Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com

End-user composition of interactive applications through actionable UI components

Giuseppe Desolda[a, *], Carmelo Ardito[a], Maria Francesca Costabile[a], Maristella Matera[b]

[a] Dipartimento di Informatica, Università degli Studi di Bari Aldo Moro, Via Orabona, 4 – 70125 – Bari, Italy
[b] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, P.zza L. da Vinci, 32 – 201233 – Milano

ARTICLE INFO

Keywords:
Web service composition
Mashup model
Utilization study

ABSTRACT

Developing interactive systems to access and manipulate data is a very tough task. In particular, the development of user interfaces (UIs) is one of the most time-consuming activities in the software lifecycle. This is even more demanding when data have to be retrieved by accessing flexibly different online resources. Indeed, software development is moving more and more toward composite applications that aggregate on the fly specific Web services and APIs. In this article, we present a mashup model that describes the integration, at the *presentation layer*, of UI components. The goal is to allow non-technical end users to visualize and manipulate (i.e., to perform actions on) the data displayed by the components, which thus become *actionable UI components*. This article shows how the model has guided the development of a mashup platform through which non-technical end users can create component-based interactive workspaces via the aggregation and manipulation of data fetched from distributed online resources. Due to the abundance of online data sources, facilitating the creation of such interactive workspaces is a very relevant need that emerges in different contexts. A utilization study has been performed in order to assess the benefits of the proposed model and of the Actionable UI Components; participants were required to perform real tasks using the mashup platform. The study results are reported and discussed.

1. Introduction

The development of user interfaces (UIs) is one of the most time-consuming activities in the creation of interactive systems. The need for proper reuse mechanisms for building UIs has become evident in the last years, especially as software development is moving more and more toward component-based applications [14]. A considerable number of resources are also available online. Thus, easy and effective mechanisms to create UIs on top of the offered data are required. In this article, we propose a mashup model that enables the integration at the *presentation layer* of “Actionable UI components”. These are components equipped with both data visualization templates and a proper logic consisting of functions to manipulate the visualized data. The model clarifies the main ingredients that are needed to design a platform that enables non-technical end-users to create component-based *interactive workspaces* by aggregating and manipulating data available on distributed online resources. The main goal is to highlight the features that can lead to frameworks able to reduce the end-users' effort for the development of *interactive workspaces* [5], by maximizing the reuse of UI components.

In our approach, UI components not only constitute “pieces” of UIs that can be assembled into a unified workspace. Each single component can also provide views over the huge quantity of data exposed by online Web services and APIs or in any data source, even personal or locally provided sources. With respect to the definition of UI components given in [10,42], we promote the notion of *Actionable UI components*, which introduce varying functions to allow end users to manipulate the contained data.

Our approach is positioned in a research context related to facilitating the access to online data sources through visual user interfaces, a problem that has been attracting the attention of several researchers in recent years [29,36]. An ever-increasing number of resources is nowadays available on the Web and provides content and functions in different formats through programmatic interfaces. The efforts of many research projects have thus focused on letting laypeople, i.e., users without expertise in programming (also called non-technical users), access and reuse the available content [2,24]. In this respect, the reuse of easily programmable UI components is a step towards the provision of en-

* Corresponding author.
Email addresses: giuseppe.desolda@uniba.it (G. Desolda); carmelo.ardito@uniba.it (C. Ardito); maria.costabile@uniba.it (M.F. Costabile); maristella.matera@polimi.it (M. Matera)

https://doi.org/10.1016/j.jvlc.2017.08.004
Received 15 January 2017; Received in revised form 22 June 2017; Accepted 13 August 2017
Available online xxx
1045-926/ © 2017.

vironments facilitating the End-User Development (EUD) of service-based interactive workspaces [5]. In general, EUD refers to allowing end users to modify and even create software artifacts while using an interactive system [19,31]. EUD activities range from simple parameter setting to the integration of pre-packaged components, up to extending the system by developing new components.

Reusing is also typical of Web mashups [14], a class of applications that emerged in the last decade, which can be created by integrating components at any of the application stack layers (presentation, business logics and data). The term mashup was originally coined in 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. The real novelty introduced by Web mashups is the possibility to synchronize components at the presentation layer by considering elements of their UI, for example, by means of event-driven composition techniques. Thanks to the possibility of reusing and synchronizing ready-to-use UI components, the mashup has resulted in an effective paradigm to let end users, even non-experts in technology, compose their interactive Web applications.

Over the last years, we have been working extensively on a mashup platform called EFESTO that, by exploiting end-user development principles, addresses the creation of component-based interactive workspaces by non-technical end users, via the aggregation and manipulation of data fetched from distributed online resources [5,6,17]. This platform also enables the collaborative creation and use of distributed interactive workspaces [4]. The platform prototype keeps improving on various aspects, based on field studies performed with real users who reveal new requirements and features that are useful to foster the adoption of mashup platforms in people's daily activities. Based on these experiences, in which we observed people creating their interactive applications easily, in this article we aim to stress the importance of this type of platforms as tools for the rapid creation of interactive applications enabling the access to Web services and APIs. In particular, the main contribution of this article is a model for UI component mashup that other designers and developers can adopt, in order to develop mashup platforms that permit to easily compose interactive workspaces whose logic is distributed across different synchronized components.

The paper is organized as follows. Section 2 illustrates the main functionality offered by the EFESTO platform for the creation of interactive workspaces. Section 3 describes the proposed mashup model; in particular, it highlights how the modus operandi supported by EFESTO is made possible thanks to some abstractions, and, in particular, to the notion of *actionable* UI *components*, around which the whole platform design has been conceived. We specifically stress how the adoption of such conceptual elements leads to the notion of a distributed User Interface as an interactive artefact that can be assembled according to lightweight technologies and that leverages on the logics of self-contained actionable UI components. Section 4 discusses the Domain-Specific Languages (DSLs) we introduced to describe the main elements of a mashup platform and that can guide the dynamic instantiation and execution of the distributed UIs. Section 5 complements Section 3 by providing some technical details on how the model elements are implemented in the EFESTO platform architecture. Section 6 reports the results of a utilization study in which participants performed reals tasks by using the EFESTO platform. On the basis of the related literature, Section 7 presents some dimensions for classifying mashup tools and discusses how EFESTO is characterized with respect to such dimensions. Section 8 finally concludes the article and outlines future work.

2. The EFESTO platform

This section describes the most important features of our mashup platform, EFESTO, by showing how it is used to create a mashup [18]. The EFESTO composition paradigm extensively exploits Actionable UI Components; therefore we illustrate it also showing how such components support the creation of interactive workspaces. The main features described in this section (highlighted in **bold)** will be then formalized in the model reported in Section 3.

2.1. Operations for mashing-up data sources

In order to describe how EFESTO works, we refer to a scenario in which a user, Tyrion, exploits the platform to compose an interactive workspace that retrieves some needed information from distributed resources and visualizes them. Tyrion does not know how to use programming languages; more in general, he is not familiar with Computer Science technical concepts.

Tyrion is going to organize his summer holidays during which he would like to attend a concert. He uses EFESTO to create an application (a mashup) that retrieves and integrates information about music events. First, Tyrion looks for pertinent services among those registered and organized by category (e.g., videos, photos, music, social) in the platform. A wizard procedure guides him to make a selection. Tyrion selects *SongKick*, a service that provides information on music events of a specific singer. Afterward, Tyrion has to select how to display the retrieved data by choosing a visualization template (**UI Template**) among the ones available in the platform. Tyrion actually selects a *map template*, since he wants to visualize the retrieved music events geo-localized in a map.

