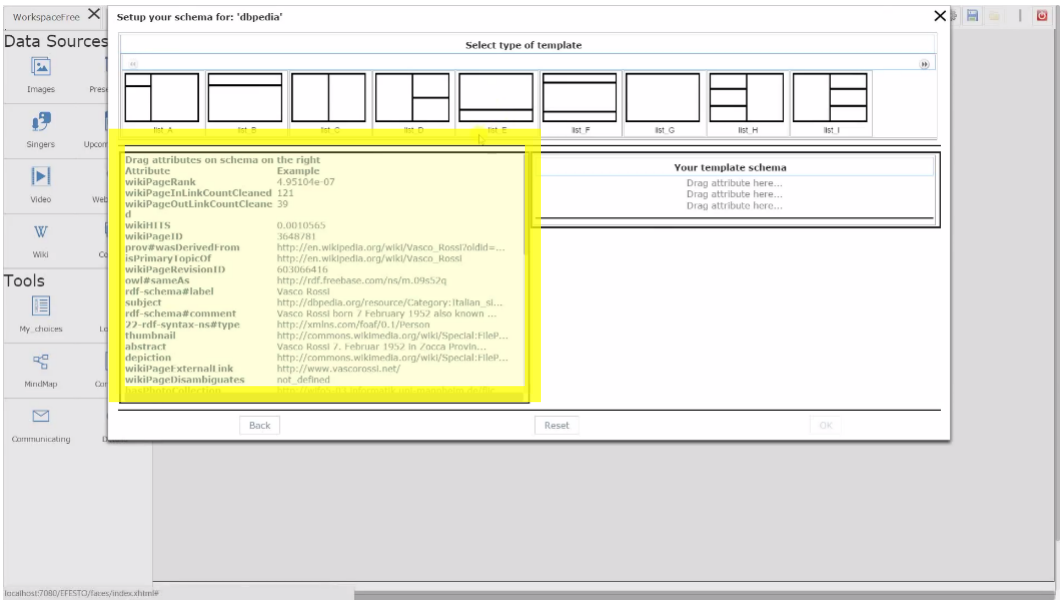

Figure 4.7: Mapping step between the DBpedia-based polymorphic data source properties and the list UI template

The Workspace and UI Component descriptors are associated to the user who creates them, and thus can be accessed depending on the users' access rights. Some "default" workspace descriptors are also available to any user; they provide the specification for pre-packaged workspaces related to specific topics or domains. In fact, users can compose their mashups starting from an empty workspace (like in Michael scenario) or choosing a thematic template filled with some ready-to-use UI Components that are relevant for particular domains/topics.

The *Semantic Annotations* repository stores the files used to describe the DBpedia classes associated to each service attribute. Finally, the *Service Visualizations* descriptors provide the abstract representations (in terms of offered UI elements) of the available UI templates.

The definition of the service descriptors and semantic annotation is a technical task that could be out of reach for non-programmers and, as such, could limit the introduction of new services within the platform by end users. To alleviate this problem, the definition of descriptors and annotations is facilitated by visual forms that only require inserting some values; then the XML specification is automatically generated by the system. Also, we envisage the adoption of our platform in meta-design scenarios, where other stakeholders (i.e., expert programmers

and domain experts) are supposed to configure the platform for its initial use by the end users.

In general, despite the difficulties that end users might encounter, the adoption of service descriptors and adapters enables a decoupling between the Mashup Engine and the external resources so that adding a new data source only requires defining a new descriptor; an adapter is also needed but only if the Source Manager does not already include one able to manage that type of data source.

A further aspect to be noted is that, although the service and mashup descriptors are codified using a custom XML grammar, the Mashup Engine is designed to work even with different grammars designed for service and mashup descriptions, like for example EMML (Enterprise Mashup Markup Language)[9] for which an open community already provided a large amount of descriptors. Any other service ecosystem, where services are homogeneously described, would work as well. However, to speed up the platform development and validation a custom XML grammar inspired to EMML has been proposed.

4.6 EFESTO Evaluation

In the next three subsections, three user studies performed to validate the proposed models, architecture and composition paradigm are reported. These studies, as well as other phases of this research, are the result of a collaboration with professors and researchers affiliated to other universities (Politecnico di Milano, La Sapienza University of Rome, University of Trento) [5, 9] and to SAP company [11]. For this reason, in the following, we will refer to the involvement of more than one researcher during the user studies.

4.6.1 Field study in the Cultural Heritage domain

A first field study was conducted in November 2012 at the archeological park of Egnathia to assess the use of EFESTO in a real setting. The study was intended as a formative evaluation to acquire insights from the use of the first version EFESTO in the field, highlighting the pros and cons and obtaining insights on the overall approach and on how to improve and extend the platform. The prototypes

[9] http://mdc.jackbe.com/prestodocs/v3.8/index.html

used in the study were implemented on desktop PCs, tablets and large interactive displays [6].

At that time, EFESTO only allowed to aggregate different UI components in a personal workspace, to bookmark content retrieved by different components and to show bookmarks on a visualization component like Google Maps. No join and union operations could be performed by end-users since these operations, at that time, were performed by developers and domain experts (park guides) during the customization of EFESTO to a specific domain (middle layer of our meta-design model see Section 3.2). After this field study, we decided to introduce in EFESTO more complex mashup operations (e.g., join and union) for non-programmers, to satisfy the emerging requirements.

Participants and design

The study involved two professional guides, named Achille (male) and Conny (female), and 28 visitors. Both guides formally agreed to participate in the study by signing an explicit consensus. Each guide accompanied a group of 14 visitors in the visit of the Egnathia park. The visitors were people who had booked a visit to the park. They were heterogeneous as regards age (from 21 to 50 years old, plus an 8-year old child), gender and cultural background. They were all Italian but one, a lady from Portugal who currently lives in a nearby city and speaks Italian very well. These visitors were randomly divided into two groups.

Procedure

The study took place on two days and consisted of two main sessions: (1) workspace composition (customization) and (2) park visit. The workspace composition session occurred on November 7th, 2012 in the guides' office. The two guides were given a 1-h demonstration of a desktop application, accessible through a PC, to be used to customize and compose the workspace (e.g., to register services, filter service attributes, chose visual templates to visualize service results, mashup services with join and union operations). After this, according to the co-discovery exploration technique [53], the two guides were invited to create together a workspace for visiting the archeological park of Monte Sannace, in the Apulia region. In this way, the guides had the possibility to become familiar with the application and, overall, to customize the platform to their domain, as required by our meta-design model (see Section 3.2). Then, they were asked to create their workspace to be used for the visit of the Egnathia archeological park. The guides individually created their workspace by positioning on an interactive map of the

park all the multimedia contents they would like to show to visitors. Moreover, during this phase, guides asked for non-available data sources that were registered in EFESTO by HCI experts. For example, they asked for data source to retrieve images that was created by a developer as union of Google Images and FlickR and then it was made available to the guides. At the end of the workspace creation session, the guides participated in a design workshop together with a platform designer and two HCI researchers, in order to discuss impressions, problems and possible modifications of the composition mechanisms and the overall system.

On November 17th, the park visit session was performed at the archeological park of Egnathia. This session consisted of two different phases: (1) the briefing phase at the beginning of the visit, in which the guide accessed his/ her workspace through a large multi-touch display (46-in.) placed at the entrance of the indoor park museum (Figure 4.8a) and (2) the tour phase, in which the guide accessed his/her workspace on a tablet (7-in.) during the tour through the remains in the park (Figure 4.8b).

Figure 4.8: The guides interacting with their workspace: (a) during the briefing phase using the multi-touch display and (b) during the tour using the tablet.

First of all, the visitors were informed that they were participating in a slightly different visit with respect to the traditional one, since multimedia materials available on different technological devices were going to be tested. Thus, some pictures would be taken but none would be published in which people's faces could be recognized. They all agreed to participate in the visit.

In the briefing phase, the guide interacted with his/her workspace on the multi-touch display to introduce visitors to the history of Egnathia and to what they were going to see in the park. After this, the tour phase began. During the tour, guides could use their workspace on the tablet to satisfy situational needs, e.g., to show visitors more details about the remains and to answer visitors' ques-

tions better by showing specific multimedia contents. In both phases, guides could search for new contents from services and, possibly, update their workspace by retrieving and bookmarking new content. The visit session lasted approximately one hour and half.

The two visits occurred in the morning. In the afternoon, another design workshop was carried out involving the same participants as the first one, focusing on the pros and cons of workspace use. Each group of visitors participated in a focus group at the end of the visit, where their impressions on the overall visit experience were discussed.

Data collection

In order to analyze the guides' experience in composing their workspace and using it, data were gathered through naturalistic observation of the guides during: (1) the workspace composition on the desktop PC, (2) the interaction with the created workspace running on the multi-touch display during the briefing phase, and (3) the interaction with the tablet during the tour phase. These data were complemented by the guides' comments gathered during the design workshops after the workspace composition and at the end of the Egnathia park visit.

In the workspace composition session, two HCI experts observed the two guides creating the workspace together: one took notes on paper and the other videotaped all interactions. At the end of the session, they participated in the design workshops together with the guides and the platform designer. The workshop was audio-taped.

Six HCI experts (three experts for each group) followed the park visit session, videotaping and taking notes of the main events. At the end of the visit, the three experts moderated the focus group with the visitors' group they had observed. The focus groups were audio-taped. Moreover, the two guides participated in a design workshop, with the same modality and participants of the previous workshop.

The set of notes collected by the experts in the two sessions was substantially extended by video- and audio analysis. Two researchers transcribed the videos and the audios and independently double-checked 65% of the material. If the inter-rater agreement was less than 70%, the researchers discussed the differences and reached an agreement. Final reliability was high (agreement over 90%).

Results of composition phase

In this phase, the guides were observed while composing their workspace for visiting the archeological park of Egnathia, using the desktop application. In general, the usability problems they experienced were not so serious to get them stuck; they were in fact able to continue the workspace composition and use without the help of the 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 materials they retrieved.

At the end of this phase, the two guides participated in the design workshop. As an 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 workspace they had created and they were confident that it would be a valuable support during the visit. Achille jokingly said to Conny: *When this system will be released, I'll call you the day before a visit to ask for suggestions about what to include in my application.*

Results of briefing phase

The briefing aimed at both introducing visitors to the history of Egnathia and providing some preliminary information about the park. The briefing time, during which the guides used the multi-touch display, lasted much longer than in traditional visits, where the briefing to introduce visitors to the archeological park is about 5 min at most. Conny used the multi-touch display for about one third of Achille's time (see Table 4.1). During his interaction with the multi-touch display, Achille experienced three interaction difficulties due to some technological limitations (Table 4.1): 1) a temporary loss of Internet connection; 2) he was not able to close the pop-up window by touching the "X" icon, which was located near the display border (our multi- touch device is not very sensitive along its borders) and; 3) in the few situations he had to use the virtual keyboard displayed on the screen, due to low precision of the device in correctly detecting the pressed key. Conny had only one problem during her interaction with the display related to the use of the virtual keyboard. However, both guides were able to autonomously manage such difficulties.

Table 4.1: Use time and interaction difficulties with the multi-touch display

Variable	***Achille***	***Conny***
Use time	28m 35s	11m 45s
Interaction difficulties	3	1

Both guides appeared quite relaxed in using the multi-touch display. They illustrated the multimedia contents they had previously inserted in their workspaces. They were able to search new content without difficulties related to the search functionality. Specifically, Table 4.2 shows the number of searches and workspace modifications Achille and Conny performed. Achille carried out 4 searches, and only 1 out of 4 was not successful because the retrieved contents did not satisfy his needs. In three cases, Achille believed the retrieved content should have been inserted in his workspace and thus he modified it. Conny performed only 1 search and she did not update her workspace.

Table 4.2: Number of performed searches and modifications of the workspace with the multi-touch display

	Achille	***Conny***
Searching new content	4	1
Modifying the workspace	3	0

It is worth noticing that, when the search for new content required more than 2 min, visitors appeared to be distracted and started chatting among themselves and looking around. Also, in the focus group, some visitors remarked that the position of the multi-touch display generated some problems since, when the guide was interacting with the display, he partially covered it and visitors had to move their heads or their bodies, since they were curious to see all the steps of the interaction.

During the briefing phase, all visitors appeared very interested in the contents illustrated by the guides on the multi-touch display: they asked their guide questions, commented on images among themselves and in general appeared engaged and stimulated by the material shown. This was confirmed in the focus group, in which visitors explicitly expressed their positive opinion about the briefing phase. Nobody complained about this longer phase; on the contrary, they all said: *It's worth it!*. They also said that they would have liked a debriefing phase at the end of the visit, i.e., a phase in which to deepen some topics and possibly look again at the multimedia resources on the multi-touch display, in order to comment with the guide those aspects that had captured their attention during the visit.

Results of 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 present the park remains better. Achille and Conny used tablet and panels in different ways (see Table 4.3). Conny was more prone to the use of such tools; in fact, she used the tablet nine times and the panels nine times. In total, she spent 7 min and 53 s commenting the contents available on the tablet, and 3 min and 58 s commenting images on the panels. Achille used such tools very little: he used the tablet once for 1 min and a panel once for 10 s. It is evident that both guides were stimulated to talk more about the contents on the tablet than those on the panels.

Table 4.3: Frequency and use time of tools

Tools	***Variable***	***Achille***	***Conny***
Tablet	Frequency	1	9
	Time	1m	7m 53s
Panels	Frequency	1	9
	Time	10s	3m 58s

Both guides performed searches through the workspace on the tablet. In 3 out of the 9 times in which Conny interacted with the tablet, she performed a search. Only one search did not provide results of her interest. The only time Achille used the tablet was to perform a search. Specifically, the Portuguese visitor said that she loved history and she had visited some Roman archeological parks in Portugal. Achille was very intrigued and started to make searches to understand similarities and differences between the Portuguese sites and Egnathia. From the retrieved images, similarities between the two archeological sites were evident.

During the tour phase, both Achille and Conny did not want to modify the workspace (see Table 4.4) since they thought that this would require time and, consequently, distract the visitors.

Table 4.4: Number of performed searches and modifications of the workspace with the tablet

	Achille	***Conny***
Searching new content	1	3
Modifying the workspace	0	0

In the design workshop after the park visit, the guides reported that, during the many visits they have performed in their career, very often visitors interrupt

them to integrate the guide's presentation with their own knowledge, e.g., history teachers report something they studied, archeologists mention something about a recent discovery in another site, etc. They said that, after the visit, they like to study and analyze more in depth the information given by such visitors, and therefore generally search on the Web, or request material from their colleagues by e-mail or by phone. This modus operandi allows them to enrich their knowledge in preparation for successive visits. The guides clearly remarked that the workspace would improve greatly the acquisition and storage of new knowledge.

Both guides also mentioned that, during the tour, searches requiring more than 2 min interrupted the narrative and distracted visitors. Even though they did not feel uncomfortable during this waiting time, they would have preferred to collect more material without the delay due to the Internet connection, i.e., from local repositories. Achille and Conny said that they would have liked to use the tablet more, since they appreciated its support in making useful material available. Conny explained that she had used her workspace so little during the tour because she had inserted in it many images that were available on the panels in the park. Thus, she had preferred to show such images on the panels since the tablet was too small for a group of 14 people.

The tablet size emerged as an issue also in the focus groups with visitors, who said that they preferred to look at images on the panels rather than flocking together around the guide to see them on the tablet. They also complained about the brightness of the tablet screen, compromised by external factors, such as sunlight.

Discussion

The objective of this study was to assess the value of the workspace, accessible from different devices, in enhancing visits to archeological parks. To this aim, we analyzed the experience of the two categories of actors involved in the visit: the guide and the visitors. The guide had a double role: (1) designer, i.e., s/he created her/his workspace and (2) end user, i.e., s/he used the workspace during the visit to illustrate the park remains better to visitors. The visitors participated in a visit which was enhanced by the availability of different types of multimedia materials and, thanks to the possibility given by the workspace to search new content, their curiosity might be better satisfied than in a traditional visit.

Composing the workspace with a desktop application did not create particular problems for the guides. They appreciated the support of the composition plat-

form 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 content made available by a third-party or on user-generated content. To limit this problem, more sensible services should be added to the platform; they can be further third-party services, if any responding to the user needs exist, but they can also be local and ad hoc created collections of contents, maintained by domain experts. Also, given that the services used for the study in the Egnathia park are Web 2.0 resources, the guides could publish online their own material (e.g., videos, pictures, Wikipedia pages) that can be easily accessed through the composition environment. This of course requires a more intensive use of the system by the guides, since they have to realize which material is missing and consequently enrich their public online collections.

Both guides and visitors appreciated very much the briefing phase with the support of the multi-touch display, which appeared to be very valuable in that phase of the visit. The display allows the guides to present much more multimedia materials related to park elements, which enrich their spoken presentation greatly. The visitors' satisfaction is confirmed by their request for a debriefing phase at the end of the visit. As pointed out in several studies, e.g., [8, 13], a debriefing phase would be very useful, since it provides the opportunity to deepen and elaborate the information received during the visit, in order to consolidate the acquired knowledge. During the debriefing phase, points of interest that, due to time constraints, were not possible to visit could be quickly illustrated. For example, in the specific case of Egnathia, the necropolis is far from the main city and often it is not visited. In a debriefing phase, the guide could present it by showing pictures or videos; later, people could visit it by themselves if they want to.

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 cm 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 cm), placing it on a platform at least 50 cm high, which the guide will get on. In this way, the display would be more easily visible to 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 workspace.

A negative aspect of the use of the workspace on the multi-touch display was the waiting time during 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 the workspace design environment is limited to the services registered 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. As already mentioned, the problem can be reduced by allowing end users to add further data sources to the workspace during its use and not only during the customization phase.

Before the study, we expected a larger use of the workspace on the tablet during the tour phase, since it could show images of monuments and other elements of interest, helping visitors to reconstruct the original appearance of such elements and figure out how life used to be in ancient times. Actually, Achille did not show any multimedia content and used the tablet only for one search of a new content. The video analysis revealed that, in a specific situation, Achille exclaimed: *It is a pity that I do not have a picture to show you!* However, he did not consider the possibility of using the tablet to search for the picture. Since in the design workshops he was clearly enthusiastic about the technological tools used, it seems that he would need more time to appropriate these tools. This also holds for Conny. She used the tablet more times but she inserted in her workspace primarily pictures that were also reproduced on the panels in the park rather than additional material that could complement what is already available. It was evident that visitors preferred to look at the images on the panels rather than on the small screen of the tablet, whose visibility is compromised by the sunlight. To overcome this problem, we implemented the possibility to visualize some contents of the guide's workspace on the visitors' smart phones.

Moreover, we also developed new solutions to allow the users to share the workspace and allow others to reuse it. We were in particular interested in investigating to what extent a collaboration paradigm would improve the usefulness of workspaces for supporting the cooperation among different stakeholders. The need for collaboration to co-create and share workspace s has emerged as a desirable feature in all the user-based evaluation sessions we have performed so far. Based on these new requirements, we defined some extensions of our composition platform to enable workspace annotation and workspace co-creation [4] and assessed them during a new field study described in the following sub-section.

4.6.2 Field study in the Technology-Enhanced Learning domain

Starting from the requirements emerged during the field study performed in Cultural Heritage domain and described in previous sub-section, a second field study was performed in the context of Technology-Enhanced Learning (TEL). Following some of the emerged requirements, we improved EFESTO with *i)* more expressive and powerful mashup mechanisms elicited during a user study (see Section 4.3), namely join and union operations, and *ii)* mechanisms to allow group of users to compose and use the same workspace synchronously and asynchronously.

This study allowed us to analyze the use of the platform in a situation in which students learn about a topic presented in class by their teacher, complementing the teacher's class by searching information on the Web. The retrieved information can also be communicated and shared with the teacher and the other students using interactive whiteboards, desktop PCs and personal devices (e.g., laptop, tablet and smartphone).

Participants and design

The study was carried out at the technical high school “Antonietta Cezzi De Castro” in Maglie, a city in Southern Italy. It was organized over 3 days, involving a class of 16 students (9 females, 19 year-old on average) and a teacher. All participant, teacher and students agreed to participate in the study by signing an explicit consensus.

Procedure

During the first day, the teacher composed his workspace relative to “Communication Networks”. Two days later, the teacher gave a lesson supported by the workspace visualized on an interactive whiteboard. At the end of the lesson, he divided the students into groups of 2–3; each group was assigned the task of creating an workspace about a specific Communication Networks sub-topic, e.g., protocols, packet switching, and latency period. After a brief individual training session, all the groups accessed the laboratory to carry out their assignments. Figure 4.11 shows a couple of students working with their workspace, to which they are adding widgets visualized through a list-based visual template, to retrieve and integrate contents from Google, Slideshare and YouTube. At the end of this session, we simulated the sharing of their workspace with the teacher by manually integrating their components into a unique workspace accessible by the teacher.

After 2 days, teacher and students met again for a class on Communication Networks; this time the class was supported by the integrated workspace running on the interactive whiteboard (see Figure 4.9). The discussion on the retrieved information lasted for an hour and a half. At the end, teacher and students had 20 min to fill in a short questionnaire inquiring about platform pros and cons they perceived.

A significant part of this field study was a design workshop that was conducted at the end of the third day, in order to better understand the need for collaborating with other people by means of workspaces. The design workshop aimed at engaging students and teacher in: (1) elicitation of positive and negative aspects of the overall interaction experience with the platform; (2) active participation in the design of new solutions, primarily stressing the envisioned possibilities of collaborative composition of an workspace. The latter objective was derived from the results of the field study in the Cultural Heritage domain indicating the willingness of professional guides to compose collaboratively the workspace to be used during a visit. Four groups were formed, each involving four students, one interaction designer, one platform developer and one HCI researcher. One group also included the teacher. Stimulated by the researcher, participants elaborated their ideas about interaction possibilities. Then, they were asked to sketch such ideas (see Figure 4.10).

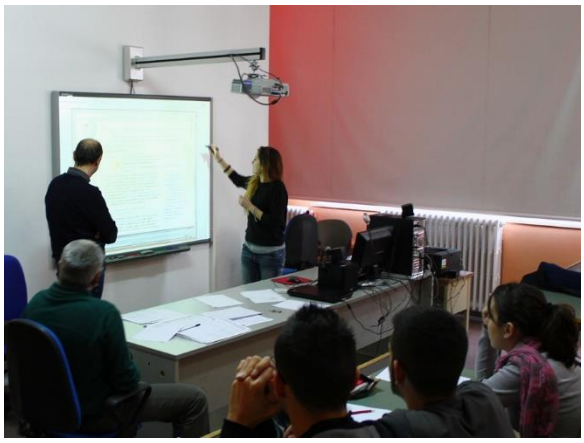

Figure 4.9: A student discussing about Communication Networks by using the integrated IW on the interactive whiteboard.

Figure 4.10: A group sketching interaction ideas during the design workshop.

The design workshop lasted for an hour and a half. At the end, a plenary session of 30 min was held and the more promising ideas were illustrated and discussed. They were instrumental for the design of both functionality and visual interface of the collaboration mechanisms that, as discussed in Section 3, we next implemented in our platform.

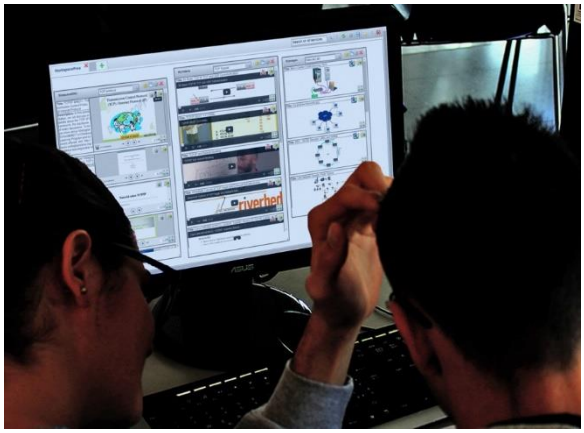

Figure 4.11: Two students working with their IW on a desktop PC.

Data collection

In order to analyse the participant experience during the workspace composition and use, different data have been collected. The first day, data about the teacher interaction were gathered through audio/video recording, observation and notes. Moreover, the teacher was asked to verbalize their thoughts and comments on his actions according to the think-aloud protocol. Lastly, a semi-structured interview was conducted at the end of the session.

At the second day, during the lesson, data about the teacher use of his workspace were gathered through audio/video recording, observation and notes. After the lesson, when the groups of students were asked to compose their workspaces

following the teacher indications, data were gathered through audio/video recording, screen-capture, observation, notes and a post-test questionnaire.

At the third day, during the lesson, data about the teacher use of his workspace were gathered through audio/video recording, observation and notes. After the lesson, during the design workshops, data were gathered through audio/video recording and notes. Moreover, also the sketches created by students have been collected.

Results and Discussion

The analysis of data collected during the three days revealed important results. In general, it turned out that both students and teacher wish more flexibility in organizing the interactive workspace, and the need emerged for visual containers in which retrieved content can be arranged and classified according to unforeseen needs. They stressed that the platform should be improved to support collaborative activities and contributed in the design of possible features and usage scenarios. The teacher proposed a "peer-learning" workspace, in which both teachers and students can share their contents, offer comments or create a discussion thread, and express their appreciation in a Facebook or YouTube style. The students envisaged the possibility of a distributed collaborative creation of a workspace, which could be asynchronous in case of a homework assignment or synchronous if carried out in class during a lesson.

4.6.3 A utilization study to evaluate TUX principles in EFESTO

The field studies performed during this research (Section 4.6) revealed some weaknesses of our approaches and, in general, of mashup tools [5], related to the manipulation of the information retrieved by the user-defined data sources. The integration of TUX principles in mashup tools aims to overcome rigid schemas for information provisioning and fruition, generally adopted by isolated, pre-packaged mashup tools (Section 3.4).

As described in Section 4.4, in EFESTO the visualizations of data retrieved from data sources are enriched by augmenting the UI templates with the notion of TUX task containers, i.e., elements whose role is to supply task-related functions for manipulation and transformation of task objects along user-defined task flows [16]. In particular, after the first two field studies, we extended EFESTO with a set of tools that, by exploiting functions local to the platform or exposed by re-

mote APIs, provide the possibility to “act” on the extracted contents, for example to collect&save favorites, to compare items, to plot data items on a map, to inspect full content details, or to arrange items in a mind map to highlight relationships [12].

To assess the benefit of this integration, a utilization study inspired by [44], i.e., a study in which participants are required to perform real tasks using the system, has been executed. This study was performed during the interactive session of the international symposium on End-User Development. That session was open to both conference participants and external visitors and was widely advertised by the conference organizers before and during the conference days. The proposed activity allowed users to interact with a mashup tool that enables them to express and respond to their task needs rather directly and dynamically. By observing the end users while using this system we aimed to assess the validity of our ideas on the integration of mashups and TUX principles, and to verify whether making mashups actionable actually provides an added value with respect to the users’ needs and expectations. We also aimed to stimulate a “new way of thinking” towards the definition of systems that really support users in shaping the software environments they interact with, according to their situational needs.

Participants and Design

We were able to recruit a total of 7 participants (4 female), aged between 20 and 60. Single-user interactions with the platform was scheduled. To guide participants through the platform use, they were provided with a scenario in which we reported 4 steps the users have to follow during their interactions.

The main person of the scenario was Maria, who wants to attend a musical event with her friends. She uses the platform to search for forthcoming music events. She also gathers information that can inform the discussion with her friends about which event to attend. Maria logs in the Web platform that offers a workspace where she can retrieve information through the mashup functionality and act on the information through specific functions provided by task containers. The platform is equipped with services providing data on music events, plus some other services of generic utility, e.g., map services. The workspace is also equipped with a collection of task containers. Each container is represented as a box widget with a labelled icon that indicates its intended purpose by highlighting a primary task function, e.g., a World globe for browsing, two side-by-side paper sheets for comparing, a call-out for communicating. When needed, a container

representation can be moved by Maria from this collection into the main area of the workspace, in order to activate its full functional scope.

(Step 1): Maria selects the task container “Events” and chooses “music” as event type. A map is displayed: every music event is represented as a pin at specific coordinates. The details of each event can be inspected through the corresponding pin.

(Step 2): Maria includes the “Selecting” container where she makes a preselection by dragging from the “Events” container those events she is more interested in. She further refines her selection by means of a “Comparing” container, which offers specific features supporting the comparative inspection of items. After this analysis, Maria chooses the three most promising events and removes the others from the “Selecting” container.

(Step 3): Maria drags the “Housing” container in the main area of the workspace, in particular touching the “Selecting” container. In this way, she synchronizes the two containers. Three lists of hotels, one for each different event place, are visualized. For each hotel it is displayed a thumbnail photo, name, price, guests’ rating. Maria performs those actions usually allowed by the hotel booking web sites, i.e., changing dates, ordering, filtering, inspecting details, visualizing the hotels on a map. She selects a couple of hotels for each location. On the basis of the housing information, she decides to reduce the candidate events to only two and eliminates the third from the “Selecting” container.

(Step 4): Maria wants to send an email with a summary of the information related to the two chosen events. Thanks to the “Communicating” container, she is not forced to use an email client external to the work- space she is working on. She has just to drop items from the “Selecting” to the “Communicating” container, where she selects the recipients and the communication channel, e.g. a post on a social network, an email, etc. She decides to send an email. The email addresses of her friends are displayed and the email body is prefilled automatically with the information about the events and the hotels. The message can be edited by Maria before sending. It is noteworthy to remark that Maria is not con- strained to a predefined flow: for example, she could directly move events from “Selecting” to “Communicating”, thus deliberately skipping the “Comparing” or the “Housing” container.

Procedure

The study took place in a quiet and isolated area in the main conference room where we installed the study apparatus 30 minutes before the interactive session. Two HCI researchers were involved in the study. In particular, one (facilitator) was in charge of introducing users to the study and following them during the scenario accomplishment; the second one (observer) took notes and was in charge for recruiting and scheduling the participants.

Each participant interacted for about 30 minutes for a total of 4 hours. They followed the same procedure. First, each participant was asked to sign a consent form. Then, the facilitator showed a quick demo of EFESTO and TUX features on a 15" laptop. Then, the participant was invited to complete the scenarios (also reported on a sheet) by using a 15" laptop. At the end, each participant filled in a printed version of AttrakDiff questionnaire.

Data Collection

Different types of data were collected during the study. In particular, during the system interactions the observer took notes about significant behaviour or externalized comments. All the interactions were audio-video recorded to extract the participants' utterances and comments. The set of collected notes was extended by video and audio analysis, performed by two researchers (audio transcription, double-check, analysis following a semantic approach [19]).

After performing the scenario, the participants filled in a printed version of AttrakDiff[10] questionnaire, that helped us to understand how users personally rate the usability and design of our system. In fact, with the help of pairs of opposite adjectives, users can indicate their perception of the system. These adjective-pairs make a collation of the evaluation dimensions possible. This questionnaire records both the perceived pragmatic quality, the hedonic quality and the attractiveness of an interactive system. In particular, thanks to AttrakDiff, the following system dimensions can be evaluated:

- Pragmatic Quality (PQ): describes the usability of a system and indicates how successfully users are in achieving their goals using the system;

[10] http://attrakdiff.de/

- Hedonic Quality - Stimulation (HQ-S): Mankind has an inherent need to develop and move forward. This dimension indicates to what extent the system can support those needs in terms of novel, interesting, and stimulating functions, contents and interaction- and presentation-styles;
- Hedonic Quality – Identity (HQ–I): indicates to what extent the system allows the user to identify with it;
- Attractiveness (ATT): describes a global values of the system based on the quality perception

Hedonic and pragmatic qualities are independent of one another, and contribute equally to rating attractiveness.

Results and Discussion

The main results of this study come from the AttrakDiff™ questionnaire. Figure 4.12 depicts a *portfolio diagram* that summarizes the *hedonic quality* (HQ) and *pragmatic quality* (PQ) system performances according to the respective confidence rectangles. In particular, in the portfolio-diagram the values of hedonic qualities are represented on the vertical axis (bottom = low value). The horizontal axis represents the value of the pragmatic quality (left = a low value). With respect to this representation, EFESTO was rated as "neutral", even if the confidence interval represented as a small dark rectangle around EFESTO, overlaps into the neighbouring zones. This indicates that there is room for improvements in terms of usability. In terms of hedonic quality, the users are stimulated by EFESTO, but there is still room for improvements. The pragmatic quality confidence interval is quite large. This could be attributed to the limited participant sample who had varying experience with other similar systems and knowledge about tasks performed.

Another perspective of the AttrakDiff™ questionnaire results is provided by the diagram shown in Figure 4.13. In this presentation, hedonic quality distinguishes between the

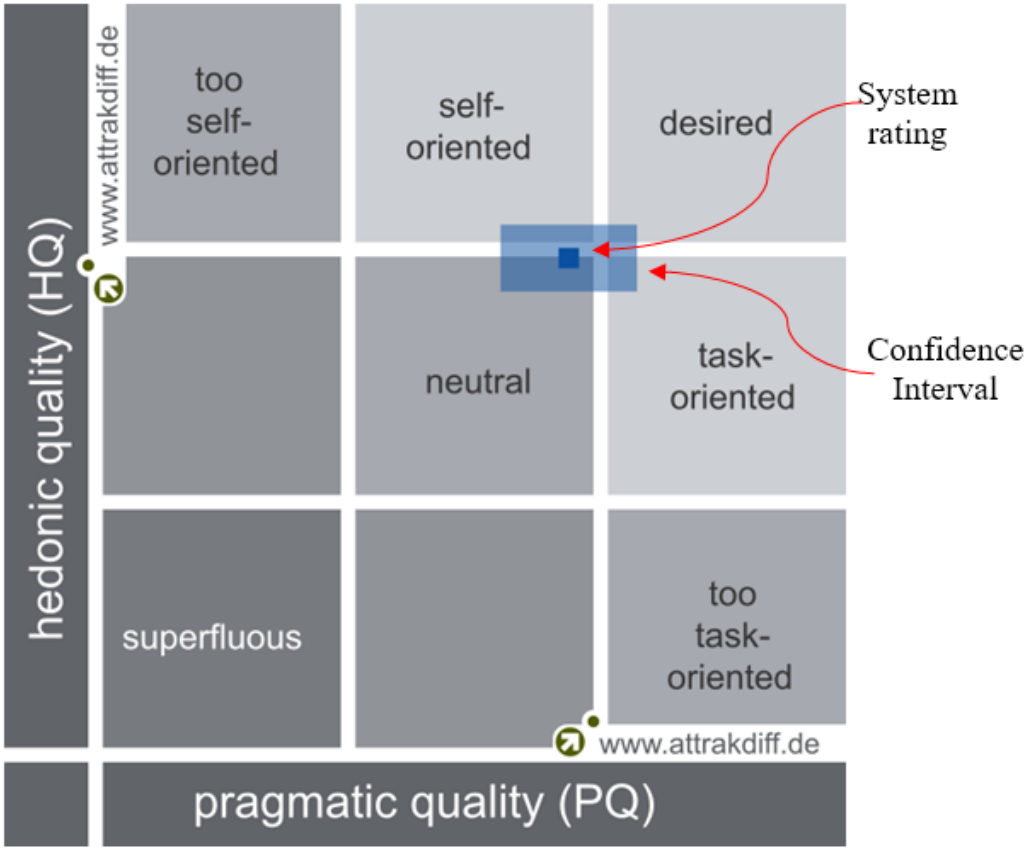

Figure 4.12: Portfolio with average values of the dimensions PQ and HQ and the respective confidence rectangles of the system

stimulation and identity aspects. The *attractiveness* (ATT) rating is also presented. In terms of pragmatic quality, EFESTO meets ordinary standards even if it is located in the average region. Thus, we should improve assistance to users. With regard to *hedonic quality – identity* (HQ-I), EFESTO is located in the average region. With respect to *hedonic quality – stimulation* (HQ-S), EFESTO is located in the above-average region, thus meeting ordinary standards. Further improvements are needed to motivate, engage and stimulate users even more. Finally, being the system's attractiveness value located in the above-average region, the overall impression is that it is very attractive. The diagram shown in Figure 4.14 provides a detailed view of the rating, given by participants to the AttrakDiff adjective-pairs questions, that determined the values of the PQ, HQ-I, HQ-S and ATT dimensions discussed above.