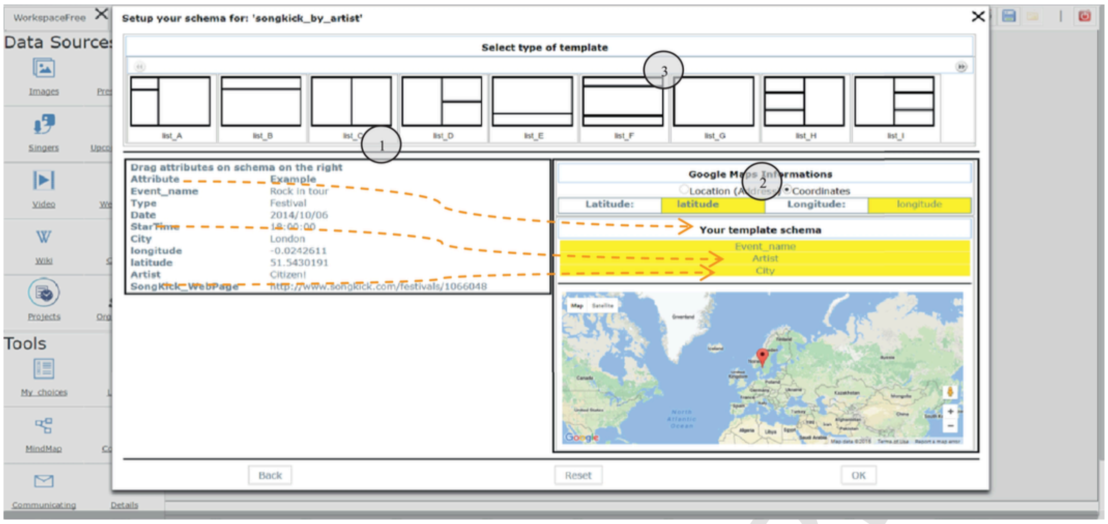
Among the different data attributes retrieved by SongKick, Tyrion has to select those he is interested in. All SongKick data attributes are visualized in a panel on the left (see Fig. 1, circle 1). Thus, Tyrion drags & drops the *latitude* and *longitude* SongKick attributes into the respective fields (called **Visual Renderers** [10]) of the map UI template (Fig. 1, circle 2). Additional details about a music event, namely *Event_name, Artist* and *City*, are visualized in a *table* template with three rows and one column (Fig. 1, circle 3). He selects the three attributes from the left panel (Fig. 1, circle 1) and drops them in the visual renderers of the UI template (highlighted in yellow in Fig. 1, circle 2).

After performing this visual mapping, Tyrion saves the mashup with the name "Upcoming events by artist name" . From now on, this mashup is a **UI Component** in the user workspace, which is immediately executed in the Web browser and represented as a map, as shown in the central panel in Fig. 2. By typing "Maroon 5″ in the search box (thus formulating a **query**), the **result set** of forthcoming events of this singer is visualized as pins on the map. By clicking on a pin representing a music event, details of that event (i.e., the attributes *Event_name, Artist* and *City*) are shown.

Tyrion can later update the created mashup by integrating data coming from other **data sources** through **union** and **join** data mashup **operations** [17]. Since a non-technical user is not familiar with these operations, EFESTO offers wizard procedures and drag&drop mechanisms to express how data have to be integrated. For example, let's suppose that Tyrion wants to retrieve additional music events from *Eventful* (another service retrieving music events), i.e., he need to perform a union operation between the SongKick and the Eventful result sets.

To perform this operation, Tyrion starts from the SongKick UI component previously created, clicks on the gearwheel icon in the toolbar (circle 1 in Fig. 2) and chooses the "Add results from new source" menu item. A wizard procedure now guides Tyrion in choosing the new service, Eventful, and in performing a new visual mapping between the Eventful attributes and the UI template already used in the previous mashup. If queried with an artist name, the newly created mashup (UI Component) now visualizes results gathered both from the SongKick and Eventful services.

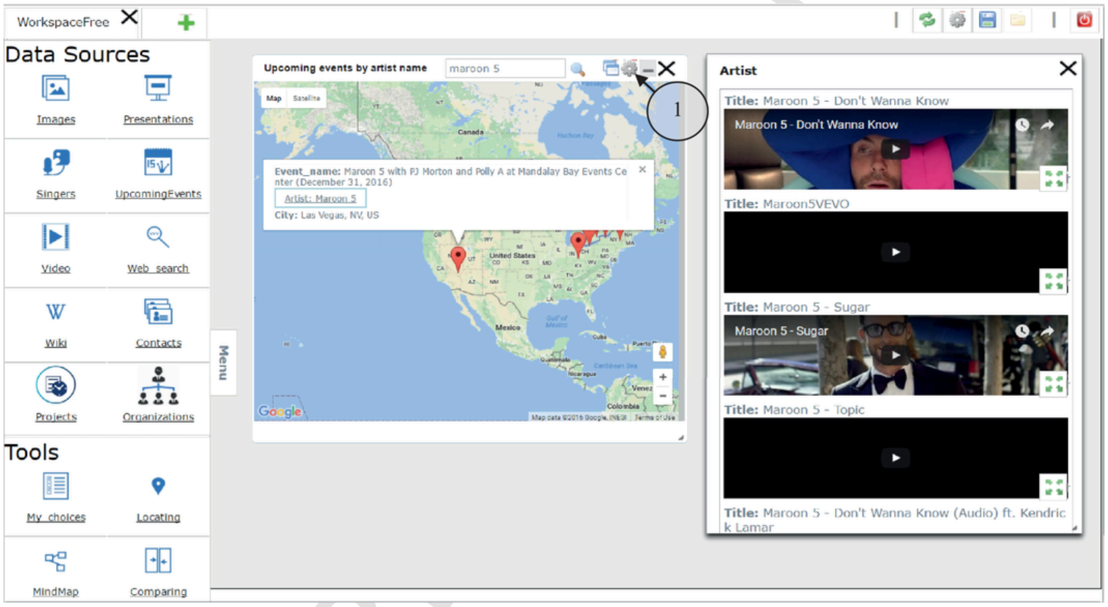
Another data integration operation available in EFESTO is the join of different sources; it is useful for merging a mashup already in place



Drag attributes on schema on the right

Attribute	Example
Event_name	Rock in tour
Type	Festival
Date	2014/10/06
StarTime	18:00:00
City	London
longitude	-0.0242611
latitude	51.5430191
Artist	Citizen!
SongKick_WebPage	http://www.songkick.com/festivals/1066048

Fig. 1. Screenshot of EFESTO platform, referring to the mapping between some SongKick attributes (circle 1) and the fields of the map template (circle 2).



Event_name: Maroon 5 with PJ Morton and Polly A at Mandalay Bay Events Center (December 31, 2016)
Artist: Maroon 5
City: Las Vegas, NV, US

Fig. 2. UI component originated from SongKick data source visualized as a map and joined with YouTube to show artist video.

with new data available in other services. For example, Tyrion now would like to show videos that can be retrieved from *YouTube*. Technically, this operation is a join between the SongKick *artist* attribute and YouTube *video title* attribute. Tyrion can perform this operation through a new wizard procedure that guides him while choosing (a) the service attribute to be used as join attribute (*artist* in this example), (b) the new data source (YouTube) and (c) how to visualize the YouTube results. Once the join operation is completed, when clicking on the artist name in the map info window, another window visualizes the YouTube videos related to the artist, as shown in the right panel of Fig. 2,

Another operation available in EFESTO is the *change of visualization* for a given UI component. For example, during the interaction with SongKick, Tyrion can decide to switch from the *map* UI template to the *list* UI template (see the result in Fig. 3, circle 1). A wizard procedure guides Tyrion to (a) choose a new UI template (*list* in this case), and (b) perform a visual mapping between the SongKick attributes and the UI template, as already described with reference to Fig. 1.

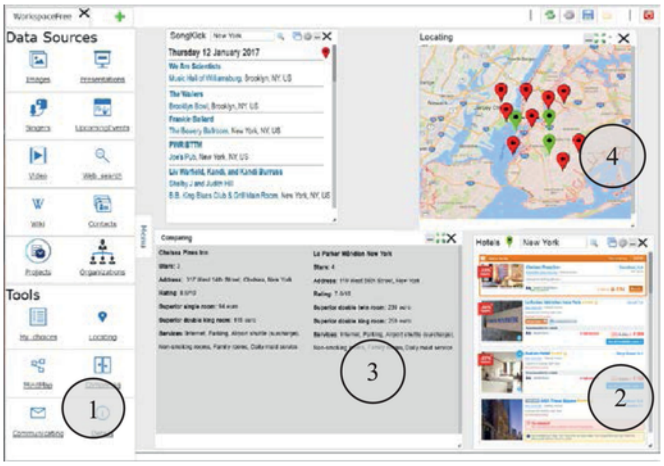


Fig. 3. Use of some tools available in EFESTO to manipulate mashup data.

2.2. A polymorphic data source

Despite the wide availability of data sources and composition operations, sometimes users can still encounter difficulties while trying to accommodate different needs and preferences. Let us suppose, for example, that during the interaction with EFESTO Tyrion wants to get some details about the artists of the music events (e.g., genre, starting year of activity and artist photo) that cannot be retrieved through the services already registered in the platform. In order to overcome this drawback, EFESTO provides a **polymorphic data source** that exploits the wide availability of information published in the Linked Open Data (LOD) cloud. It is called polymorphic because, when it is composed with another source *S*, it can dynamically adapt the set of exposed attributes depending on the source *S*, so that a “navigation” from the source *S* to the polymorphic data source is possible [16].

EFESTO exploits DBpedia as LOD cloud as it provides a vast amount of information. Thus, Tyrion can join the SongKick artist attribute with a DBpedia-based polymorphic data source. The platform now shows a list of attributes related to the musical artist class (available in the DBpedia ontology), and Tyrion enriches the current UI Component with the attributes *genre, starting year of activity* and *artist photo*. Henceforward, Tyrion can find a list of upcoming events and also visualize artist's information when clicking on the artist's name.

2.3. Actionable UI component

Some field studies that we conducted in the past to validate our prototypes [4,5] revealed that mashups generally lack data manipulation functions that end users would like to exploit in order to “act” on the extracted contents, e.g., functions that allow to perform tasks such as collecting&saving favourites, comparing items, plotting data items on a map, inspecting full content details, organizing items in a mind map in order to highlight relationships. In this section, we remark another very innovative feature of EFESTO: it offers tools that enable specific tasks, allowing users to manipulate the information in a novel fashion, i.e., without being constrained to pre-defined operation flows typical of pre-packaged applications.

In order to perform more specific and complex sense-making tasks, a set of Tools is available in the left-panel of the workspace (see Fig. 3, circle 1). These Tools are added to the workspace by clicking the corresponding icon. Let us describe an example of their usage with reference to our scenario. Tyrion is looking for hotels in New York located nearby the places where upcoming musical events will be held. He is more interested in finding a good hotel and then looking for possible musical events to attend. Therefore, first he adds the *Hotel* data source into his workspace (see Fig. 3, circle 2) and then performs a search by typing “New York” in the *Hotel* search bar. After including the *Comparing* tool in the workspace, Tyrion drags&drops inside it the first five hotels from the *Hotel* UI component. The *Comparing* tool supports Tyrion in the identification of the most convenient hotels, which are now represented as cards providing further details, such as average price, services and category (see Fig. 3, circle 3). Afterwards, he moves three hotels from the *Comparing* tool inside the *Locating* tool (Fig. 3, circle 4) in order to visualize them as pins on the map. Finally, Tyrion performs a search on the *SongKick* data source by using “New York” as keyword and then moves all the results, i.e., the upcoming musical events, inside the *Locating* container. The map now shows pins indicating both the selected hotels (red pins) and the upcoming musical events in New York (green pins). Tyrion can now easily identify which musical events are close to the hotels he has previously chosen.

As shown in the previous example, the EFESTO tools allow users to interact with information within dedicated *actionable UI components*, which enable specific tasks. Such flexible environments are based on the model presented in [7] that promotes easy transitions of information between different contexts. This model implements some of the principles defined within the Transformative User eXperience (TUX) framework [8,30]. The goal of TUX is to overcome common application boundaries enabling user interaction with information within dedicated, contextual task environments called *task containers*. Depending on their situational needs, users move data items along different task containers; the container semantics then progressively transform data. Users are therefore empowered to be more active upon the retrieved information.

3. Model for actionable UI mashups

The main contribution of this article is a model highlighting the most important abstractions that allowed us to define the composition paradigm described in the previous section. With respect to other composition paradigms already presented in literature, the one we illustrate in this article has been designed and validated through a series of user studies that have also allowed us to focus on the main elements that can promote the EUD of interactive workspaces. Therefore, the goal of the model described in the rest of this section is to guide designers and software engineers in the development of EUD environments able to support non-programmers to build UIs by reusing and synchronizing the logic of different pieces of UI. The model also leads to the notion of *distributed UIs*, built by aggregating different components, each one independent from the others but with the intrinsic capability to be synchronized with the others.

The proposed model refines and extends the one presented in [10]. It has been iteratively refined by adding further components starting from requirements we gathered during our research, namely: i) a different way to integrate service data by means of *join* and *union* operations for data mashup; ii) the *Actionable UI Components* that implement some Transformative User eXperience principles [8,30]; iii) the polymorphic data sources to access LOD. The new model is depicted in Fig. 5. In the following, we report the definitions of the most salient concepts that contribute to the notion of distributed UIs.

Definition 1. UI component

It is the core of the model since it represents the main modularization object the user can exploit to retrieve and compose data extracted from services. According to [42], a UI component is a JavaScript/HTML stand-alone application that can be instantiated and run inside any Web browser and that, unlike Web services or data sources, is equipped with a UI that enables the interaction with the underlying service via standard HTML. In our approach, a UI component also allows the interaction with services data and functions thanks to its own UI (see Fig. 4). More specifically, it supplies a view according to specific UI Templates (see Definition 2) over one or more services whose data can be composed by means of data mashup operations. In addition, two or more UI components can also be synchronized according to an event-driven paradigm: each of them can implement a set *E* of events that the user can trigger during the interaction with its user interface, and a set *O* of operations activated when events are generated by others UI components.

Definition 2. UI template

It plays two fundamental roles inside the UI component: first, it guides the users in materializing abstract data sources by means of a mapping between the data source output attributes and the UI template visual renderers; second, at runtime, it displays the data source according to the user mapping. A *UI Template* can be represented as the triple

$$uit = <type, VR>$$

where *type* is the template (e.g., list, map, chart) selected by the

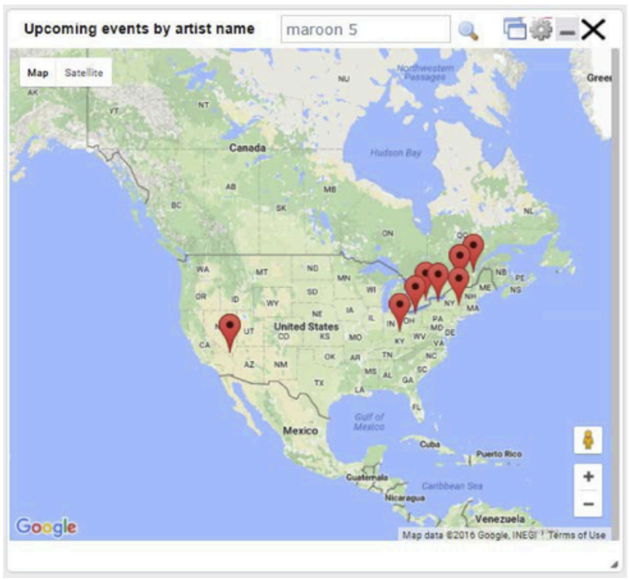


Fig. 4. Example of UI component that shows musical events on Google Maps.

user while *VR* is a set of *visual renderers*, i.e., UI elements that act as receptors of data attributes.

Definition 3. Actionable UI component (auic)
In addition to visualizing Web service data, as managed by UI components, *auic* provides containers that encapsulate UI components to supply task-related functions for manipulation and transformation of data items retrieved from a source [7].

An *auic* can be defined as a pair:

$$auic = <TF,\ uit>$$

where TF is the set of *functions* for manipulation and transformation of data, while uit is a UI template used to visualize data to facilitate the user's current task.

Definition 4. Event-driven coupling
It is a synchronization mechanism among two UI components that the users define according to an event-driven, publish-subscribe integration logic [42]. In particular, the users define that, when an event is triggered on a UI component, an operation will be performed by another UI component. This enables reusing the logic of single UI components, still being able to introduce some new behaviour for the composite UIs. More in general, given two UI Components uic_i and uic_j, a coupling is a pair:

$$c = <uic_i\,(<output>)\,,\ uic_j\,(<input>)\,>$$

Definition 5. Layout template
It is an abstract representation of the workspace layout defining the visual organization of the UICs included in the interactive workspace under construction. For example, the UI components can be freely located or can be constrained to a grid schema, where in each cell only one UI Component can be placed.

Definition 6. Actionable UI mashup
An Actionable UI Mashup is the final interactive application built by the end users by means of the integration of different UI components within a workspace. It can be formalized as the tuple:

$$UI\_Mashup = <UIC,\ AUIC,\ C,\ LT>,$$

where UIC is the set of UI Components integrated into the workspace, AUIC is the set of Actionable UI Components to manipulate data extracted from UIC, C is the set of couplings the users established among UIC and LT is the layout template chosen to arrange the UIC within the workspace.

The following definitions are reported to clarify how actionable UI components are instantiated by means of data extracted from data sources.