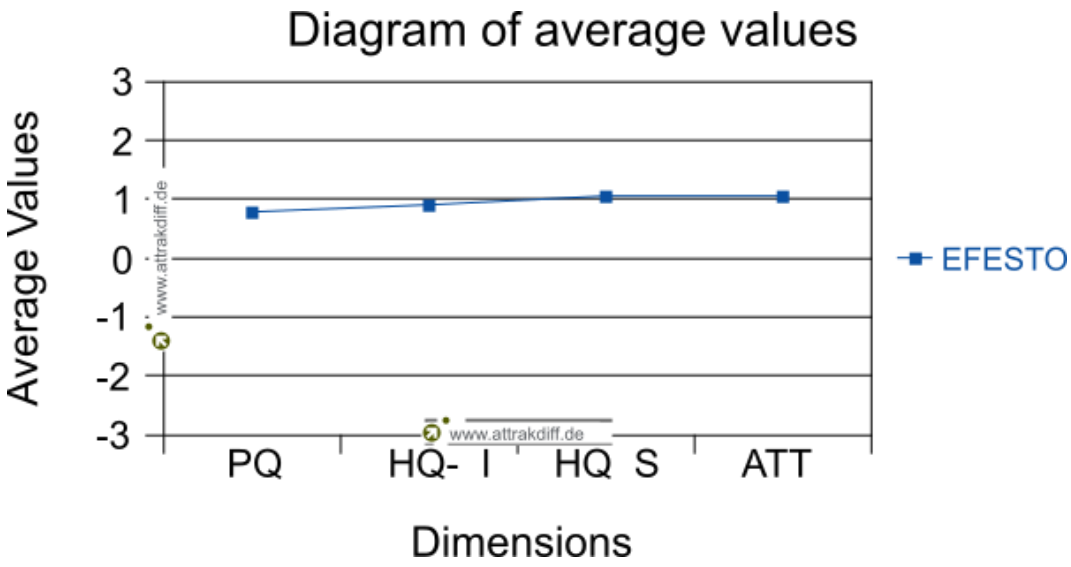

Figure 4.13: Mean values of the four AttrakDiff™ dimensions of our system

From the qualitative data collected with notes and audio-video analysis, different usability problems emerged, for example:

- drag&drop mechanisms are required for including data sources and containers from the left tool bar into the workspace;
- the font size is too small for a 15-inc laptop;
- multiple selection and filter mechanisms misses in data sources and containers;
- a button for deleting all data in a container misses;
- data sources and containers have to be synchronized.

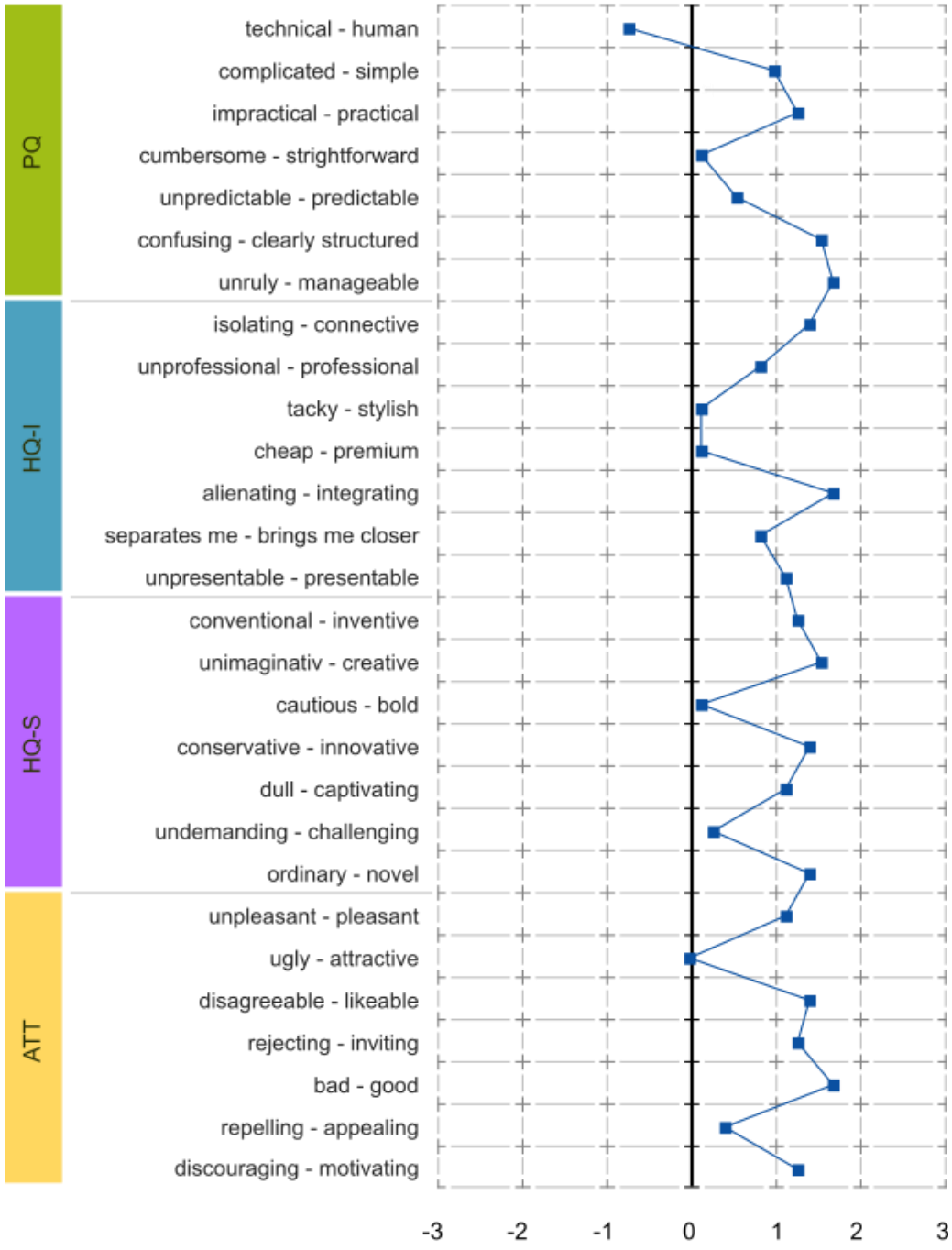

Figure 4.14: Mean values of the AttrakDiff™ adjective-pairs for EFESTO

4.7 Conclusion

The studies presented in the previous sections, allowed us to validate the meta-design model in different contexts and to iteratively improve the EFESTO platform. In particular, the emerged requirements lead to the introduction of the polymorphic data source and the TUX principles.

In light of the results of the studies, the meta-design model can be considered mature and general enough to allow a successful platform customization to different domains. With respect to the polymorphic data source and TUX principles integrated in EFESTO, only preliminary evaluations have been performed. However, early results appear encouraging but other studies should be performed to improve and generalize these aspects.

Chapter 5. Linked Open-Data as New Data Source

5.1 Introduction and Motivations

At the end of 2015, the site programmableweb.com listed more than 14000 API to retrieve data or exploit functionalities. Despite the wide availability of data sources, due to the specific and diverse end users' information needs often no data source can satisfy these needs, as emerged during our field studies (see Section 4.6).

This negatively impacts on the mashup quality. According to [23], *completeness,* together with *accuracy, timeliness,* and *availability,* is a dimension of *data quality*. Data sources available nowadays refer to a portion of a domain and often do not include many details. It is sometimes possible to overcome this limitation by composing different data sources, but, often, when the end users' information need is more specific, no data source could provide the useful information. This is an important limitation in exploiting mashup platforms in real contexts.

To overcome this lack of information and better satisfy the end users' information needs, an original contribution of this PhD research is the definition of a new *polymorphic* data source built upon the Linked Open Data cloud [35]. It is called polymorphic because it provides information that might change with respect to the data sources of which it is composed.

The polymorphic data source is built by exploiting the huge amount of information available in the Linked Open Data (LOD) cloud. In 2009, Tim Berners-Lee defined Linked Data as "a set of best practices for publishing and connecting structured data on the Web" [18]. The goal of the Linked Data project is to publish data so that they are readable by a human and by an automatic agent. The LOD are Linked Data distributed under an open license that allows its reuse for free. At the time of this thesis, there are more than 1000 KB datasets published in the LOD cloud [11].

Nowadays, one of the biggest KBs in the LOD cloud is DBpedia (the structured version of Wikipedia). The DBpedia English version describes 4.58 million things, out of which 4.22 million are classified in its ontology. In the DBpedia ontology, there are 685 classes. Thanks to the availability of this huge amount of information and its semantics structured in the DBpedia ontology, DBpedia has been chosen as the starting point to create the polymorphic data source.

[11] https://datahub.io/dataset

An annotation algorithm, described in Section 5.3, has been developed to annotate each attribute of each service available in a platform with a class of the DBpedia ontology. Each class has to be semantically similar to the attribute.

5.2 Polymorphic data source: a source for many purposes

To explain the idea of a polymorphic data source, let us consider the following scenario that refers to a typical situation in which laypeople want to mashup services, but at a certain point they leave it because they do not find useful data sources. “John is using a mashup platform. He adds the SongKick service to his workspace to find upcoming musical events in his city. He also needs to retrieve, for each event artist, a list of related videos. For this purpose, John composes SongKick artist attribute with YouTube. Now, John has two widgets in his workspace: SongKick and YouTube; the first allows him to search upcoming musical events and the second automatically performs a search (with the artists’ name) each time John clicks on a specific musical artist in the list of upcoming events. Afterwards, he wants to know, for each artist, details such as genre, starting year of activity and artist photo. Searching for useful services on the composition platform, John does not find any service that satisfies his needs. Thus, John is not supported anymore by the platform and he should go to the Web for a usual (manual) search for the specific information. Likely, John has been warned that a new feature (i.e., the polymorphic data sources) is available. Thus, to retrieve the desired information, John expands the SongKick artist attribute with the polymorphic data source. When he chooses this source, the platform shows a list of new properties related to the concept of musical artist. Thus, John decides to create the new data source with the *genre*, the *starting year of activity* and the *artist photo* properties. Henceforward, John can find a list of upcoming events on SongKick and can visualize the additional artist’s information on the polymorphic data source by clicking on a specific artist on SongKick (Figure 5.1)”.

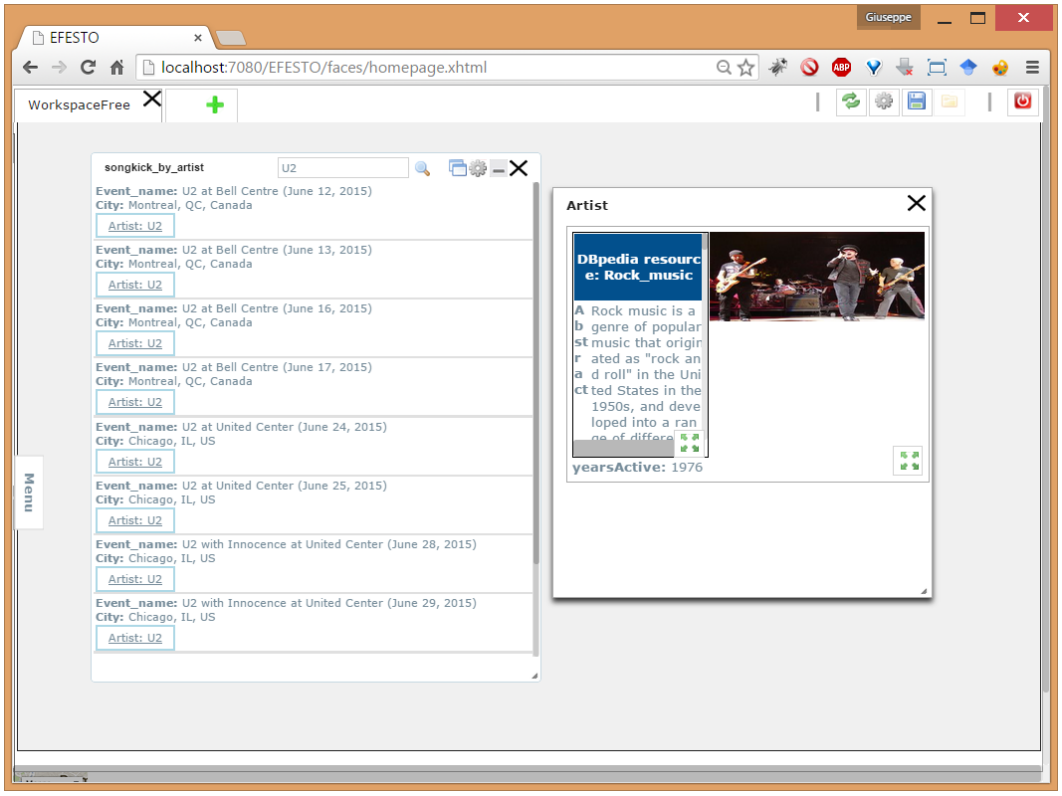

Figure 5.1: Composition of DBpedia polymorphic data source with SongKick artist attribute.

In the previous scenario, John could continue to compose SongKick with the same polymorphic data source starting from other SongKick attributes. For each attribute that John decides to expand, the polymorphic data source provides different properties related to the semantics of the starting attribute (for example, for the SongKick place attribute, properties like borough, census, year and demographics should be shown). Thus, this type of data source is considered polymorphic because it can provide different information (properties) according to the data source attribute that is selected. On the contrary, the classic data sources (YouTube, Wikipedia, etc.) provide the same properties independently of the selected attribute.

With the join function the user composes a service A with a service B, in order to expand the results of service A with details provided by service B. This composition is assisted by a wizard procedure that the user activates by clicking on a gearwheel button in the upper right corner of the service widget. By clicking on this icon, the user 1) selects the attribute that he wants to expand, 2) selects the data source from which to gather information (DBpedia in our case), 3) chooses a visual template to visualize the new results and finally 4) uses drag&drop to map a subset of service attributes into the visual template. The fourth step is the most interesting for the aim of this data source. In fact, while composing a service with

another ‘traditional’ service the list of attributes in step 4 is always the same, by choosing the polymorphic data source the list of attributes is different in relation to the semantics of the selected attribute during step 1.

In the current implementation, the service that provides details is shown as a window only when the user clicks on a specific item (e.g., click on “U2” artist in Figure 5.1). This visualization emerged as requirement during the users’ study described in section 4.3.

5.3 An algorithm for data source annotation

This section describes a semi-automatic algorithm that creates semantic annotations for services available in a composition platform. Its performance evaluation is also reported.

5.3.1 Generation of a set of candidate classes to annotate attributes

In order to create this polymorphic behaviour, a mapping step is required between all data source attributes registered in the platform and the DBpedia ontology classes. In general, this problem falls in the ontology matching area. In order to start from consolidated approaches, the methodologies surveyed in [46, 70] were investigated with the aim of creating an ad-hoc solution based on the literature, without the pretension of building a new ontology matching methodology. Furthermore, a great deal of specific literature in the semantic web area has already been produced for the problems of semantic annotation of a service [78]. These approaches have been taken into account to design the proposed algorithm.

The proposed solution can be classified as a semi-automatic and instance-based annotation algorithm. As is described in the following, it is defined as semi-automatic because a user has to provide a set of example queries (about 10) when a data source is registered in the platform. Furthermore, it is defined as instance-based because it infers a set of candidate classes to annotate the attributes starting from the results (instances) of the queries. The main goal of the algorithm is to annotate each attribute of each service with a DBpedia class that is semantically similar to the attribute. The algorithm is reported in Table 5.1. The criterion for choosing the most important class, as specified in line 12 of Table 5.1, is illustrated in Section 5.3.2.