Definition 7. Data component
It is an abstract representation of the resource that can be used to retrieve data. In particular, a data source *dc* is a triplet:

$$dc = <t,\ I,\ A>$$

where *t* indicates the type of resource, for example *REST Data Source* or *Polymorphic Data Source* in our model, *I* indicates the set of input parameters to query the resources, *A* indicates the set of output attributes. Data can be retrieved from data sources and aggregated through the following operations:

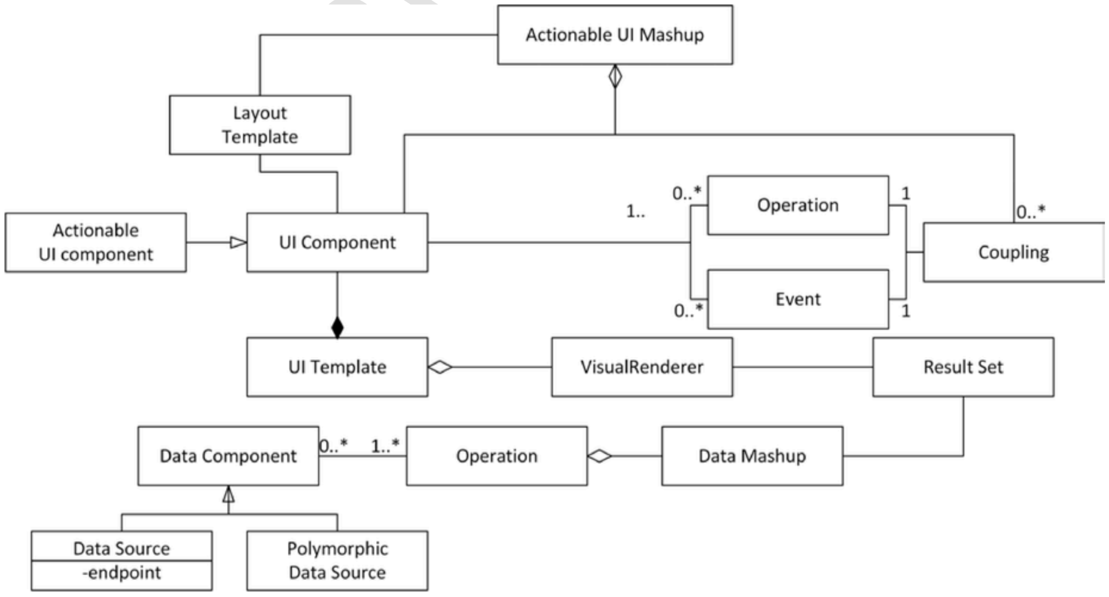


Fig. 5. The mashup model.

Definition 8.a. Selection

Given a data component *dc*, a selection is a unary operator defined as:

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

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

Definition 8.b. Join

This operator allows one to establish connections among data in different data components. Given a couple of data components dc_i and dc_j, a *Join* is a binary operator on dc_i and dc_j defined as:

$$dc_i \bowtie_F dc_j = \sigma_F\ (dc_i\ x\ dc_j)$$

The result of this operation consists of all the combinations of instances in dc_i and dc_j that satisfy F. The condition F is applied on the joined instances $x \in dc_i$ and $y \in dc_j$ and it is true when an attribute a_i of x is related to (e.g., equal, minor, major) an attribute a_j of the instance y. For example, given two data components dc_i and dc_j defined on top respectively of SongKick and YouTube web services

SongKick		
Event_name	Artist	City
Event_name1	Artist_1	City_1
Event_name2	Artist_2	City_2
Event_name3	Artist_3	City_3
Youtube		
Title	Video_URL	City
Title_1	URL_1	City_1
Title_2	URL_2	City_2

all the combinations of instances in dc_i and dc_j are represented in the following table:

SongKick ⋈ YouTube				
Event_name	Artist	City	Title	Video_UR
Event_name1	Artist_1	City_1	Title_1	URL_1
Event_name1	Artist_1	City_1	Title_2	URL_2
Event_name2	Artist_2	City_2	Title_1	URL_1
Event_name2	Artist_2	City_2	Title_2	URL_2
Event_name3	Artist_3	City_3	Title_1	URL_1
Event_name3	Artist_3	City_3	Title_2	URL_2

The final data component is a sub-set of the previous table determined by applying the condition F. In our approach, the condition F is true when the string of the dc_i attribute the user wants to extend (e.g., artist of SongKick) is contained in the string of the dc_j attribute (e.g., title of YouTube).

Definition 8.c. Union

Given a couple of data components dc_i and dc_j, a *Union* is a binary operator defined as:

$$dc_i U dc_j = \left\{x \middle| x \in dc_i\ or\ x \in dc_j\right\}$$

The result of this operation consists of aggregating all the instances in dc_i and dc_j. In relational algebra, as prerequisites to apply the union on two relations, the unified relations must have the same number of attributes of the same type. In our union operation, the unified data components must have the same number of attributes but there is not the need to have the attributes of the same types. For example, given two data components dc_i and dc_j that are connected respectively to SongKick and Eventful web services

SongKick			Eventful	
Event_name	Artist	Date	Title	Artist_name
Event_name1	Artist_1	Date_1	Title_1	Artist_name_1
Event_name2	Artist_2	Date_2	Title_2	Artist_name_2
Event_name3	Artist_3	Date_3		

their union is an aggregation of their instances, as shown in the following table.

SongKick U Eventful		
Event_name1	Artist_1	City_1
Event_name2	Artist_2	City_2
Event_name3	Artist_3	City_3
Title_1	Artist_name_1	City_1
Title_2	Artist_name_2	City_2

The attributes of the unified data components can be of different types, for example, their *Date* attributes can be formatted in different ways.

Definition 9. Data mashup

It is the results of the integration of data extracted by different data components. It is a pair:

$$dm = <DC, O>$$

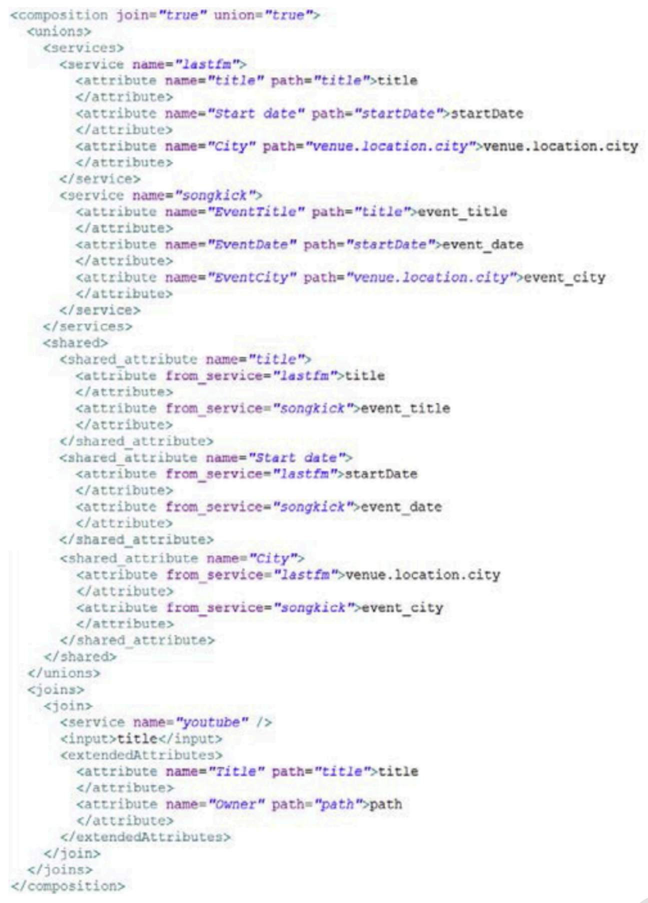
where DC represents the set of data components involved in the composition; O is the set of operations (e.g., join and union) performed between data components in DC.

Data mashup represents an important advance w.r.t. the original model presented in [10] 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 result of the data mashup could not be reused with other UI templates. In our model, the data mashup is a new integrated result-set published as a new data source that in our platform can be also visualized by means of user-selected UI templates.

4. Platform descriptors

In order to make the previous abstractions concrete in the implemented platforms, we defined some Domain-Specific Languages (DSLs) inspired to EMML [39]. New languages were adopted instead of EMML because the composition logic implemented in the EFESTO refers only to a small sub-set of the composition operators available in EMML. Each of these new languages allows us to define internal specifications of the main elements (e.g., UI components, service, UI template) that can guide the dynamic instantiation and execution of the distributed UIs.