Table 5.1: The instance-based semi-automatic annotation algorithm

	Input: Set *T* of Triples *t*=(*s*,*A*,*Q*), *s* is a data source, *A* is a set of attributes *a* for s, and *Q* is a set of queries on *s*
	Output: Set *R* of results *r*=(*s*, *L*), *s* is a data source, *L* is a set of <a_i,l_i> where *a* is an attribute of *s* and *l* is its label
1:	**for** each $t \in T$ **do**
2:	**create** *I* as empty set of instance results for queries and an empty set M
3:	**for** each $q \in Q$ **do**
4:	**query** *s* by using *q* and collect instance results into *I*
5:	**end for**
6:	**for** each attribute *a* of *s* **do**
7:	**create** *L_temp* as empty set of labels for *a*
8:	**for** each $i \in I_a$ **do**
9:	**query** DBpedia by using value of *i* and obtain a set *C* of classes
10:	**put** values of *C* into *L_temp*
11:	**end for**
12:	**calculate** most important class *l* in *L_temp* and put <a,l> into L
13:	**end for**
14:	**end for**

An example of the algorithm execution on a real data source is here reported. Let us consider SongKick data source *s* and a set *Q* of queries on it (e.g., London, Liverpool, Rome). This set Q is manually provided only once at the time of service registration in the platform. Each instance of the SongKick results is characterized by the set of attributes *A= {artist, place, event_type, event_name, date}*. The algorithm starts by executing all the queries in Q on SongKick and by collecting all the results in set *I*. For each attribute a_i in A, the algorithm considers all the instances selecting only the a_i attribute values. For example, after having queried SongKick with queries in Q, for the artist attribute the algorithm creates set I_{artist}= *{Ligabue, U2, One Direction, Taylor Swift,...},* that is a set of musical artists who will perform at places stored in Q. The algorithm uses each instance of I_{artist} to query DBpedia. The aim is to find the same instance of each element in I_{artist} as a DBpedia thing and add its classes in set *L_temp*. Obviously, although the instances of each attribute have the same meaning (for example, all artist instances are singers), not all the retrieved DBpedia things are instances of the same class. Thus, at the end of the execution on all the attributes, the results appear as shown in Table 5.2, where the *Class* column indicates the DBpedia classes inferred for each attribute and the *%* column indicates the frequency of the classes at the end of DBpedia queries. Furthermore, when the algorithm queries DBpedia, sometimes the retrieved things are wrongly classified. For example, when DBpedia is queried with the 'Ligabue' string, three thing instances of different classes are retrieved: one instance of Artist (Luciano Ligabue, singer), one instance of Agent (Antonio Ligabue, painter) and one instance of Italian_opera_singer (Ilva Ligabue, opera singer). Obviously, the second and third things are false positives that create noise in the set L_temp. However, it is empirically observed that this noise represents only a tiny percentage (typically less than 4%). For this reason, no classes less than 4% are considered in the rest of the algorithm.

Table 5.2: Candidate classes for annotable attributes of the SongKick data source; the *Class* column indicates the DBpedia class associates; the % coloumn indicates the frequency of each class

Artist		**EventType**		**Location**	
Class	***%***	***Class***	***%***	***Class***	***%***
Agent	14	Event	25	Place	17
Work	13	FilmFestival	25	Settlement	11
MusicaWork	12	Organization	13	PopulatedPlace	11
Organization	7	Television	13	Work	6
Album	7	Agent	12	Album	6
Person	7	TelevisionShow	12	MusicalWork	6
Band	6			Agent	6
Artist	6			Organization	6
Single	5			Building	6
MusicalArtist	5			Architectural	6
…	…			…	…

Until now, the algorithm has collected a set of promising (candidate) classes to annotate attribute data sources. The easiest annotation solution could be to select the most frequent class (In Table 5.2 *Agent* for Artist, *Event* for EventType and *Place* for Location), but the performance of this solution is improved by the second step of the proposed algorithm that takes into account both the class frequency and the ontology tree structure.

5.3.2 Choosing the best class from the set of candidates

The starting point for choosing the best class for each data source attribute is the set of promising classes that the algorithm has built in the previous step. The goal of the next step is to assign to each class in Table 5.2 a rating that takes into account both the class frequency and the ontology tree structure. In the end, the class with the highest score is used to annotate the service attribute. To explain why the tree structure is important, consider the DBpedia sub-tree in Figure 5.2. In that figure the sub-tree of the DBpedia ontology is depicted; it has been built by considering, for an easy explanation, the first ten candidate classes of the attribute Artist in Table 5.2. If the algorithm annotates the Artist attribute with the most frequent class, then the Agent class (14%) will be chosen. However, by looking inside the semantics and the properties that characterize this class, it is evident that the Agent class is too general for the concept of the musical artist of the SongKick Artist attribute. We need to choose a more specific class. There are two aspects to consider, in order to annotate data sources with the best classes: the *class coverage* and the *number of properties*. The class coverage is the percentage of retrieved DBpedia instances covered by each class (node percentages in Figure

5.2) and it is calculated as sum of class percentage with all its sub-class percentages. This value is higher in the ontology top level classes and vice versa, because each class also cover the sub-class instances. On the other hand, the more specific is the class, the more properties the user can choose when creating the polymorphic data source. For this reason, the proposed algorithm tries to find a trade-off between the class coverage and the number of properties.

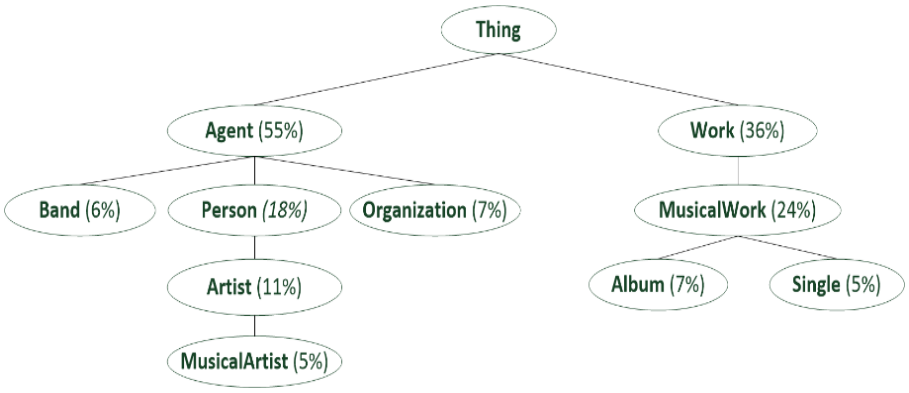

Figure 5.2: Sub-tree of the DBpedia ontology built by using the SongKick artist attributes. For each node, the percentage indicates the class coverage

In order to take into account these aspects, the *x-value* is introduced. It is an index that quantifies the semantic similarity between a class and an attribute considering both the frequency and the ontology structure. In particular, to consider the ontology structure, the algorithm starts by generating all the combinations of the sub-trees built with the classes in the list of candidates. The length of these groups ranges from 2 up N (where N is the number of candidate classes of an attribute). For each class in these sub-trees, the algorithm calculates the x-value. At the end of the computation, each class has many x-values, but only the highest value of each class is considered for the final ranking. The generation of these groups is performed to explore all possible paths in the ontology tree, as also performed in [70]. In particular, when the algorithm generates all sub-trees, new classes could be added to the candidate list. For example, let us consider the Figure 5.2 and suppose that the MusicalWork class is not generated in the candidate list of Table 5.2. During the generation of all sub-trees, when the Single and Album classes are used to create a sub-tree, also MusicalWork is considered because it is their common ancestor. Thus, MusicalWork is added in the candidate list and its x-value is calculated. Choosing the MusicalWork, or an ancestor in general, the coverage could be higher than its sub-class coverage, maintaining a good number of properties that describe the semantics of the considered data source attribute.

The formula of an x-value that estimates the power of a class into each sub-tree is:

$$x - value = (classAncestorRatio + parentPower + nodePower)\,\%class$$

.

Let us analyse in detail each component. The first one is *classAncestorRatio*.

$$classAncestorRatio = \frac{\%class * nLevelScore}{\%commonFather * nLevelScore}$$

This value takes into account the coverage of the current class with respect to the coverage of the first common ancestor in the sub-tree. To penalize classes in the higher levels (such as the ancestors), the numerator and denominator are multiplied by the *nLevelScore*, a number that ranges from 1 to 100 and it is calculated as:

$$nLevelScore = (100/OntologyDepth) * classLevel$$

In this formula, the *OntologyDepth* indicates the maximum depth of the ontology (7 in DBpedia), while *classLevel* specifies the depth of the considered class. It is evident that nLevelScore is high in deeper levels and low in higher levels. In this way, in the classAncestorRatio, the classes at higher levels are penalized.

The second component in the x-value is the *parentPower*. It quantifies the impact of the common ancestor with respect to all the classes at the same level.

$$parentPower = \frac{\%TotRoot}{\%TotRootLevel} \qquad classPower = \frac{\%class}{\%allClasses}$$

In this component, *%TotRootLevel* is the sum of the coverage rate of all classes at the same level of the sub-tree root. The *%TotRoot* is the coverage rate of the sub-tree root. This component is introduced in the x-value to solve the problem of the class sparsity in a tree. Let us consider in the Figure 5.2 the Work and Agent classes. When sub-trees with Work or Agent as ancestors are generated, this component has a high value in Agent sub-classes. In this way, the x-value rewards sub-classes in the Agent branch instead of those in the Work branch.

The third component is the *classPower*, which indicates the weight of the considered class in the sub-tree. In classPower, the *%class* is the percentage of the

class considered in the sub-tree and *%allClasses* is the sum of the percentage of all classes in the considered tree.

At the end of the generation of all trees and the calculation of the x-values, the list of candidate classes is expanded with all the new ancestor classes of the generated sub-tree. Each class has several ratings, one for each group in which it appears. The class with the highest score is selected to annotate the service attributes.

5.3.3 Performance evaluation of the annotation algorithm

To the best of our knowledge, no datasets with data sources attributes exist annotated with DBpedia class. Thus, to establish the performance of the algorithm, two experts created and manually annotated a set of 7 services (for a total of 18 annotable attributes) by using DBpedia classes. In fact, not all the services attributes can be annotated with a DBpedia class (i.e. URL attributes). Furthermore, due to the nature of the algorithm, the numerical attributes (ticket price, temperature, humidity, height, weight, etc.) cannot be annotated because it is impossible to infer the classes from numerical values. As described in section 5.4, this limit can be overcome by combining the proposed approach with natural language processing of the attribute name [49, 78].

To evaluate the performance of the automatic annotation algorithm, a new metric called *Accuracy* is introduced. First, a score has been associated with each attribute comparing its automatic annotation to the manual one (AAA stands for Attribute Automatically Annotated; AMA stands for Attribute Manually Annotated). In particular:

- 10 points if AAA = AMA;
- 8 points if AAA is at the same level of AMA (not the same, but very similar semantic);
- 7 points if AAA is 1 level up/down from AMA as a sub-class or a super-class;
- 5 points if AAA is 2 levels up/down from AMA as a sub-class or a super-class;
- 0 points in all other cases.

The *Accuracy* of the overall automatic annotation is calculated as:

$$Accuracy = \frac{\sum_i^N A_i}{\sum_i^N MAXscoe}$$

In the Accuracy formula the numerator is the sum of the accuracy of all the attributes, instead the denominator is the sum of the MAXscore that is the maximum accuracy that an attribute can have (10 in our case). The final accuracy ranges from 0 to 100. The Accuracy is calculated both for the annotation performed by associating the most frequent classes in Table 5.2 (baseline) and for the annotation performed by associating the classes with the proposed algorithm.

Table 5.3: Accuracy comparison between the baseline and the algorithm

	Accuracy
Baseline	56%
Algorithm	91%

As shown in Table 5.3, it is evident how important it is to consider the ontology structure during the automatic annotation procedure. In fact, in the latter (91%) the accuracy is clearly improved.

This metric is quite different with respect to the ones such as precision and recall used for the service semantic annotations [49]. In fact, the classic precision and recall consider true and false values, if the automatic annotation matches the manual one. However, in our case, we can also consider as good annotations classes like super-classes/subclasses, but penalizing them because they do not match exactly the manual annotation. The penalizing factor has been empirically established.

5.4 Conclusions and future work

This chapter describes a polymorphic data source, which aims to address an important limitation that affects the use of a composition platform in real contexts. In order to build this new polymorphic data source, an annotation algorithm has been developed. Early evaluation results indicate the efficacy of the polymorphic data source thank to the quality of the annotations produced by the algorithm.

Although the algorithm performance appears encouraging, the proposed algorithm does not aim to solve ontology matching problems. It is only based on literature evidences and is an ad-hoc solution for the specific problem. Thus, one aspect that could be addressed in the future is the improvement of the algorithm by investigating techniques as, for example, natural language processing [49, 78], to improve the annotation accuracy and to annotate non-annotable attributes (e.g.,

the numerical attributes). Finally, user studies are planned to evaluate the benefits of a polymorphic data source.

Chapter 6. Cross-Device Mechanisms to Mashup Mobile Devices

6.1 Introduction

Mobile devices, such as tablets and smartphones, are widely available today. One of the more recent contribution of this PhD research is a novel solution to exploits the physical presence of different mobile devices held by a group of users, and cross-device mechanisms to create mashups [11].

For more than the last two decades, cross-device systems have been envisioned to support people during co-located collaborative group work [17, 85]. Most proposals involve different types of devices and often include large interactive displays [6]. Very recent approaches, however, are based on mobile devices, because in co-located groups cross-device interactions are much more likely to occur on personal mobile devices than interaction with large, expensive displays [6, 44, 58, 60, 75, 76]. Designs for mobile cross-device interactions have been already proposed, but they mostly support simple tasks, such as file transfer (e.g., photos, videos, contacts) across devices [25, 27, 44, 45, 58, 59, 76, 89].

With respect to such works, we aim to support complex information seeking and sensemaking tasks that require the mashup of data from different data sources (e.g., web sites or apps displayed on different mobile devices) by multiple group members. As pointed out in this thesis, today such tasks are typically supported by mashup tools that are not conceived for supporting co-located groups of people but rather single users [2, 25] or users that collaborate remotely [5, 52]. There is still little or no support in combining data or Web search results from multiple mobile devices. Existing mobile devices, operating systems and apps are typically not designed for user experiences which seamlessly cross devices; thus people in co-located groups are forced to tedious, time-consuming, and error-prone verbal or written exchange of information about queries to perform and retrieved results [44].

This chapter describes a mashup solution consisting of a set of cross-device mechanisms that allow group of users to formulate their queries and reconfigure the data flow between different mobile devices by physically rearranging them on a desk and/or performing cross-device touch gestures on their display. Our design, enabled by the HuddleLamp technology (see Section 6.3) [75], considers the desk as a physical workspace in which data sources are materialized in space by the mobile devices that display them. In order to quickly express the situational information needs of the group, data sources can be flexibly combined by moving the devices around on the desk or by performing touch gestures.

We first elicited the set of interaction mechanisms during a user study. Such mechanisms were then implemented in a prototype that was assessed during a utilization study whose results are reported. We have also planned an experiment to compare if and how our system outperforms traditional tools and mechanisms in supporting group of co-located users that need to execute data composition tasks.

6.2 Elicitation study

To facilitate users' idea collection and to better understand their mental model about the addressed data composition tasks, we decide on an elicitation study, with carefully selected materials, questionnaires, and tasks.

In order to foster the elicitation of different suggestions, comments and explanations by the involved participants, we used the "partners technique"[64] that consists in performing several focus groups of partners. By enabling partners to fruitfully build upon one another's ideas and asking them to decide on a single preferred action, we hoped to facilitate more reflection and discussion and to elicit more diverse opinions about the possible designs. Similar to [60, 82], our sessions therefore contained an element of co-creation beyond pure elicitation. Participants were encouraged to demonstrate their ideas by using physical paper prototypes, pens and papers, so that they would think about the capabilities and affordances of mobile device form factors instead of technological restrictions (see Figure 6.1). As described later, this study organization enabled participants to provide many novel, elaborate indications, including some details on physical input and visual output. The study protocol was preliminarily assessed by involving two groups of four users each.

Figure 6.1: A group discussing and working with paper prototypes during the elicitation study.

6.2.1 Rationale

The goal of our study was the elicitation of touch gestures and/or physically device arrangement in space to perform the typical data composition operations that, in technical terms, are: *union* of data sources, *join* of data sources (also called merge in many mashup tools [34]), and *data visualization*. We have chosen these tree operations since they are the most common operations allowed by mashup tools [10, 34]. In particular:

- the *union* operation produces one dataset obtained by composing the datasets returned by different data sources,
- for the *join* operation, given two data sources A and B, mashups give the possibility to create a data synchronization schema so that, to query B, users have to click an instance on A. This synchronization consists in linking an attribute of A with the B input;
- the *data visualization* operation shows the dataset returned by a data source according to a visualization technique.

We envisioned the union, join and data visualization operations respectively for the following three tasks:

T1) to query multiple data sources, each displayed on a different device, typing a keyword just once on one device. The synchronized datasets remain on each device and the user performs a “visual” union of them;

T2) to query the data source on the device B with portions of text displayed on the device A;

T3) to visualize on a target device T the data coming from another source device S; the visualization offered on T (e.g., map) could be different from that on S (e.g., list of items).

6.2.2 Task scenarios

In order to facilitate the participants' understanding of the usage context, we proposed three scenarios, each addressing one task. In this way, participants were more focused on discussing a specific topic. A fourth scenario was more complex and required the execution of both tasks T1 and T3. The aim of this last scenario was to bring out inconsistences in the proposed solutions: as emerged during the pilot study, it happens that the same interaction mechanism is proposed for different purposes.