Fig. 6 reports an example of our XML language specifying a UI component that renders a data mashup consisting of a union between two services (YouTube and Vimeo) and a join of the unified services with a



```
<composition join="true" union="true">
  <unions>
    <services>
      <service name="lastfm">
        <attribute name="title" path="title">title
        </attribute>
        <attribute name="Start date" path="startDate">startDate
        </attribute>
        <attribute name="City" path="venue.location.city">venue.location.city
        </attribute>
      </service>
      <service name="songkick">
        <attribute name="EventTitle" path="title">event_title
        </attribute>
        <attribute name="EventDate" path="startDate">event_date
        </attribute>
        <attribute name="EventCity" path="venue.location.city">event_city
        </attribute>
      </service>
    </services>
    <shared>
      <shared_attribute name="title">
        <attribute from_service="lastfm">title
        </attribute>
        <attribute from_service="songkick">event_title
        </attribute>
      </shared_attribute>
      <shared_attribute name="Start date">
        <attribute from_service="lastfm">startDate
        </attribute>
        <attribute from_service="songkick">event_date
        </attribute>
      </shared_attribute>
      <shared_attribute name="City">
        <attribute from_service="lastfm">venue.location.city
        </attribute>
        <attribute from_service="songkick">event_city
        </attribute>
      </shared_attribute>
    </shared>
  </unions>
  <joins>
    <join>
      <service name="youtube" />
      <input>title</input>
      <extendedAttributes>
        <attribute name="Title" path="title">title
        </attribute>
        <attribute name="Owner" path="path">path
        </attribute>
      </extendedAttributes>
    </join>
  </joins>
</composition>
```

Fig. 6. A UI component descriptor codified with our XML language.

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, which 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.*

Each service listed in the *service* tag is detailed in a separate service descriptor XML file. In Fig. 7, the YouTube service descriptor is reported: inside the root tag called *service*, there are the tags *source, inputs, parameters, attributes* and *flags.* The first three nodes represent all the information useful to query a data source. The fourth node, *attributes*, describes the instance attributes. The last node, *flag*, is introduced to solve the heterogeneity problem of the data sources. In fact, the remote web services typically send the results by using JSON files but the list of results is formatted in different ways (e.g. inside a JSON array).

Another XML descriptor introduced in our model regards the UI Template. In Fig. 8, 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.

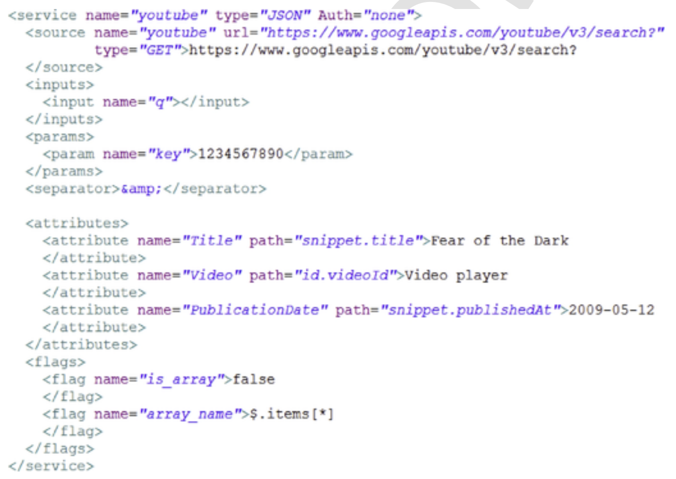
The UI template descriptor is linked with the VI schema through the XML mapping descriptor. An example of mapping is reported in Fig. 9. In this descriptor, the root node, *mappings*, has two attributes: *templatetype* and *templatename*. The first one recalls the name of a UI Template (e.g. *list*), the second one the name of its sub-template (*list_A*).

5. From the model to the platform architecture

The model presented in Section 3 guides designers and software engineers in developing mashup platforms targeting non-programmers. The model highlights the main concepts of a mashup platform without emphasizing technical aspects. In this section, we report a high-level overview of the architecture of the EFESTO mashup platform, in order to illustrate how it implements the mashup model.

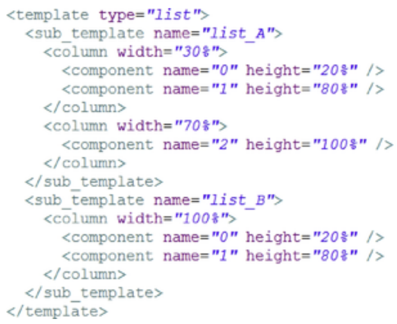
The architecture is characterized by tree-layers (Fig. 10). On top, the *UI layer* provides and manages the visual language that allows end users to perform mashups without requiring technical skills. Such language is based on *UI Components* that use *UI Templates* and *Actionable UI Components* to allow users to visualize and manipulate data extracted from remote sources. The *UI layer* runs in the user's Web browser and communicates with the *Logic* and *Data layer* that run on a remote Web server.

The *Logic Layer* implements components that translate the actions performed by end users at the *Interaction Layer* into the mashup executing logic. In particular, the *Mashup Engine* is invoked each time an event, requiring the retrieval of new data or the invocation of service operations, is generated. The *Event Manager*, instead, manages the UI Components coupling. In particular, when users define a synchronization between two UI Components A and B, it instantiates a listener that waits for an event on A that, when triggered, causes the execution of an operation on B, according to the coupling defined by the user.



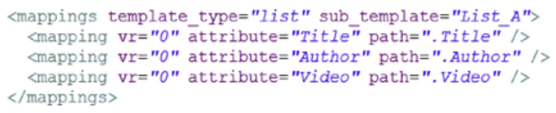
```
<service name="youtube" type="JSON" Auth="none">
  <source name="youtube" url="https://www.googleapis.com/youtube/v3/search?"
          type="GET">https://www.googleapis.com/youtube/v3/search?
  </source>
  <inputs>
    <input name="q"></input>
  </inputs>
  <params>
    <param name="key">1234567890</param>
  </params>
  <separator>&amp;</separator>

  <attributes>
    <attribute name="Title" path="snippet.title">Fear of the Dark
    </attribute>
    <attribute name="Video" path="id.videoId">Video player
    </attribute>
    <attribute name="PublicationDate" path="snippet.publishedAt">2009-05-12
    </attribute>
  </attributes>
  <flags>
    <flag name="is_array">false
    </flag>
    <flag name="array_name">$.items[*]
    </flag>
  </flags>
</service>
```

Fig. 7. An example of service descriptor codified with our XML language.



```
<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>
```

Fig. 8. An example of list UI template descriptor codified with our XML language.



```
<mappings template_type="list" sub_template="List_A">
  <mapping vr="0" attribute="Title" path=".Title" />
  <mapping vr="0" attribute="Author" path=".Author" />
  <mapping vr="0" attribute="Video" path=".Video" />
</mappings>
```

Fig. 9. An example of mapping descriptor.

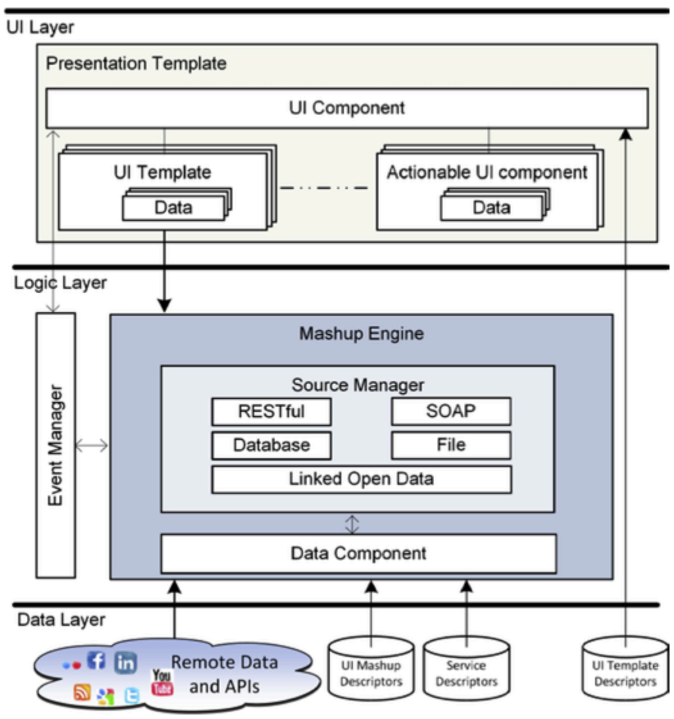


Fig. 10. An high-level overview of the EFESTO three-layer architecture.

The *Data Layer* stores the XML-based descriptors described in Section 2 into proper repositories. In addition, at this layer there are the remote data sources that reside on different Web servers.