In the first scenario, addressing task T1 (for union operation), participants had to find videos of the "U2" rock band by querying the sites YouTube, Vimeo and Dailymotion, each displayed on a different device. They had to execute the query just once on one of the three devices, avoiding re-typing the same query on the other devices.

In the second scenario, addressing task T2 (for join operation), the participants were asked to use the SongKick site on a smartphone to search for upcoming concerts in London. When a concert was chosen, they had to find details (e.g., recordings, photos) about the concert singer on Wikipedia, which was available on a second smartphone.

In the third scenario, addressing task T3 (for data visualization operation), the participants interacted with the site Zoopla to look for a property in different districts of London. Zoopla provided a list of properties and the participants had to visualize them as pins on Google Maps available on a tablet.

Finally, the fourth scenario, addressing both T1 and T3 tasks, required looking for an apartment in different districts of London using Zoopla. Further information on pollution and public transport in those districts, retrieved by using the same keywords (e.g. London Stratford) and visualized by two sites available on two further smartphones, had to be taken into account (T1). Finally, apartments, air pollution stations and bus/metro stations had to be visualized on Google Maps displayed on a tablet (T3).

6.2.3 Participants

For this study we recruited a total of 25 participants (6 female) aged between 20-28 years ($\bar{x}$=22.6, SD=1.95). They were students at the third year of the bachelor degree course in Computer Science. They were randomly allocated into 5 groups. As revealed by our post-test questionnaire, they were familiar with smartphones and tablets, but they had no experience with mashup of information coming from different web services.

6.2.4 Procedure

The study was performed in collaboration with another HCI researcher on two consecutive days in a university laboratory. It consisted of 5 sessions, one for each group. Three sessions took place the first day. In every session, the group sat around a table and was provided with paper prototypes enabling the task scenarios and sheets of paper and markers for sketching their suggestions. Each participant was also provided with a sheet of paper reporting the four scenarios.

One of the HCI researchers gave a 10-minute presentation to explain the motivation for composing data coming from different sources, also showing examples of some traditional mashup tools. Moreover, a researcher briefly described HuddleLamp and showed a 3-minute video demonstrating some examples of HuddleLamp usage to simplify and introduce the participant to use the spatially-aware features. No solutions for composing data sources were shown to avoid any bias in the participants' proposals. After reading aloud and commenting the first scenario, the researcher asked participants how they would perform the scenario activities. He also specified that they were completely free to propose any data source composition idea, with or without spatial-aware features. The researcher stimulated participants to elaborate new interaction ideas, which were also expressed by sketching new paper prototypes or demonstrated by using the available paper prototypes (e.g., a swipe gesture from one device prototype to another). The researcher also encouraged the discussion on positive and negative aspects of the suggested solutions. Before moving to the next scenario, the group had to agree on the interaction mechanisms they were proposing for carrying out the assigned composition task. The same procedure was repeated for the other three scenarios. The second researcher took notes. Each session was also audio-video taped.

At the end of the session, group participants filled in a questionnaire composed of 21 questions. Eleven questions aimed at collecting participant's demo-

graphic data, and determining their expertise with mobile devices, programming, data retrieval and composition. Four questions investigated participants' understanding of and comfort with the proposed tasks. Two questions addressed the perceived usefulness of the proposed interaction mechanisms for data source composition. The last four questions addressed the pros and cons on the ideas the participants suggested during the study.

6.2.5 Data collection

The data analyzed in the study were collected through: 1) the set of notes taken by the researchers in the study sessions; 2) the audio recording of participants' discussions; 3) the sketches drawn during the sessions; 4) the answers participants gave to the questionnaire; 5) the videos recorded during the sessions.

Two researchers transcribed their notes and the audios and independently double-checked some 65% of the material. The initial reliability value was 85%, thus the researchers discussed the differences and reached a full agreement. The transcripts were analyzed by thematic analysis following a semantic approach [19]. The answers to the open questions of the questionnaire were analyzed through the affinity diagram technique proposed in [72].

6.2.6 Results and Discussion

Each session lasted on average 70 minutes. The analysis of the four questions on participants' understanding of the study design revealed that the four tasks were clearly understood by the participants, who did not have difficulties in realizing what they needed to do during the study. Indeed, each question consisted of a Likert scale that ranged from 1 to 7 (1 very easy – 7 very difficult) and the results revealed a good understating of the four tasks (T1 $\overline{x}$ = 2.12, SD = 1.36; T2 $\overline{x}$ = 2.4, SD = 1.44; T3 $\overline{x}$ = 2.32, SD = 1.41; T4 $\overline{x}$ = 2.8, SD = 1.5).

In the following subsections, we discuss the interaction mechanisms that were proposed by the participants. We also show how they were implemented in the system prototype that was employed in the utilization study reported later in this chapter. In the light of the solutions that users expressed for performing union, join and data visualization, we called the cross-device interaction mechanisms: *Query Broadcasting*, *Flying Join* and *Aggregation&Visualization*.

Query Broadcasting

All the five groups involved in the elicitation study agreed that *spatial proximity*, i.e., putting devices close to create a *physical group*, was the best solution for accomplishing the main requirement of scenario T1, i.e. for performing a union. In fact, the participants proposed that, for automatically executing the query on each device, they had to: 1) put the devices physically close to create a group and then, 2) query one of them to automatically *broadcast* the same query to the other devices. This is the reason why we also call this mechanism *Query Broadcasting*. A device can be simply removed from a group by placing it far from the group.

As an alternative solution, two groups also proposed device bumping, i.e., colliding the devices to create a virtual group that can be queried as in the previous case, similar to the synchronous gesture proposed in [45].

The final solution was to implement the spatial-aware mechanism based on the spatial proximity of the devices. The design and implementation of the prototype then required us to deepen three aspects that superficially emerged during the study. The first one regarded the feedback to show to the users when they create and eliminate a group of devices. To notify users that a device is part of a group, display borders of the grouped devices are highlighted in orange, as shown in Figure 6.2. This also provides feedback on how much a device should be moved close to the others to be a part of a group since the orange borders appear only if the devices are close enough. Moreover, it improves the system status visibility, because all users can easily understand which devices are grouped even if they did not create or see the group creation.

The second aspect we considered was the mechanism to broadcast the query. In fact, different solutions were expressed by the participants. For example, one of them said: “I like that when I move a device close to another one, the second device automatically executes the same query already performed on the first device”. However, this gives rise to the following problem: let us suppose that there are two devices, not yet grouped, on which two different queries have been executed. What happens when they are grouped? Which one broadcasts its query to the other one? In the final design, we opted for broadcasting any query at grouping time; in other word, queries are broadcasted only after the group has been established.

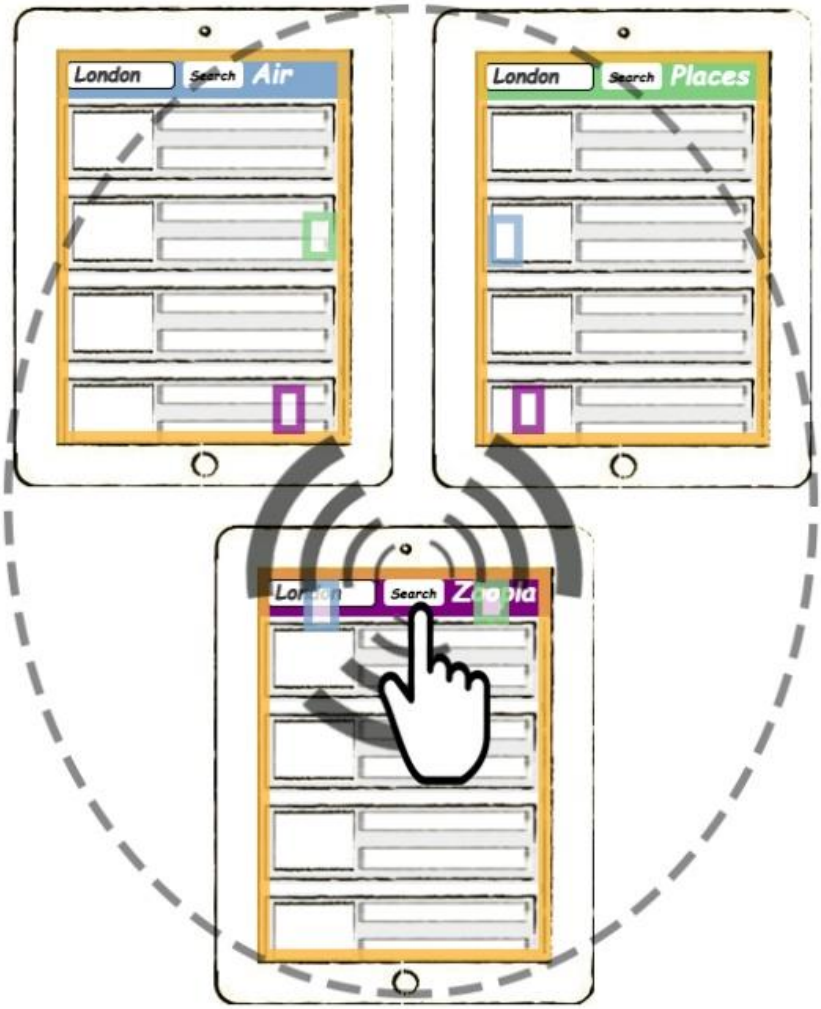

Figure 6.2: Example of *Query Broadcasting*.

The third aspect we deeply investigated was the distance threshold between the devices to consider them a group. Some participants suggested that a small distance should be kept between the devices; other participants said that devices should adhere each other. However, we did not consider this second possibility, because of the HuddleLamp constraint, which is not able to recognize the devices if there is not a distance of at least 0,5 centimetres between them. Taking into account that up to five tablets can be positioned over the desk area recognized by HuddleLamp, we empirically established that devices are grouped when the distance between them is in the range 0.5 – 3 centimetres.

Flying join

All the groups decided that to perform a join, i.e., to indicate that a piece of text has to be used for querying the data source displayed on another device, the most suitable mechanism is the *directional flick.* This solution is a spatial-aware gesture

performed on the device display for "launching" the selected portion of text in the direction of the target device, where it will be automatically used for querying the displayed sites (an interaction similar to the gesture described in [77]). Moreover, two groups also suggested that this task could be performed by touching, for a couple of seconds, a piece of text to activate a contextual menu, where they can choose the target device from a list of devices on the desk. The second solution is a spatial-agnostic interaction similar to the one proposed in [44]. Both the proposed solutions were already compared in [77]: for large targets, directional flick was significantly faster than radar (a technique very similar to the menu proposed by the participants), but was inaccurate for small targets. *Edge Bubbles*, presented in [76], enhances the flick technique; it was shown that it outperforms other methods like contextual menu and radar. Edge Bubbles consists in coloured semi-circles around the edges of the screen, which act as visual proxies for remote devices and indicate the direction in which the remote devices are located on the desk. The locations of the bubbles are defined by imaginary lines connecting the central point of the source device and the central point of the target devices. Each bubble is located where this imaginary line intersects with the edges of the local screen. The positions of the bubbles are updated in real-time and thus always reflect changes in the physical configuration of the devices. In the Edge Bubble technique, dragging and dropping an item onto one of the edge bubbles means copying the dragged item in the target device.

We adopted the Edge Bubble solution to implement the *Flying Join*. In particular, in our prototype dragging and dropping a piece of text onto one of the edge bubbles generates a query on the target device with the dragged text as the keyword. For example, in Figure 6.3 YouTube is queried by dragging and dropping a specific singer's name into the LastFm red rectangle. According to the suggestions of some users participating in the successive pilot study, in order to improve bubble affordance, in our final design, edge bubbles are represented as rectangles instead of semi-circles, thus referring to tablet or smartphone shape (see Figure 6.3). Furthermore, the size of a bubble is provisionally doubled when the dragged text is approaching the proxy, because, during the pilot study users had difficulty dragging a large piece of text into the small bubbles (compare the size of the edge bubbles on the two devices represented in Figure 6.3).

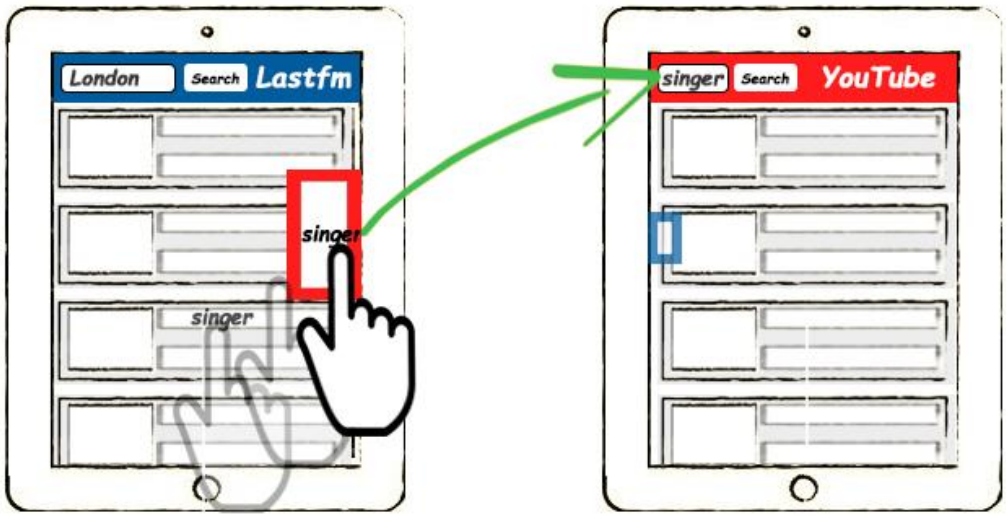

Figure 6.3: Example of *Flying Join*.

Aggregation&Visualization

The third composition mechanism related to the data visualization operation consists in synchronizing one or more source devices S_i, $1 \leq i \leq n$, and one target device T, so that data coming from $S_1 \ldots S_n$ are visualized in T. It is called *Aggregation&Visualization* because it allows the visualization on T of data gathered from a single device or an aggregation of devices previously grouped with the *Query Broadcasting* mechanism. The aggregated data are visualized according to the visualization provided by T, for example, Google Maps.

Data visualization was the most debated operation during the study. Group 1 decided for the bumping between the source device and the target device, a synchronous gesture as the one proposed in [45]. Groups 2 and 3 opted for a button on the target device to open a popup window where the user can choose the source device. Group 4 and 5 decided for spatial proximity, but this solution was already used for the *Query Broadcasting* task. Such an inconsistency clearly emerged during Scenario 4.

In the end, we implemented the proposal of Groups 2 and 3, i.e., a menu on the target device T to select the source device S from which to visualize the data. Moreover, as required by each group, we allowed the selection of not only a source device S, but also devices grouped by using the *Query Broadcasting* mechanism. When a user opens a site on T that provides a specific visualization

(for instance, Google Maps), in the title bar the button *Decant*[12] is available (see Figure 6.4a). By clicking this button, a pop-up appears, which lists all the devices on the desk each identified by the name of the site it is executing. If multiple devices have been grouped for query broadcasting, the name of the group, which has been automatically created by the system by concatenating the site names, is displayed. The user chooses the source devices from the list. From now on, the target device T visualizes the data gathered from the source devices S (Figure 6.4b). Any data updating on S is immediately propagated on T. The synchronization between T and S can be cancelled by using the menu in T or by taking away S from the table. Although at a first glance this can appear to be a spatial-agnostic technique, it also exploits spatial awareness system features, since T lists all the devices currently on the desk.

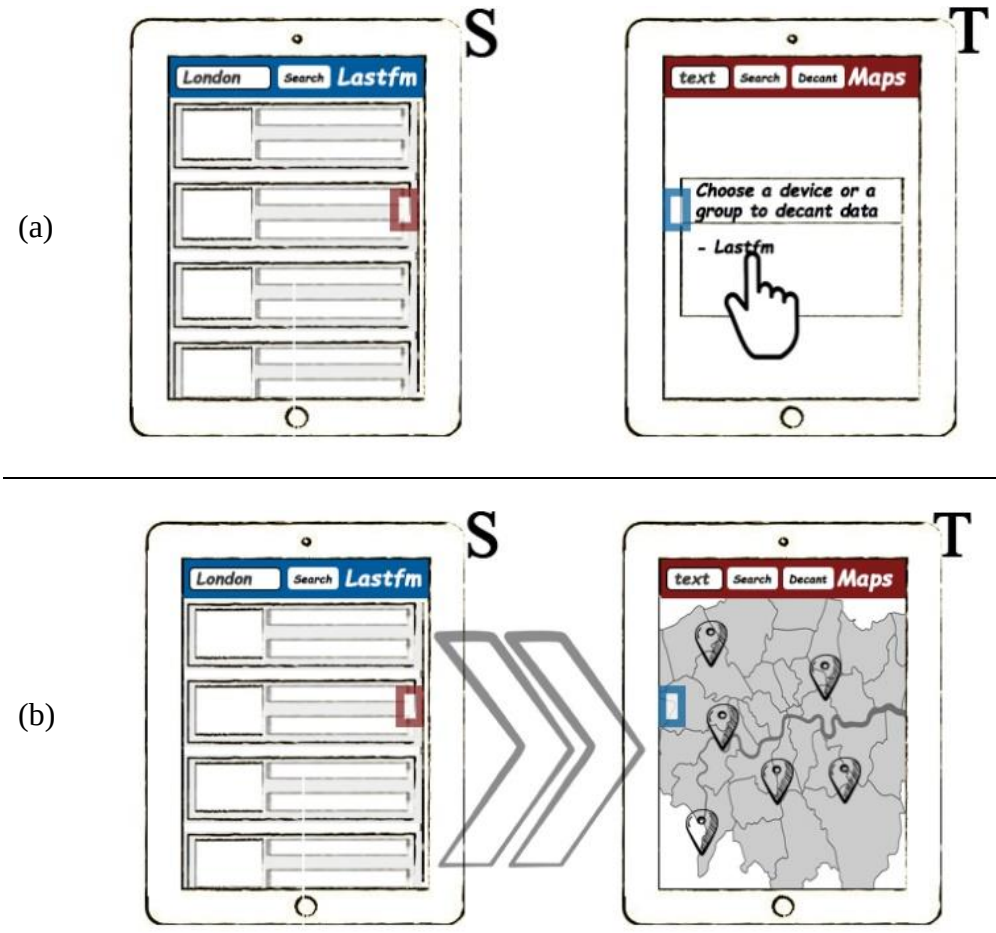

Figure 6.4: Example of *Aggregation&Visualization*: (a) selection of Lastfm as source; (b) visualization of Lastfm items as pins on Google Maps.