6. A utilizazion study

The benefits of the proposed model, and in particular of the Actionable UI components, was assessed during a utilization study inspired by the one in [22]. Participants were required to perform real tasks using the system. The study took place during a scientific meeting in which the prototype of EFESTO platform was presented. Participants were asked to interact with EFESTO platform to express and satisfy their information needs in a direct and dynamical way. By observing people using the platform, we aimed to assess whether the notion of Actionable UI Component actually provides an added value with respect to the users' needs and expectations.

6.1. Participants and design

We recruited a total of 7 participants (4 female), aged between 20 and 60. Single-user interactions with the platform were scheduled. To guide participants in using the platform, they were provided with a scenario consisting of 4 steps that the users had to follow during their interactions.

The main persona of the scenario was Maria; she wants to attend a musical event with her friends and 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 offering data on music events and 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 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 pre-selection 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 workspace 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. Maria can edit the message before sending it. 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.

6.2. Procedure

The study took place in a quiet and isolated laboratory where we installed the study apparatus. Two HCI researchers were involved in the study. One of the two researchers acted as facilitator. The second researcher took notes.

Each participant interacted for about 30 minutes for a total of 4 hours. They all followed the same procedure. First, each participant was asked to sign a consent form. Then, the facilitator showed a quick demo of EFESTO 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 the AttrakDiff questionnaire.

6.3. 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 [9]).

The AttrakDiff questionnaire filled in by the participants at the end of the scenario, helped us to understand how users personally rate the usability and design of our system. This questionnaire records the perceived pragmatic quality, the hedonic quality and the attractiveness of an interactive system. In particular, the following system dimensions are evaluated: *Pragmatic Quality* (PQ): describes the usability of a system and indicates how successfully users are in achieving their goals using the system; *Hedonic Quality - Stimulation* (HQ-S): 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 one of another, and contribute equally to rating attractiveness.

6.4. Results and discussion

The main results of this study come from the AttrakDiff™ questionnaire. Fig. 11 depicts a *portfolio diagram* that summarizes the *hedonic quality* (HQ) and *pragmatic quality* (PQ) system performances according to the respective confidence rectangles. The values of hedonic qualities are represented on the vertical axis (bottom = low value), while 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 room for improvements. The pragmatic quality confidence interval is quite large. This could be attributed to the limited sample of participant who had varying experience with other similar systems and knowledge about tasks performed.

Another perspective on the questionnaire results is provided by the diagram shown in Fig. 12. In this presentation, hedonic quality distinguishes between the 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

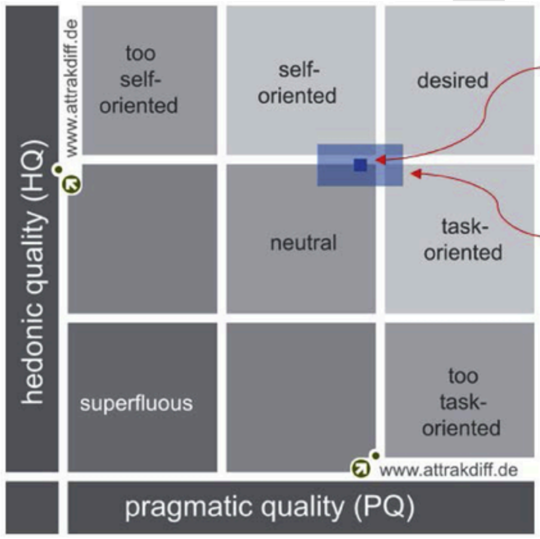


Fig. 11. Portfolio with average values of the dimensions PQ and HQ and the respective confidence rectangles of the system.

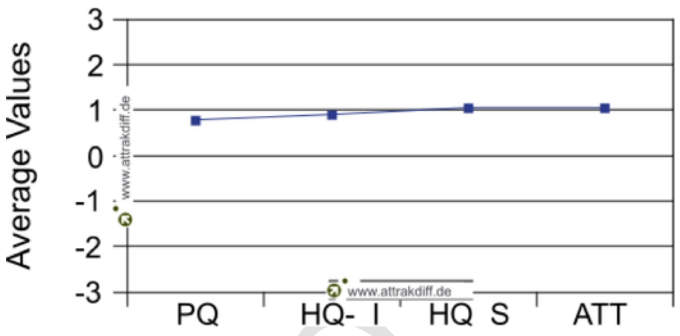


Fig. 12. Mean values of the four AttrakDiff™ dimensions of our system.

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 attractiveness value located in the above-average region, the overall impression is that it is very attractive. The diagram shown in Fig. 13 provides a detailed view of the rating given to the AttrakDiff adjective-pairs questions that determined the values of the PQ, HQ-I, HQ-S and ATT dimensions discussed above.

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 – a double click selection would be preferred; the font size is too small for a 15'' laptop; multiple selection and filter mechanisms are not provided in data sources and containers; a button for deleting all the data in a container is missing; data sources and containers have to be synchronized.

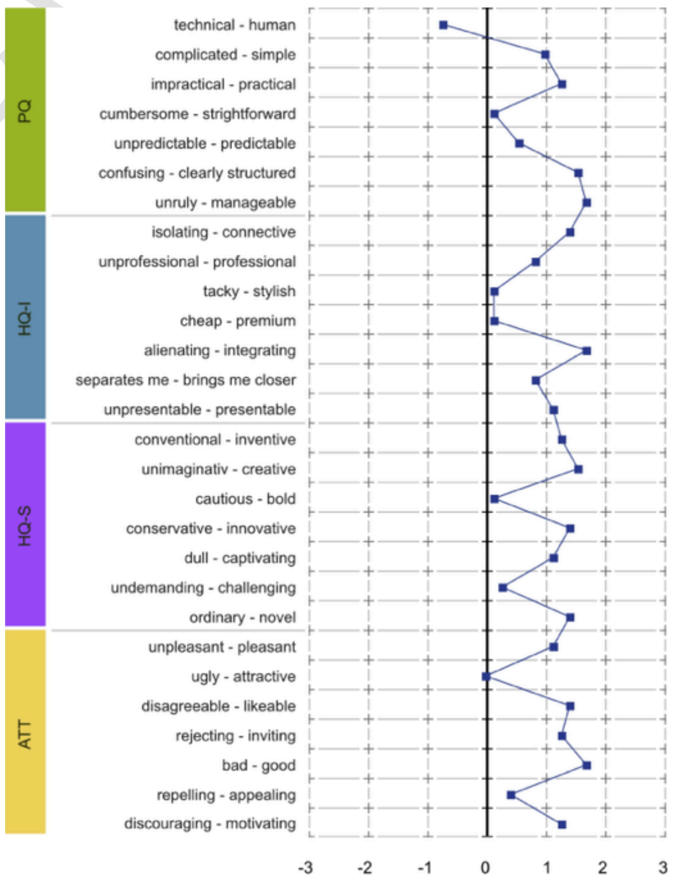


Fig. 13. Mean values of the AttrakDiff™ adjective-pairs for EFESTO.

7. Classifying dimensions and EFESTO characterization

In this section, we present some classifying dimensions that can help clarify the main concepts and techniques underlying mashup design and development, which in turn lead to identify the salient ingredients of mashup composition platforms. The discussed dimensions take into account the contribution of some works published in literature. The book authored by Daniel and Matera is a comprehensive reference for mashups [14], and systematically covers the main methods and techniques for 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. Some other publications also outline several mashups and mashup tool features identified by reviewing the literature on mashups [34]. In [1] the authors propose a design space of mashup tools by surveying more than 60 articles on mashup tools and pointing out that only 22 tools are online. Based on these 22 tools, they propose a model focused on the main perspectives occurring in the design of mashup tools. On the basis of this model, and taking into account what is reported in [14,34], in the rest of this section we provide a further characterization of EFESTO in relation to the dimensions that most characterize mashup tools. The considered dimensions are reported in Table 1, indicating with * the ones derived from the design issues in [1].

7.1. Targeted end users

In terms of programming skills, the end users of mashup platforms range from *non-programmers* to experienced *programmers*, with in the middle professional end users without programming skills, but who are interested in computers and technology - sometimes they are also called *local developers* [33].

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 do not require learning/use of programming languages and technical mechanisms common for ICT experts and engineers (e.g., the use of logical operators and 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 the execution of parameterized mashups.

Local developers are users with knowledge in ICT technology and software usage, with an attitude to explore software. However, they might lack specific skills for some technologies. Tools like mashup platforms can thus provide them with composition functionality to assemble from scratch Web applications easily, by composing predefined components or by customizing and changing existing examples and templates. Even for these end users, mashup tools have to provide a high level of abstraction that ideally hides all the underlying technical complexity of the mashup development.