[12] We have used 'Decant' as metaphor of pouring and mixing wine in another container, where it assumes the shape of the target container and the new flavour due to the mixing.

6.3 System Technical Details

In our view, the final software should be implemented as a plugin for mobile device browsers that enables the orchestration of web site data and functionalities according to the cross-device interaction mechanisms, which emerged from the elicitation study. Nowadays, this solution is hampered by technological limitations like the strong heterogeneity of data formatting in web pages and the communication lag between devices. At the current stage of our research, we are focusing on the design and evaluation of the composition paradigm. Therefore, we have created a simplified environment reproducing, as accurately as possible, a set of web sites for accomplishing the scenarios designed for testing the cross-device interaction mechanisms.

The system installation consists in setting up the HuddleLamp desk hardware (see Figure 6.1), already described in the related work, and deploying our web application on a Meteor Web server [63]. Users interact with the application using the web browser of the mobile devices connected to a Meteor server and placed on the desk. Initially, the device displays a QR code that allows the lamp to recognize each device on the desk. Once the device is recognized, the application shows a homepage that acts as a hub, where the user can choose which of the available sites will be displayed on that device.

Exiting solutions for cross-device interaction require the installation of either hardware (in the environment [77, 82] or on the devices) or software (to manage the registration of devices [44] or the addition of a functionality [31]). Thanks to HuddleLamp, no hardware or software needs to be installed on the mobile device, since users have just to open a web site. This aspect fosters the adoption of this system in real contexts.

6.4 Utilization study

We designed a utilization study inspired by [44], i.e., a study in which participants are required to perform real tasks using the system (or prototype). This study was performed during the interactive session of the International Symposium on End-User Development (IS-EUD) held in Madrid, Spain, during May 2015. The session was open to both conference participants and external visitors and was widely advertised by the conference organizers before and during the conference days.

Since the aim was understand how novel cross-device mechanisms support seeking and sensemaking tasks, in this study we addressed three research questions:

RQ1) *Are users able to perform co-located collaborative tasks by exploiting the proposed interaction mechanisms*?

RQ2) *Do users like the proposed interaction mechanisms to perform co-located collaborative tasks*?

RQ3) *Would users prefer different interaction mechanisms for the three composition operations?*

6.4.1 Participants and Design

We were able to recruit a total of 13 participants (5 female), aged between 23 and 64 ($\bar{x}$=32, SD=13.74). More specifically, the participants were:

- 2 groups of technology experts (G1 with 3 participants, G3 with 4 participants), i.e. conference attendees who, according to the research papers they presented at the conference or to the comments they provided during the interaction, can be considered technology experts.
- 2 groups of non-experts (G2 with 4 participants, G4 with 2 participants), i.e. conference attendees without technical savvy (e.g. researchers of other disciplines than Computer Science or Engineering) or external visitors.

The study was presented to each group of participants as a demo that gave them the opportunity to user the prototype to accomplish two scenarios. In Scenario 1, the groups were asked to act as friends planning to move to the UK to find a new job. They had to look for a property in a UK city (using the Zoopla site), also considering factors like air pollution (using the UK Air pollution site) and property proximity to bus/metro stations (using the Google Places site). To evaluate the distance between properties, air pollution stations and bus/metro stations, they could use Google Maps to visualize the site results on the map. This scenario was designed to stimulate users in performing *Query Broadcasting* (for example, to query Zoopla, UK Air pollution and Google Places together using keywords like London, Liverpool, etc.) and *Aggregation&Visualization* (for example to visualize results on Google Maps) mechanisms.

In Scenario 2, participants were asked to act as a group of colleagues, who were in Madrid to attend a conference. They wanted to spend an evening to a mu-

sic concert. Their goal was to find a concert in Madrid during the conference days using Last.fm site. They could also use YouTube to search for videos related to the concert singer(s) and Google Maps to locate the concert venue. This scenario was designed to stimulate users to perform a *Flying Join*, for example to search for a specific Last.fm artist on YouTube (by drag & drop of a Last.fm artist name in the YouTube proxy), as well as to search for a specific concert venue (by drag & drop of a Last.fm location name in the Google Maps proxy).

The scenarios execution order was counterbalanced across the four groups to neutralise learning biases. Participants were asked to verbalize their thoughts and comments on their actions according to the think-aloud protocol.

Figure 6.5: A group of four participants (not all visible) and the facilitator discussing during the utilization study.

6.4.2 Procedure

The study took place in a quiet and isolated area in the main conference room where we installed the study apparatus 30 minutes before the interactive session (see Figure 6.5). Two HCI researchers were involved in the study. In particular, one (facilitator) was in charge of introducing users to the study and following

them during the scenario accomplishment; the second one (observer) took notes and was responsible for recruiting and scheduling groups of participants.

Each group interacted for about 60 minutes for a total of 4 hours. They followed the same procedure. First, each group member was asked to sign a consent form. Then, the facilitator showed a quick introduction to the three composition mechanisms by using a 5-minute video (no bias was introduced since the video showed different scenarios). Then, the group was provided with four Apple iPads (9.7" diagonal) on the desk, already set with the Safari browser connected to our web application. The group was invited to complete the scenarios S1 and S2, reported on the two sides of a sheet, by using the available sites with or without the presented special-aware features, as they preferred. At the end, participants filled in an online questionnaire.

6.4.3 Data Collection & Analysis

Different types of data were collected during the study. In particular, during the system interactions the observer took notes about significant behaviour or externalized comments. The set of collected notes was extended by video and audio analysis, performed by two researchers following the same procedure performed after the elicitation study (audio transcription, double-check, analysis following a semantic approach [19]). All the interactions were audio-video recorded to extract the participants' utterances and comments.

After performing the two scenarios, the participants filled in an online questionnaire, which included the System Usability Scale (SUS) form with a further 22 statements. The SUS is a "quick and dirty" tool for measuring the system usability by means of 10 statements, rated by a 5-point Likert scale ranging from *Strongly agree* to *Strongly disagree* [20]. It was chosen because it is high reliable [14], technology agnostic (tested in the last 25 years on hardware, consumer software, websites, cell-phones, etc.) and effective also for evaluating usability of modern technology [21]. We extended the SUS questionnaire with 10 new statements on the users' background and 12 on the evaluation of the proposed composition mechanisms and the performed scenarios. Thus, the final questionnaire had 32 questions. All the quantitative questionnaire data were analysed by *Statsplorer*, a novel software that guides researchers in performing inferential statistical tests [83].

6.4.4 Results

System Usability and Learnability

The 10 SUS questions gave us useful indications about the perceived system usability and learnability. The SUS global score was 68.3 (SD=10.6, 95% CI= [65.4 , 71.2]) and in line with the average SUS scores (69.5) of one thousand studies reported in [15]. To better summarize the meaning of our SUS score, we translated the final score into a subjective label, according to the adjective rating scale introduced in [15]. As shown in Figure 6.6, the vertical line indicates the positioning of our system with respect to the two usability scales. Specifically, our system can be seen as just *good* considering the Adjective Rating scale and, similarly, nearly *acceptable* considering the Acceptability Range scale.

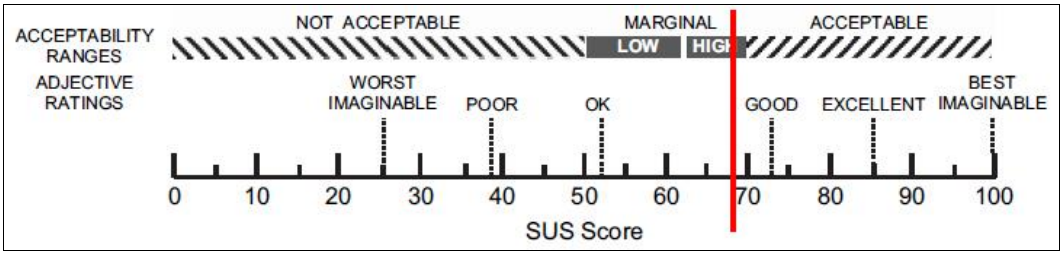

Figure 6.6: System SUS score mapping the adjective ratings, acceptability scores, and school grading scales.

We also investigated the effects of genre and expertise on the system usability. A Mann-Whitney U test was adopted to perform this analysis. No significant differences emerged for genre and expertise, as shown in Table 6.1.

Table 6.1: Mann-Whitney U test to assess the effects of *genre* and *expertise* on system usability

	Perceived Usability with SUS score
Males	$\bar{x}$ = 1.84, 95% CI [1.8,1.9], SD = 0.03
Females	$\bar{x}$ = 1.83, 95% CI [1.7,1.9], SD = 0.09
Test	*U = 24.00, p = .6039*
Experts	$\bar{x}$ = 67.50, 95% CI [63.4,71.6], SD = 5.51
Non-Ex	$\bar{x}$ = 69.17, 95% CI [58.2,80.2], SD = 13.74
Test	*U = 19.50, p = .885*

According to [56], we split the SUS score into two factors, i.e. *System Learnability* (considering statements #4 and #10) and *System Usability* (all the other statements). In particular, the *System Learnability* score was 62,5 (SD=25.5, 95% CI= [55.6, 69.4]), while the *System Usability* score was 69.7 (SD=8.5, 95%

CI=[67.4, 72.0]). A T-Test revealed a significant difference at $p < 0.05$ (T-value = 2.63, P-Value = 0.01). Thanks to the observer notes and audio-video analysis, we could associate the lower learnability score to some drawbacks related to the *Aggregation&Visualization* mechanism. This is the most evident mechanism that pushed users to ask for the support of a technical person (SUS statement #2), i.e., the help of the observer, or to learn different things before getting with this system (SUS statement #10). i.e., to acquire further knowledge for the right task accomplishment.

Cross-device Mechanism Usability

The administered questionnaire included 12 statements (2 open questions and 10 close questions) to collect data on the perceived usability of specific aspects, in particular on the mechanisms to perform the three composition operations and on the system use in the specific scenarios. The closed questions were formulated as a 5-point Likert scale ranging from *Very difficult* (1) to *Very easy* (5).

Answers to the three closed questions addressing each interaction mechanism indicated a high level of perceived usability. The obtained values were: for *Query Broadcasting* $\bar{x}$=3.92, SD 0.86; for *Flying Join* $\bar{x}$=4.00, SD=0.81; *Aggregation&Visualization* $\bar{x}$=4.15, SD= 0.90. Indeed, participants had no problems in understanding and performing the three mechanisms.

We also investigated the effects of genre and expertise for the usability of the three composition mechanisms. For the analysis, a Mann-Whitney U test was conducted and no significant differences emerged in all the cases, as shown in Table 6.2.

Table 6.2: Details of the Mann-Whitney U test used to assess the effects of *genre* and *expertise* on the interaction mechanisms

	Query Broadcasting	**Flying Join**	**Aggregation&Visualization**
Males	$\bar{x}$ = 3.75 95% CI [3.3,4.2] SD = 0.66	$\bar{x}$ = 4.13 95% CI [3.7,4.5] SD = 0.60	$\bar{x}$ = 4.38 95% CI [3.9,4.9] SD = 0.70
Females	$\bar{x}$ = 4.20 95% CI [3.3,5.1] SD = 0.98	$\bar{x}$ = 3.80 95% CI [2.9,4.7] SD = 0.98	$\bar{x}$ = 3.80 95% CI [2.9,4.7] SD = 0.98
Test	*U = 19.50* *p = .8796*	*U = 22.50* *p = .7367*	*U = 26.50* *p = .3451*
Experts	$\bar{x}$ = 3.86 95% CI [3.4,4.3] SD = 0.64	$\bar{x}$ = 4.14 95% CI [3.7,4.6] SD = 0.64	$\bar{x}$ = 4.43 95% CI [3.9,5.0] SD = 0.73
Non-Ex	$\bar{x}$ = 4.00 95% CI [3.2,4.8] SD = 1.00	$\bar{x}$ = 3.83 95% CI [3.1,4.6] SD = 0.90	$\bar{x}$ = 3.83 95% CI [3.1,4.6] SD = 0.90
Test	*U = 19.50* *p = .8796*	*U = 24.00* *p = .6817*	*U = 28.50* *p = .2824*

Perceived User Satisfaction

Besides evaluating each composition mechanism, we also assessed the perceived satisfaction with respect to performing the two assigned scenarios. In particular, we asked participants if they were satisfied with the *ease* and the amount of *time* given to complete the scenarios. As a result, for Scenario 1, participants felt satisfied about the ease of completing the scenario ($\bar{x}$=3.76, SD=0.72) and the required amount of time ($\bar{x}$=3.76, SD=0.83). For Scenario 2, however, they felt even more satisfied about the ease of completing the scenario ($\bar{x}$=4.14, SD=0.55) and the required amount of time ($\bar{x}$=4.23, SD=0.93). These results revealed that the participants thought that the system adequately supported the two different co-located collaborative scenarios.

Furthermore, we also investigated the effects of genre and expertise for the easiness and the time given to perform the two scenarios. A Mann-Whitney U test was conducted and no significant differences emerged in all the cases, as shown in Table 6.3 and

Table 6.4.

Table 6.3: Details of the Mann-Whitney U test used to assess effects of *genre* and *expertise* on the scenarios easiness

	Scenario 1 easiness	**Scenario 2 easiness**
Males	$\bar{x}$ = 3.75 95% CI [3.2,4.3] SD = 0.83	$\bar{x}$ = 3.75 95% CI [3.2,4.3] SD = 0.83
Females	$\bar{x}$ = 3.80 95% CI [3.5,4.2] SD = 0.40	$\bar{x}$ = 3.80 95% CI [3.5,4.2] SD = 0.40
Test	*U = 18.00, p = .8113*	*U = 18.00, p = .8113*
Experts	$\bar{x}$ = 3.86 95% CI [3.2,4.5] SD = 0.83	$\bar{x}$ = 3.86 95% CI [3.2,4.5] SD = 0.83
Non-Ex	$\bar{x}$ = 3.67 95% CI [3.3,4.0] SD = 0.47	$\bar{x}$ = 3.67 95% CI [3.3,4.0] SD = 0.47
Test	*U = 23.00, p = .8158*	*U = 23.00, p = .8158*

Table 6.4: Details of the Mann-Whitney U test used to assess effects of *genre* and *expertise* on the scenarios time

	Scenario 1 time	**Scenario 2 time**
Males	$\bar{x}$ = 3.75 95% CI [3.08,4.42] SD = 0. 79	$\bar{x}$ = 4.38 95% CI [3.69,5.06] SD = 0.99
Females	$\bar{x}$ = 3.80 95% CI [3.45,4.15] SD = 0.40	$\bar{x}$ = 4.00 95% CI [3.45,4.55] SD = 0.63
Test	*U = 17.50, p = .753*	*U = 27.50, p = .2661*
Experts	$\bar{x}$ = 3.71 95% CI [3.1,4.4] SD = 0.88, n = 7	$\bar{x}$ = 4.14 95% CI [3.4,4.9] SD = 0.99, n = 7
Non-Ex	$\bar{x}$ = 3.83 95% CI [3.3,4.4] SD = 0.69	$\bar{x}$ = 4.33 95% CI [3.8,5.0] SD = 0.75
Test	*U = 18.50, p = .7587*	*U = 19.50, p = .8768*

System Usage and User Behaviour

The qualitative data collected with notes and audio-video analysis provided important hints on user behaviour, ideas and system improvements. The most significant problem observed and confirmed by the participants' utterances was related to the *Aggregation&Visualization* mechanism, sometimes confused with *Query Broadcasting*. This problem occurred for three out of four groups. Indeed, when these three groups tried to visualize for the first time Zoopla data in Google Maps, they moved the Zoopla device close to the Google Maps device, thus communicating a *Query Broadcasting* instead of *Aggregation&Visualization*. In these cases, the groups were helped by the facilitator to remember the video previously

shown. No other help was needed. This problem is in line with the results obtained during the elicitation study, where 50% of groups liked to change visualization by moving the devices close and 50% by using a menu. More attention has to be dedicated to this aspect in future work.

Minor problems were detected for the *Query Broadcasting* and the *Flying Join* mechanism. For the former, we observed that participants took some time to appropriate the spatial-aware cross-device mechanism. In fact, in two groups, when they tried at the beginning of Scenario 1 to query different devices with the same city name, they typed the same query on different devices. Only after some seconds, they remembered that they could broadcast the query. For example, a member of G4 said: "*ops, we could search for London more quickly using the function shown in the video*".

Regarding the *Flying Join*, the main problem observed is related to the way it was implemented in the prototype. In fact, in a real setting, every piece of text can be dragged from the web page onto the proxies, while, in our system, we narrowed this possibility allowing users to drag only one specific piece of text (e.g. the properties addresses from Zoopla) onto the specific proxies of Google Maps and YouTube. This constraint limited some needs, which emerged during the study. In fact, some users (in G1, G2 e G3) wanted to perform unforeseen Flying Joins, for example, the drag of a pollution station from UK Air Pollution onto the Zoopla proxy to find on the Zoopla device all the properties around a non-polluted area.

We also collected important hints on how to improve the system usability. One of the most important is related to the proxy memorability. In fact, to support moving data across different devices, inspired by the Edge Bubble solution [76], we created proxies as rectangles with the same colour of the toolbar shown on the other devices. Some participants had difficulty remembering this association. Two of them (in G2 and G4) suggested adding the site name or logo inside the proxy.

6.5 Discussion

The main goal of this study was to answer the three research questions. For RQ1, about users' ability to perform real co-located collaborative tasks by exploiting spatial awareness mechanisms, we can say that users were able to exploit the designed solutions in different co-located collaborative scenarios. According to the questionnaire results, participants appreciated the easiness and efficiency in exe-

cuting the tasks required by the two scenarios (see Table 6.3 and Table 6.4). No differences emerged between experts and non-experts, or between males and females. We also triangulated the quantitative questionnaire results with the qualitative data (open questions, video, audio) to better answer RQ1. In particular, the four groups were able to complete the scenarios, even though some problems related to the *Aggregation&Visualization* mechanism forced three groups to ask once for help. As already highlighted, this is an aspect that needs further investigation.

RQ2 addressed user satisfaction about the interaction mechanisms to perform co-located collaborative tasks. The questionnaire results showed a high level of satisfaction about the proposed interaction mechanisms. No differences were found between experts and non-experts, or between males and females (see Table 6.2). Useful indications emerged on how to further enhance the three interaction mechanisms. For example, the proxy memorability could be improved by adding the logo/site name inside, as suggested by some participants. It is worth remarking that another study already investigated proxy usage, for example, to compare different solutions for cross-device interaction [76]. In that study only simplified tasks in laboratory setting were performed, thus proxy memorability was not a problem. Instead, it emerged in our study because it considered a more complex and real situation. Finally, system implementation should be also improved in order to allow *Flying Join* to and from any site.

About RQ3, i.e. if users would prefer different interaction mechanisms for the three composition operations, participants provided some remarks only for the *Aggregation&Visualization* mechanism. Indeed, in some cases, participants did not immediately understand the right usage. Nevertheless, it took very little to realize how to correctly interact. Since we want to reduce as much as possible users' misunderstanding about the actions to perform, we are going to experiment some alternatives for the *Aggregation&Visualization* mechanism.

Chapter 7. Conclusion and future work

The research reported in this thesis focused on the definition of models and interaction paradigms to enable end users in creating pervasive workspaces by mashing-up heterogeneous resources. In fact, despite the growing availability of mashup tools, the composition paradigms they adopt are still difficult for non-technical users. We grounded our work on experiences reported in literature about new paradigms for mashup composition and on the lessons learnt on End-User Development (EUD). EFESTO is the mashup platform that implements the main results of the research work carried out. It supports end users in performing mashups thanks to novel visual interaction paradigms; it allowed us to validate our models and composition paradigms. The EFESTO three-layer architecture allows domain customization activities, which simplify the composition languages that represent a barrier for the adoption of mashup platforms by non-technical people. Thus, this architecture enables all the stakeholders to contribute to the overall process of creating mashup applications.

According to the requirements emerged in the field studies performed during the iterative design, development and testing process, the research interests became wider and new directions were investigated, leading to novel contributions, i.e. the creation of a new polymorphic data-source based on Linked Open Data and the implementation of Transformative User eXperience principles in EFESTO. As last step, a novel interaction paradigm to perform mashup with mobile devices is reported in this thesis. In particular, a set of interaction mechanisms, both spatial-aware and cross-device, allows groups of users to perform mashup by using mobile devices. These mechanisms were implemented in a prototype and assessed during a utilization study. This is still work in progress, but the first results appear encouraging and show that participants quickly managed to use and appreciated the proposed spatial-aware cross-device interaction mechanisms to perform co-located collaborative tasks. Interesting indications for improvements and further research also emerged.

As future work, we will introduce part of the knowledge built during this research in the Internet of Things (IoT) technology. The current integrated technologies confer intelligence to any type of objects and connect them to the network to be controlled remotely. Unfortunately, the behaviour of these objects is determined by specific programs, thus only programmers can do it. The challenge we are going to address is to empower end-users to determine the behaviour of smart objects.

As final remark, some parts of this thesis describe work that has been published in articles of which Giuseppe Desolda is author. Such parts are Sections 3.2

and 3.4, Chapter 4, 5 and 6. Chapter 5 and 6 refer to work autonomously carried out by Giuseppe Desolda, or in which his contribution has been preeminent. The articles related to Sections 3.2 and 3.3 and to Chapter 4 have been written by Giuseppe Desolda and other authors, in particular Prof. Maria F. Costabile, who has been his supervisor, and other researchers of the IVU Laboratory of the Computer Science Department of the University of Bari Aldo Moro (of which he is a member). The collaboration has been primarily on models and methodologies whose definition is based on the joint work of more people, everyone giving his/her specific contribution. Even in such articles, the contribution of Giuseppe Desolda has been very significant: he has implemented the mashup platform, he has carried out the interaction design of the developed prototypes, he has played an important role in the design and execution of the reported user studies.

References

1. Aghaee, S., Nowak, M., Pautasso, C. (2012). Reusable decision space for mashup tool design. In: *Proc. of ACM SIGCHI symposium on Engineering Interactive Computing Systems (EICS '12)*. Copenhagen (Denmark). 25 – 28 June. pp. 211-220. ACM, New York, NY, USA.
2. Aghaee, S., Pautasso, C. (2014). End-User Development of Mashups with NaturalMash. In *Journal of Visual Languages & Computing* 25(4)**,** 414-432
3. Altinel, M., Brown, P., Cline, S., Kartha, R., Louie, E., Markl, V., Mau, L., Ng, Y.-H., Simmen, D., Singh, A. (2007). Damia: a data mashup fabric for intranet applications. In: *Proc. of International Conference on Very Large Data Bases (VLDB '07)*. University of Vienna (Austria). 23 - 27 September pp. 1370-1373. VLDB Endowment,
4. Ardito, C., Bottoni, P., Costabile, M.F., Desolda, G., Matera, M., Piccinno, A., Picozzi, M. (2013). Enabling End Users to Create, Annotate and Share Personal Information Spaces. In Dittrich, Y., Burnett, M., Mørch, A., Redmiles, D. *End-User Development - Is-EUD 2013*. (Vol. LCNS 7897, pp. 40-55), Berlin Heidelberg, Springer Verlag.
5. Ardito, C., Bottoni, P., Costabile, M.F., Desolda, G., Matera, M., Picozzi, M. (2014). Creation and Use of Service-based Distributed Interactive Workspaces. In *Journal of Visual Languages & Computing* 25(6)**,** 717-726
6. Ardito, C., Buono, P., Costabile, M.F., Desolda, G. (2015). Interaction with large displays: a survey. In *ACM Computing Survey* 47(3)**,** 1-38
7. Ardito, C., Buono, P., Costabile, M.F., Lanzilotti, R., Piccinno, A. (2012). End users as co-designers of their own tools and products. In *Journal of Visual Languages & Computing* 23(2)**,** 78-90
8. Ardito, C., Costabile, M.F., De Angeli, A., Lanzilotti, R. (2012). Enriching archaeological parks with contextual sounds and mobile technology. In *ACM Transactions on Computer-Human Interaction (TOCHI)* 19(4)**,** 29
9. Ardito, C., Costabile, M.F., Desolda, G., Lanzilotti, R., Matera, M., Piccinno, A., Picozzi, M. (2014). User-Driven Visual Composition of Service-Based Interactive Spaces. In *Journal of Visual Languages & Computing* 25(4)**,** 278-296
10. Ardito, C., Costabile, M.F., Desolda, G., Lanzilotti, R., Matera, M., Picozzi, M. (2014). Visual Composition of Data Sources by End-Users. In: *Proc. of International Working Conference on Advanced Visual Interfaces (AVI '14)*. Como (Italy). 28-30 May pp. 257-260. ACM, New York, NY, USA.
11. Ardito, C., Costabile, M.F., Desolda, G., Latzina, M., Matera, M. (2015). Hands-on Actionable Mashups. In Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. *End-User Development - Is-EUD 2015*. (Vol. LCNS 9083, pp. 295-298), Berlin Heidelberg, Springer Verlag.
12. Ardito, C., Costabile, M.F., Desolda, G., Latzina, M., Matera, M. (2015). Making Mashups Actionable Through Elastic Design Principles. In Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. *End-User Devel-*

opment - Is-EUD 2015. (Vol. LCNS 9083, pp. 236-241), Berlin Heidelberg, Springer Verlag.
13. Ardito, C., Lanzilotti, R., Costabile, M.F., Desolda, G. (2013). Integrating traditional learning and games on large displays: an experimental study. In *Journal of Educational Technology & Society* 16(1)**,** 44-56
14. Bangor, A., Kortum, P., Miller, J. (2008). The system usability scale (SUS): An empirical evaluation. In *International Journal of Human-Computer Interaction* 24(6)**,** 574-594
15. Bangor, A., Kortum, P., Miller, J. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. In *Journal of usability studies* 4(3)**,** 114-123
16. Beringer, J., Latzina, M. (2015). Elastic workplace design. In *Designing Socially Embedded Technologies in the Real-World*. (Vol. pp. 19-33), Springer.
17. Biehl, J.T., Bailey, B.P. (2004). ARIS: an interface for application relocation in an interactive space. In: *Proc. of Graphics Interface (GI '04)*. London, Ontario (Canada). 17 - 19 May. pp. 107-116. Canadian Human-Computer Communications Society, London, Ontario, Canada.
18. Bizer, C., Heath, T., Berners-Lee, T. (2009). Linked data-the story so far. In *Semantic Services, Interoperability and Web Applications: Emerging Concepts* 205-227
19. Braun, V., Clarke, V. (2006). Using Thematic Analysis in Psychology Qualitative Research in Psychology, 3, 77-101. In *Bristol: University of the West of England*
20. Brooke, J. (1996). SUS: A quick and dirty usability scale. In Jordan, P.W., Weerdmeester, B., Thomas, A., McLelland, I.L. *Usability evaluation in industry*. (Vol. pp. 189-194), Taylor and Francis.
21. Brooke, J. (2013). SUS: a retrospective. In *Journal of Usability Studies* 8(2)**,** 29-40
22. Cabitza, F., Fogli, D., Piccinno, A. (2014). “Each to His Own”: Distinguishing Activities, Roles and Artifacts in EUD Practices. In Caporarello, L., Di Martino, B., Martinez, M. *Smart Organizations and Smart Artifacts*. (Vol. 7, pp. 193-205), Springer International Publishing.
23. Cappiello, C., Daniel, F., Matera, M. (2009). A Quality Model for Mashup Components. In Gaedke, M., Grossniklaus, M., Díaz, O. *Web Engineering - ICWE 2009*. (Vol. 5648, pp. 236-250), Springer Berlin Heidelberg.
24. Cappiello, C., Matera, M., Picozzi, M. (2015). A UI-Centric Approach for the End-User Development of Multidevice Mashups. In *ACM Transaction Web* 9(3)**,** 1-40
25. Cappiello, C., Matera, M., Picozzi, M., Sprega, G., Barbagallo, D., Francalanci, C. (2011). DashMash: A Mashup Environment for End User Development. In Auer, S., Díaz, O., Papadopoulos, G. *Web Engineering - ICWE 2011*. (Vol. 6757, pp. 152-166), Springer Berlin Heidelberg.
26. Casati, F. (2011). How End-User Development Will Save Composition Technologies from Their Continuing Failures. In Costabile, M.F., Dittrich, Y., Fischer, G., Piccinno, A. *End-User Development - Is-EUD 2011*. (Vol. 6654, pp. 4-6), Springer Berlin Heidelberg.
27. Chen, N., Guimbretière, F., Sellen, A. (2013). Graduate student use of a multi-slate reading system. In: *Proc. of SIGCHI Conference on Human*

Factors in Computing Systems (CHI '13). Paris (France). 27 April - 2 May. pp. 1799-1808. ACM, New York, NY, US.

28. Costabile, M.F., Fogli, D., Mussio, P., Piccinno, A. (2006). End-User Development: The Software Shaping Workshop Approach. In Lieberman, H., Paternò, F., Wulf, V. *End User Development - Is-EUD 2006*. (Vol. 9, pp. 183-205), Springer Netherlands.
29. Costabile, M.F., Fogli, D., Mussio, P., Piccinno, A. (2007). Visual Interactive Systems for End-User Development: A Model-Based Design Methodology. In *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 37(6)**,** 1029-1046
30. Costabile, M.F., Mussio, P., Provenza, L.P., Piccinno, A. (2009). Supporting End Users to Be Co-designers of Their Tools. In: *Proc. of International Symposium on End-User Development (IS-EUD '09)*. Siegen (Germany). 2 - 4 April. pp. 70-85. Springer-Verlag, 1530509.
31. Dachselt, R., Buchholz, R. (2009). Natural throw and tilt interaction between mobile phones and distant displays. In: *Proc. of CHI '09 Extended Abstracts on Human Factors in Computing Systems (CHI '09 EA)*. Boston, MA (USA). 4 - 9 April. pp. 3253-3258. ACM, New York, NY, US.
32. Daniel, F. (2015). Live, Personal Data Integration Through UI-Oriented Computing. In Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. *Engineering the Web in the Big Data Era - ICWE 2015*. (Vol. 9114, pp. 479-497), Springer International Publishing.
33. Daniel, F., Casati, F., Benatallah, B., Shan, M.-C. (2009). Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In Laender, A.F., Castano, S., Dayal, U., Casati, F., Oliveira, J. *Conceptual Modeling - ER 2009*. (Vol. 5829, pp. 428-443), Springer Berlin Heidelberg.
34. Daniel, F., Matera, M.: Mashups: Concepts, Models and Architectures. Springer (2014)
35. Desolda, G. (2015). Enhancing Workspace Composition by Exploiting Linked Open Data as a Polymorphic Data Source. In Damiani, E., Howlett, R.J., Jain, L.C., Gallo, L., De Pietro, G. *Intelligent Interactive Multimedia Systems and Services - KES-IIMSS 2015*. (Vol. 40, pp. 97-108), Berlin Heidelberg, Springer International Publishing.
36. Desolda, G., Ardito, C., Matera, M. (2015). EFESTO: A platform for the End-User Development of Interactive Workspaces for Data Exploration. In Daniel, F., Pautasso, C. *Rapid Mashup Development Tools - Rapid Mashup Challenge in ICWE 2015*. (Vol. 591, pp. 63 - 81), Berlin Heidelberg, Springer Verlag.
37. Díez, D., Mørch, A., Piccinno, A., Valtolina, S. (2013). Cultures of Participation in the Digital Age: Empowering End Users to Improve Their Quality of Life. In Dittrich, Y., Burnett, M., Mørch, A., Redmiles, D. *End-User Development - Is-EUD 2013*. (Vol. 7897, pp. 304-309), Springer Berlin Heidelberg.
38. DIS, I.: 9241-210: 2010. Ergonomics of human system interaction-Part 210: Human-centred design for interactive systems. International Standardization Organization (ISO). Switzerland (2009)
39. Ennals, R., Brewer, E., Garofalakis, M., Shadle, M., Gandhi, P. (2007). Intel Mash Maker: join the web. In *SIGMOD Rec.* 36(4)**,** 27-33

40. Fischer, G. (2011). Understanding, fostering, and supporting cultures of participation. In *interactions* 18(3)**,** 42-53
41. Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A.G., Mehandjiev, N. (2004). Meta-design: a manifesto for end-user development. In *Commun. ACM* 47(9)**,** 33-37
42. Fogli, D., Piccinno, A. (2013). Co-evolution of End-User Developers and Systems in Multi-tiered Proxy Design Problems. In Dittrich, Y., Burnett, M., Mørch, A., Redmiles, D. *End-User Development - Is-EUD 2013*. (Vol. 7897, pp. 153-168), Springer Berlin Heidelberg.
43. Ghiani, G., Manca, M., Paternò, F. (2015). Authoring context-dependent cross-device user interfaces based on trigger/action rules. In: *Proc. of 4th International Conference on Mobile and Ubiquitous Multimedia (MUM '15)*. Linz, Austria. pp. 313-322. ACM, New York, NY, USA.
44. Hamilton, P., Wigdor, D.J. (2014). Conductor: enabling and understanding cross-device interaction. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. Toronto, Ontario (Canada). 18 - 23 April. pp. 2773-2782. ACM, New York, NY, USA.
45. Hinckley, K. (2003). Synchronous gestures for multiple persons and computers. In: *Proc. of ACM symposium on User interface software and technology (UIST '03)*. Vancouver (Canada). 02 - 05 November. pp. 149-158. ACM, New York, NY, USA.
46. Hooi, Y., Hassan, M.F., Shariff, A. (2014). A Survey on Ontology Mapping Techniques. In Jeong, H.Y., S. Obaidat, M., Yen, N.Y., Park, J.J. *Advances in Computer Science and its Applications*. (Vol. 279, pp. 829-836), Springer Berlin Heidelberg.
47. HousingMaps: http://www.housingmaps.com/. Last access on Nov. 26th, 2015
48. IFTTT: https://ifttt.com/. Last access on Dec 3th, 2015
49. Jain, P., Hitzler, P., Sheth, A.P., Verma, K., Yeh, P.Z. (2010). Ontology alignment for linked open data. In *The Semantic Web – ISWC 2010*. (Vol. pp. 402-417), Springer.
50. Jenkins, H.: Confronting the challenges of participatory culture: Media education for the 21st century. Mit Press (2009)
51. Jhingran, A. (2006). Enterprise information mashups: integrating information, simply. In: *Proc. of International Conference on Very Large Data Bases (VLDB '06)*. 12 - 15 September. pp. 3-4. VLDB Endowment,
52. Jung, J. (2012). ContextGrid: A contextual mashup-based collaborative browsing system. In *Information Systems Frontiers* 14(4)**,** 953-961
53. Kemp, J., Van Gelderen, T. (1996). Co-discovery exploration: an informal method for the iterative design of consumer products. In *Usability evaluation in industry* 139-146
54. Krug, M., Wiedemann, F., Gaedke, M. (2014). SmartComposition: A Component-Based Approach for Creating Multi-screen Mashups. In Casteleyn, S., Rossi, G., Winckler, M. *Web Engineering - ICWE 2014*. (Vol. 8541, pp. 236-253), Springer International Publishing.
55. Krummenacher, R., Norton, B., Simperl, E., Pedrinaci, C. (2009). SOA4All: Enabling Web-scale Service Economies. In: *Proc. of International Conference on Semantic Computing (ICSC '09)*. Berkeley, CA (USA). 14-16 September. pp. 535-542. IEEE Computer Society, 1679938.