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

Table 1
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 Metaphor *	*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*
	Runtime environment	*Desktop – Mobile – Cloud*
	Supported Resources	*RESTful – SOAP – smart things – file – database – CSV – excel – smart things*

7.1.1. Prominent mashup tools

Typically, tools for experienced programmers are very powerful but less usable. An example is *Node-RED* [27], a tool for wiring together hardware devices, APIs and online services by means of nodes representing control statements, JavaScript functions and debug procedures. On the contrary, tools for non-programmers implement simplified mechanisms that sacrifice expressive power. An example has been described in [20]: it supports the development of adaptive user interfaces that react 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.

A tool for local developers is presented in [13], 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 page 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 can append a list of Vimeo videos. This data integration can be extended 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 drop-ping them into a target page (e.g. personal Web page) and choosing options (filtering and ordering).

7.1.2. EFESTO classification

EFESTO and the model it implements are strongly oriented to non-programmers and local developers. In fact, our approach is targeted towards the End User-Development of Mashups, and is therefore devoted to non-technical users, to provide them with a composition paradigm that fits their mental model.

7.2. 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, in [1] the author identified two categories: *semi-automation* and *full-automation*. We also introduce a new category, *manual*, to cover tools without any support in mashup creation.

Tools that offer a *semi-automated* creation of mashup partially support users by providing low levels of guidance and assistance. A semi-automated tool requires users to have more skills, but guarantees a high degree of freedom in creating a mashup that satisfies their 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, and play a supervisory role of just providing input or validating mashup results. These tools require a short learning curve and decrease the effort in mashup development. However, these facilities limit the possibility of creating a mashup that fits all the user needs. The *manual* category then refers to those tools that do not provide any support to the users during the mashup creation.

7.2.1. Prominent mashup tools

An example of semi-automated creation tool is *We-Wired Web* [3], a web app that allows non-technical people to easily share data between web services; it also allows technical people to extend the system by adding new web services, triggers, and actions via wiring diagrams.

An example of full-automated tool is *NaturalMash*: it supports users to express in natural language what services they want to include in their mashup and how to orchestrate them [2]. However, to improve accuracy of the user requests, NaturalMash constraints the expression of requirements to a subset of a natural language with limited vocabulary and grammar.

A tool in the manual category is the *Yahoo! console* where programmers can create data mashups by formulating queries written by using the *Yahoo! Query Language (YQL)*. No assistance is given to the users to help them formulate their queries following the YQL syntax. If the query is expressed correctly the JSON or XML result is produced, otherwise a syntax error is shown.

7.2.2. EFESTO classification

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

7.3. Liveness level

The concept of liveness for visual languages presented in [37] is also adopted in the mashup domain [1] with reference to the capacity of tools to immediately interpret and execute the mashups under construction.

According to the classification reported in [1], *Level 1* refers to tools used to compose a mashup as a non-runnable prototype, i.e., 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, on the other hand, in order to run the prototype, the end user is in charge to manage the connection of the mashup presentation layer with an execution environment .

Level 2 refers to executable prototypes: tools in this category produce a 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 mashups through these tools requires skills in programming, since users need to define low-level technical details and thus their use is limited only to programmers.

Level 3 refers to capabilities such as the edit-triggered updates: in this case mashup tools generate mashups that can be easily deployed into operation. Users produce their mashups without devoting too much effort in the manual deployment typically by using two environments: one for the mashup editing and another for mashup execution. The deployment of the mashup under creation in the editing environment could be obtained, for example, by just clicking a run button that produces a deployment in the execution environment.

Level 4 finally refers to the stream-driven updates: it represents 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.

7.3.1. Prominent mashup tools

Microsoft Visio is an example for level 1. It produces UI prototypes that can be completed with data and executed by Microsoft Excel [40].

Activiti is a lightweight workflow and Business Process Management (BPM) platform characterized by features such as modeling environment, validation and remote user collaboration. This tool can be considered as representative of Level 2 of liveness.

An example of a mashup tool for Level 3 is *JackBe Presto*, characterized by a design environment to model the mashup and a detached runtime environment that interprets and runs mashup models and can be used for debugging and monitoring purposes [26].

DashMash supports Web APIs synchronization at the presentation layer, by means of an event-driven paradigm [12]. It exploits a WYSIWYG composition language, which provides immediate feedback on any composition action, thus creating an interleaving between mashup design and execution. *DashMash* can be considered a representative of Level 4 because it supports live modification of the mashup under creation, without distinction between editing and execution time.

7.3.2. EFESTO classification

EFESTO supports live modification of mashups, since it blends into a single environment both the editing and the execution phases (level 4 - Stream-driven updates). The end users edit and run their mashups in the same environment, without needing to switch between two or more different environments. This mechanism is in line with our goal of proposing a mashup tool for non-technical end users. This level of liveness in fact provide the users with an immediate feedback of what they are composing, which allow them understand and control the effect of their composition actions.

7.4. Interaction metaphor

One of the most important aspects affecting the adoption of mashup tools is the interaction metaphor to compose Web services. Actually, this dimension is called *Interaction Technique* in [1]. This is one of the most critical aspects that have limited the adoption of mashup tools in recent years, since the interaction metaphors proposed by several tools were not suitable for non-technical people. In the following, we outline the most adopted interaction metaphors.

The *domain specific language* class requires technical skills since it refers to script languages targeted to solve specific problems for specific domains. These languages are characterized by a formal textual syntax, proper of programming languages. They therefore require users to have strong technical knowledge and skills.

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 identify two sub-dimensions of visual programming languages: *visual wiring languages* and *iconic visual languages*. In the former case, 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. Mashup tools often adopt this mechanism being it the most explicit thanks to the one-to-one mapping between the elements of the control flow (e.g., data passing from one component to another) and the element of the visual notation (e.g., visual boxes wired to each other). In the latter case, tools that implement iconic visual languages translate objects needed for mashup design into visual icons. In this way, if the icons are properly designed, users are facilitated in understanding how to compose these elements.

WYSIWYG (What You See Is What You Get) interaction mechanisms permit the creation and modification of a mashup through a visual interface, without any need to switch from an editing 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, sometimes they also represent a limitation, since users cannot access advanced features, like filtering and conversion, which are typically hidden in the tool backend and thus are not available to the users.

An alternative to the previous interaction metaphors is *Programming by Demonstration*, which supports programming starting from specifying examples of the final artifacts or tasks. Typically, this metaphor is very useful to reduce or remove the need to learn programming languages and therefore it is also adopted in the context of mashup tools. With these composition techniques, users can "show" to the mashup tools how a mashup should be. Tools are then in charge of converting the given examples into a runnable mashup.

Similar to the previous one, the *Programming-by-Example Modification* paradigm allows users to modify a 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 little effort by users.

Spreadsheets are one of the most popular end-user programming approaches to store, manipulate and display data. Mashup tools that implement spreadsheets are oriented towards data mashups, but typically they don't support the creation of mashup with their own user interface [28].

Finally, *form-based* interaction requires users to fill out forms to create an object or to edit an already existing one. Since the form filling is a common practice today on the Web for all kinds of users, mashup tools that implement this technique are easy to use by a wide range of users. However, these tools cannot produce complex mashups.

7.4.1. *Prominent mashup tools*

An example of approach based on specification through a Domain Specific Language is *Swashup* [32]. It is a Web-based development environment where programmers can specify a textual mashup schema based on the Ruby on Rails framework (RoR).

Spacebrew [21] is based on a visual programming language paradigm. It is a toolkit for choreographing web services and smart objects by means of event-condition-actions rules. Rules can be created in a workspace vertically divided in two parts, the left-hand panel for the configuration of the services publishing the events and the right-hand panel for the configuration of services providing actions in response to events. Such services can be connected in a wired fashion.

An example of tool based on the WYSIWYG paradigm is the *SAP Knowledge Workspace* [35], a recently launched commercial platform. It is inspired to the design principles and experience qualities defined within TUX framework [8]. The distinctive feature of the approach is that it provides containers that transform the data they include according to a specific task semantics, and immediately show to the user the results of this transformation. The task flow is not predefined, but it is determined at runtime based on the users' actions, as the users select proper containers depending on the current situation and on the functionality (e.g., data manipulations) needed to further proceed with their task. The framework therefore promotes the definition of elastic environments to natively support users in a variety of spontaneously self-defined task flows, not limiting them to work along highly specific use cases, as typical for applications which are driven by workflow engines or which adopt pre-defined patterns of guided procedures.