56. Lewis, J., Sauro, J. (2009). The Factor Structure of the System Usability Scale. In Kurosu, M. *Human Centered Design - HCD 2009*. (Vol. LNCS 5619, pp. 94-103), Springer Berlin Heidelberg.
57. Loton, T.: Introduction to Microsoft Popfly, No Programming Required. Lotontech Limited (2008)
58. Lucero, A., Holopainen, J., Jokela, T. (2011). Pass-them-around: collaborative use of mobile phones for photo sharing. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. Vancouver, BC (Canada). 7 - 12 May. pp. 1787-1796. ACM, New York, NY, USA.
59. Lucero, A., Keränen, J., Korhonen, H. (2010). Collaborative use of mobile phones for brainstorming. In: *Proc. of International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '10)*. Lisbon, Portugal. 7 - 10 September. pp. 337-340. ACM, New York, NY, USA.
60. Marquardt, N., Hinckley, K., Greenberg, S. (2012). Cross-device interaction via micro-mobility and f-formations. In: *Proc. of ACM symposium on User Interface Software and Technology (UIST '12)*. Cambridge, Massachusetts (USA). 7-10 October. pp. 13-22. ACM, New York, NY, USA.
61. Matera, M., Picozzi, M., Pini, M., Tonazzo, M. (2013). PEUDOM: A mashup platform for the end user development of common information spaces. In *Web Engineering - ICWE 2013*. (Vol. pp. 494-497), Springer.
62. Maximilien, E.M., Wilkinson, H., Desai, N., Tai, S.: A domain-specific language for web apis and services mashups. Springer (2007)
63. Meteor: Meteor, https://www.meteor.com/. Last access on Sept. 25th, 2015
64. Morris, M.R., Danielescu, A., Drucker, S., Fisher, D., Lee, B., schraefel, m.c., Wobbrock, J.O. (2014). Reducing legacy bias in gesture elicitation studies. In *interactions* 21(3)**,** 40-45
65. Namoun, A., Nestler, T., De Angeli, A. (2010). Conceptual and Usability Issues in the Composable Web of Software Services. In Daniel, F., Facca, F. *Current Trends in Web Engineering - ICWE 2010* (Vol. 6385, pp. 396-407), Springer Berlin Heidelberg.
66. Namoun, A., Wajid, U., Mehandjiev, N. (2010). Service Composition for Everyone: A Study of Risks and Benefits. In Dan, A., Gittler, F., Toumani, F. *Service-Oriented Computing - ICSOC/ServiceWave 2009 Workshops*. (Vol. 6275, pp. 550-559), Springer Berlin Heidelberg.
67. Nardi, B.A.: A small matter of programming: perspectives on end user computing. MIT Press (1993)
68. Netvibes: https://www.netvibes.com/. Last access on Nov 26th, 2015
69. Paredes-Valverde, M.A., Alor-Hernández, G., Rodríguez-González, A., Valencia-García, R., Jiménez-Domingo, E. (2015). A systematic review of tools, languages, and methodologies for mashup development. In *Software: Practice and Experience* 45(3)**,** 365-397
70. Pavel, S., Euzenat, J. (2013). Ontology Matching: State of the Art and Future Challenges. In *IEEE Trans. on Knowl. and Data Eng.* 25(1)**,** 158-176
71. Porter, J.: Designing for the social web. Peachpit Press (2010)
72. Preece, J., Sharp, H., Rogers, Y.: Interaction Design-beyond human-computer interaction. John Wiley & Sons (2015)

73. Presto, J.: http://mdc.jackbe.com/prestodocs/v3.7/raql/cacheStore.html. Last access on Nov 26th, 2015
74. Pruett, M.: Yahoo! pipes. O'Reilly (2007)
75. Rädle, R., Jetter, H.-C., Marquardt, N., Reiterer, H., Rogers, Y. (2014). HuddleLamp: Spatially-Aware Mobile Displays for Ad-hoc Around-the-Table Collaboration. In: *Proc. of ACM International Conference on Interactive Tabletops and Surfaces (ITS '14)*. Dresden (Germany). 16-19 November. pp. 45-54. ACM, New York, NY, USA.
76. Rädle, R., Jetter, H.-C., Schreiner, M., Lu, Z., Reiterer, H., Rogers, Y. (2015). Spatially-aware or Spatially-agnostic?: Elicitation and Evaluation of User-Defined Cross-Device Interactions. In: *Proc. of ACM Conference on Human Factors in Computing Systems (CHI '15)*. Seoul (Republic of Korea). April 18 - 23. pp. 3913-3922. ACM, New York, NY, USA.
77. Reetz, A., Gutwin, C., Stach, T., Nacenta, M., Subramanian, S. (2006). Superflick: a natural and efficient technique for long-distance object placement on digital tables. In: *Proc. of Graphics Interface 2006 (GI '06)*. Quebec (Canada). June 7 - 9. pp. 163-170. Canadian Information Processing Society, Toronto, Ont., Canada, Canada.
78. Reeve, L., Han, H. (2005). Survey of semantic annotation platforms. In: *Proc. of ACM symposium on Applied Computing (SAC '05)*. Santa Fe (New Mexico). 13 -17 March. pp. 1634-1638. ACM, 1067049.
79. Spillner, J., Feldmann, M., Braun, I., Springer, T., Schill, A. (2008). Ad-Hoc Usage of Web Services with Dynvoker. In Mähönen, P., Pohl, K., Priol, T. *Towards a Service-Based Internet - ServiceWave 2008*. (Vol. 5377, pp. 208-219), Springer Berlin Heidelberg.
80. Tanimoto, S.L. (1990). VIVA: A visual language for image processing. In *Journal of Visual Languages & Computing* 1(2)**,** 127-139
81. Technology, I.E.: http://nodered.org/. Last access on Nov 26th, 2015
82. Voida, S., Podlaseck, M., Kjeldsen, R., Pinhanez, C. (2005). A study on the manipulation of 2D objects in a projector/camera-based augmented reality environment. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. Portland, Oregon (USA). 02 - 07 April. pp. 611-620. ACM, New York, NY, USA.
83. Wacharamanotham, C., Subramanian, K., Völkel, S.T., Borchers, J. (2015). Statsplorer: Guiding Novices in Statistical Analysis. In: *Proc. of ACM Conference on Human Factors in Computing Systems (CHI '15)*. Seoul (Republic of Korea). 18 - 23 April. pp. 2693-2702. ACM, New York, NY, USA.
84. Wajid, U., Namoun, A., Mehandjiev, N. (2011). Alternative Representations for End User Composition of Service-Based Systems. In Costabile, M.F., Dittrich, Y., Fischer, G., Piccinno, A. *End-User Development - Is-EUD 2011*. (Vol. 6654, pp. 53-66), Springer Berlin Heidelberg.
85. Weiser, M. (1991). The computer for the 21st century. In *Scientific american* 265(3)**,** 94-104
86. White, R.W., Roth, R.A. (2009). Exploratory search: Beyond the query-response paradigm. In *Synthesis Lectures on Information Concepts, Retrieval, and Services* 1(1)**,** 1-98
87. Wong, J., Hong, J.I. (2007). Making mashups with marmite: towards end-user programming for the web. In: *Proc. of SIGCHI Conference on Hu-*

man Factors in Computing Systems (CHI '07). San Jose, California (USA). 28 April - 3 May. pp. 1435-1444. ACM, 1240842.

88. Wright, S., Bakmand-Mikalski, D., bin Rais, R., Bishop, D., Eddinger, M., Farnhill, B., Hild, E., Krause, J., Loriot, C., Malik, S. (2011). Designing Mashups with Excel and Visio. In *Expert SharePoint 2010 Practices*. (Vol. pp. 513-539), Springer.
89. Yang, J., Wigdor, D. (2014). Panelrama: enabling easy specification of cross-device web applications. In: *Proc. of SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. Toronto, Ontario (Canada). 26 April - 1 May. pp. 2783-2792. ACM, New York, NY, USA.
90. Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M. (2007). A framework for rapid integration of presentation components. In: *Proc. of International Conference on World Wide Web (WWW '07)*. Banff, Alberta (Canada). May 8-12. pp. 923-932. ACM,
91. Zang, N., Rosson, M.B. (2008). What's in a mashup? And why? Studying the perceptions of web-active end users. In: *Proc. of IEEE Symposium on Visual Languages and Human-Centric Computing (VL-HCC '08)*. Herrsching, Ammersee (Germany). September 15 - 19. pp. 31-38. IEEE Computer Society, 1550043.

Giuseppe Desolda's publications during 2013-2015

Papers in International Journals

- Ardito C., Buono P., Costabile M. F., and **Desolda G.** (2015). Interaction with large displays: a survey. *ACM Computing Survey* 47, 3, 1-38.
- Lanzilotti, R., Ardito, C., Costabile, M.F., De Angeli, A., **Desolda, G.** (2015). Pupils' Collaboration around a Large Display. In *Journal of Visual Languages & Computing* 31, B, 206-214.
- Ardito C., Bottoni P., Costabile M. F., **Desolda G.**, Matera M., and Picozzi M. (2014). Creation and Use of Service-based Distributed Interactive Workspaces. *Journal of Visual Languages & Computing* 25, 6, 717-26.
- Ardito C., Costabile M. F., **Desolda G.**, Lanzilotti R., Matera M., Piccinno A., and Picozzi M. (2014). User-Driven Visual Composition of Service-Based Interactive Spaces. *Journal of Visual Languages & Computing* 25, 4, 278-96.
- Ardito C., Lanzilotti R., Costabile M. F., and **Desolda G.** (2013). Integrating traditional learning and games on large displays: an experimental study. *Educational Technology & Society* 16, 1, 44-56.

Chapters in International Collections

- **Desolda, G.**, Ardito, C., Matera, M. (2015). EFESTO: A platform for the End-User Development of Interactive Workspaces for Data Exploration. In Daniel, F., Pautasso, C. *Rapid Mashup Development Tools - Rapid Mashup Challenge in ICWE 2015*. (Vol. 591, pp. 63 - 81), Berlin Heidelberg, Springer Verlag.
- **Desolda, G.** (2015). Enhancing Workspace Composition by Exploiting Linked Open Data as a Polymorphic Data Source. In Damiani, E., Howlett, R.J., Jain, L.C., Gallo, L., De Pietro, G. *Intelligent Interactive Multimedia Systems and Services - KES-IIMSS 2015*. (Vol. 40, pp. 97-108), Berlin Heidelberg, Springer International Publishing.
- **Desolda, G.**, Jetter, H.-C. (2015). Spatial Awareness in Mobile Devices to Compose Data Source: A Utilization Study. In Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. *End-User Development - Is-EUD 2015*. (Vol. LNCS 9083, pp. 291-294), Berlin Heidelberg, Springer International Publishing.
- Ardito, C., Costabile, M.F., **Desolda, G.**, Latzina, M., Matera, M. (2015). Hands-on Actionable Mashups. In Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. *End-User Development - Is-EUD 2015*. (Vol. LCNS 9083, pp. 295-298), Berlin Heidelberg, Springer Verlag.
- Ardito, C., Costabile, M.F., **Desolda, G.**, Latzina, M., Matera, M. (2015). Making Mashups Actionable Through Elastic Design Principles. In Díaz, P.,

Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. *End-User Development - Is-EUD 2015*. (Vol. LCNS 9083, pp. 236-241), Berlin Heidelberg, Springer Verlag.

- **Desolda, G.** (2014). Using semantic techniques to improve service composition by end users. In Casteleyn, S., Rossi, G., Winckler, M. *Web Engineering - ICWE 2014*. (Vol. LCNS 8541, pp. 567-570), Berlin Heidelberg, Springer International Publishing.
- Ardito, C., Bottoni, P., Costabile, M.F., **Desolda, G.**, Matera, M., Piccinno, A., Picozzi, M. (2013). Enabling End Users to Create, Annotate and Share Personal Information Spaces. In Dittrich, Y., Burnett, M., Mørch, A., Redmiles, D. *End-User Development - Is-EUD 2013*. (Vol. LCNS 7897, pp. 40-55), Berlin Heidelberg, Springer Verlag.

Proceedings of International Conferences

- Ardito, C., **Desolda, G.**, Matera, M. (2015). Fostering Innovation through End-User Development: a Mashup-based Approach. In: *Proc. of International Forum on Knowledge Asset Dynamics (IFKAD '15)*. Bari (Itay). 10-12 June 2015. pp. 1454 - 1464.
- **Desolda, G.**, Ardito, C., Matera, M., Piccinno, A. (2015). Mashing-up smart things: a meta-design approach. In: *Proc. of Workshop on End User Development in the Internet of Things Era (EUDITE '15) - CHI '15 EA*. Seoul (Korea). April 19, 2015. pp. 33 - 36. ACM, New York, NY, USA.
- Ardito, C., Costabile, M.F., **Desolda, G.**, Lanzilotti, R., Matera, M., Picozzi, M. (2014). Visual Composition of Data Sources by End-Users. In: *Proc. of International Working Conference on Advanced Visual Interfaces (AVI '14)*. Como (Italy). 28-30 May pp. 257-260. ACM, New York, NY, USA.
- **Desolda, G.** (2014). A platform for enabling end users to compose data and services. In: *Proc. of Demo Session of the International Working Conference on Advanced Visual Interfaces (AVI '14)*. Como, Italy. 28-30 May, 2014. pp. 1-4. ACM,
- Buono, P., **Desolda, G.** (2014). Visualizing collaborative traces in distributed teams. In: *Proc. of International Working Conference on Advanced Visual Interfaces (AVI '14)*. Como, Italy. May 28-30, 2014. pp. 343-344. ACM, New York, USA.
- Ardito, C., Costabile, M.F., **Desolda, G.**, Lanzilotti, R., Matera, M., Piccinno, A., Picozzi, M. (2013). Personal information spaces in the context of visits to archaeological parks. In: *Proc. of Biannual Conference of the Italian Chapter of SIGCHI (CHItaly '13)*. Trento, Italy. September 17-19, 2013. pp. 1-4. ACM, New York, NY, USA.
- **Desolda, G.** (2013). Empowering End Users to Create Interactive Workspaces by Service Composition. In: *Proc. of Doctoral Consortium of the Biannual Conference of the Italian Chapter of SIGCHI (CHItaly '13)*. Trento, Italy. September 16, 2013. pp. 1-4.

- Ardito, C., **Desolda, G.**, Lanzillotti, R. (2013). Playing on large displays to foster children's interest in archaeology. In: *Proc. of International Conference on Distributed Multimedia Systems (DMS '13)*. Brighton (UK). August 8-10, 2013. pp. 79-84. Knowledge Systems Institute, Skokie, Illinois, USA.
- Buono, P., **Desolda, G.**, Lanzillotti, R. (2013). Scene extraction from telementored surgery videos. In: *Proc. of International Conference on Distributed Multimedia Systems (DMS '13)*. Brighton (UK). August 8-10, 2013. pp. Knowledge Systems Institute, Skokie, Illinois, USA.
- Ardito, C., Costabile, M.F., **Desolda, G.**, Lanzilotti, R., Matera, M. (2013). Combining Composition Technologies and EUD to Enhance Visitors' Experience at Cultural Heritage Sites. In: *Proc. of Conference on User Modeling, Adaptation and Personalization (UMAP '13)*. Rome (Italy), June 10-14, 2013. pp. 1-9.

Proceedings of National Conferences

- Buono, P., **Desolda, G**., Lanzilotti, R. (2013). A telementoring system for supporting laparoscopic surgeries (poster session). In: *Proc. of Annual Conference of the Associazione Italiana per il Calcolo Automatico (AICA '13)*. Fisciano (SA), Italy. September 18-20, 2013.