Karma [38] is an example of tool based on a programming-by-demonstration paradigm. It addresses the problems of extracting data from Web sources, cleaning and modeling the extracted data, and integrating the data across sources. Instead of requiring users to select and customize a set of widgets, by using the programming-by-demonstration paradigm, Karma can learn the operation that the users want to perform indirectly by looking at the data provided by them.

d.mix [23] offers a Programming-by-Example Modification paradigm. It supports users to browse annotated web sites and select elements of interest. Starting from the user-defined examples, d.mix generates the underlying service calls that extract those elements. The user can then edit, execute, and share the generated code in the d.mix's wiki-based hosting environment. This sampling approach leverages pre-existing web sites as example sets and supports fluid composition and modification of examples.

Mashup tools often exploit spreadsheets to support users in creating mashups. For example, spreadsheet connectors are integrated in tools like *JackBe Presto, IBM Mashup Center*, and *Kapow*, and also allow end-users to easily re-use already-built mashups (outside spreadsheets) in the spreadsheet environment. A different approach implemented in tools like *StrikeIron SOA Express for Excel* and *Extensio Extender for Microsoft Excel* allow the data contained in the web services to be pulled in Microsoft Excel workbook, to "live" in cells, and to be integrated directly by users while still take advantages of all the analytical powers and flexibility of the spreadsheet tools. A different solution is implemented in *AMICO: CALC*, an *OpenOffice Calc* extension used to create mashup by manually writing formula to compose services. In [25] it is reported a qualitative survey on a set of spreadsheet-based mashup tools.

An example of form-based interaction is adopted by *FeedRinse*. Users can exploit wizard procedures and form-based mechanisms to filter and combine multiple RSS feeds, and republish the results in a single RSS feed.

7.4.2. *EFESTO classification*

With respect to this dimension, EFESTO implements a WYSIWYG interaction mechanism to make the mashup modification simpler. In fact, during the wizard procedures that assist the users in editing their mashups, all the Web service details are always visible and under the control of the end users in a WYSIWYG fashion.

7.5. *Runtime environment*

Different devices can be used to run a mashup tool. The desktop PCs are the most common ones since they are equipped with wide screens that offer enough space to visualize mashup components.

7.5.1. *Prominent mashup tools*

All the tools already discussed for the previous dimensions are executed on desktop PCs, typically within Web browsers. However, in some cases, also mobile devices are used to create mashups. For example, the *Atooma* app transforms a smartphone into a "personal assistant", since the users can automate all the manual operations they usually perform with their phone, e.g., combining 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 enables the definition of rules following the syntax "IF something happens DO something else" .

7.5.2. *EFESTO classification*

With respect to this dimension, the EFESTO runtime environment 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.).

7.6. *Supported resources*

This dimension is related to the type of resources that can be mashed-up. In order to create a mashup with different services, mashup tools have to support invoking and composing different types of services, e.g., RESTful and SOAP Web services, data sources, CSV files, databases. The more types of resources the tool is able to support more flexible and powerful the tool is.

7.6.1. Prominent mashup tools

The most common resource adopted in mashup tools are RESTful web services. For example, tools like *NaturalMash, Yahoo! console, DashMash, Atooma* ground their approach on the use of RESTful web services. Another common data source is the spreadsheet, exploited by tools like *StrikeIron SOA Express for Excel* and *Extensio Extender for Microsoft Excel*, which we deeply discussed in the *Interaction Metaphor* dimension. In some cases, also the RSS feed are used as data source, as in the case of *FeedRinse* that we described in the previous dimension. In some cases, mashup tools also exploit private databases, as in the case of *SAP Knowledge Workspace* that allows to dynamically work with information across all SAP and non-SAP applications and data sources.

7.6.2. EFESTO classification

With respect to this dimension, EFESTO implements a mashup engine that permits the manipulation of different data sources such as RESTful web services, Linked Open Data, CSV files and databases. The modularity of this engine fosters an easily integration of new types of data sources.

8. Conclusion

This article discussed some abstractions to promote mashup platforms as tools that permit the easy creation (i.e., even by non-technical end users) of interactive workspaces, whose logic is distributed across different components that are, however, synchronized with each other. One of the main contributions of mashups is the introduction of novel practices, enabling integration of available service and data at the presentation layer, in a component-based fashion - an aspect that so far has been scarcely investigated. Few papers, indeed, discuss and motivate the so-called UI-based integration [10,15,41] as a new component-based integration paradigm, which privileges the creation of fully-fledged artifacts, also equipped with UIs; this is in addition to the traditional service and data integration practices that, instead, mainly act at the logic and data layers of the application stack. In this direction, this article highlights how interactive artifacts can be composed by reusing the presentation logics (i.e., the UIs) and the execution logics of self-contained modules, the so-called Actionable UI Components, providing for the visualization of data extracted from data sources and for data manipulation operations through task-related functions. A model is also provided to describe the most salient elements that enable the integration, at the presentation layer, of Actionable UI components. The results of a utilization study with seven participants demonstrated that users are adequately supported in creating interactive workspaces to access and manipulate data without a predefined task-flow.

By capitalizing on the experience that we gained in recent years in the development of mashup platforms, this article aims to propose a systematic view on concepts and techniques underlying mashup platform design and on the way such concepts materialize into composition paradigms and architectures of corresponding development tools. Indeed, independently of our specific approach and the adopted technologies, our research aims to stimulate a "new way of thinking" towards the definition of systems that really support users in shaping the software they interact with, according to their situational needs. Of course, this new paradigm has to be extensively validated. Therefore, our current work is devoted to enriching the EFESTO platform and to customize it to different application domains, such as e-health, home automation, and cultural heritage. Further validation studies will be thus performed to verify in which measure the paradigm can be fruitfully exploited in such different domains.

Additional efforts are needed to improve the composition paradigm to offer assistance to the end users while composing their workspaces. In some previous works, we already experimented mechanisms to provide recommendations about services and composition patterns to be exploited in a mashup [11]. In situations where a mashup platform has to support the creation of complex workspaces, some mechanisms are also needed to avoid errors, for example checking whether collisions and inconsistencies occur among concurring events at the UI level. Our future work will be also devoted to refine in this direction the composition paradigm and its implementation within the EFESTO platform.

References

[1] S. Aghaee, M. Nowak, C. Pautasso, Reusable decision space for mashup tool design, In: Proc. of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '12). Copenhagen (Denmark), ACM, New York, NY, USA, 2012, pp. 211–220, June 25–28.
[2] S. Aghaee, C. Pautasso, End-user development of mashups with naturalmash, J. Vis. Lang. Comput. 25 (4) (2014) 414–432.
[3] We Wired Web. Retrieved from https://wewiredweb.com/. Last Access March 25.
[4] C. Ardito, P. Bottoni, M.F. Costabile, G. Desolda, M. Matera, M. Picozzi, Creation and use of service-based distributed interactive workspaces, J. Vis. Lang. Comput. 25 (6) (2014) 717–726.
[5] C. Ardito, M.F. Costabile, G. Desolda, R. Lanzilotti, M. Matera, A. Piccinno, M. Picozzi, User-driven visual composition of service-based interactive spaces, J. Vis. Lang. Comput. 25 (4) (2014) 278–296.
[6] C. Ardito, M.F. Costabile, G. Desolda, R. Lanzilotti, M. Matera, M. Picozzi, Visual composition of data sources by end-users, In: Proc. of the International Conference on Advanced Visual Interfaces (AVI '14). Como (Italy), ACM, New York, NY, USA, 2014, pp. 257–260, May 28-30.
[7] C. Ardito, M.F. Costabile, G. Desolda, M. Latzina, M. Matera, Making mashups actionable through elastic design principles, in: P. Díaz, V. Pipek, C. Ardito, C. Jensen, I. Aedo, A. Boden (Eds.), End-User Development - Is-EUD 2015, 9083, Springer Verlag, Berlin Heidelberg, 2015, pp. 236–241, Lecture Notes in Computer Science.
[8] J. Beringer, M. Latzina, Elastic workplace design, Designing Socially Embedded Technologies in the Real-World Computer Supported Cooperative Work, Vol. Part I, Springer, 201519–33.
[9] V. Braun, V. Clarke, Using thematic analysis, Psychol. Qual. Res. Psychol. 3 (2) (2006) 77–101.
[10] C. Cappiello, M. Matera, M. Picozzi, A UI-centric approach for the end-user development of multidevice mashups, ACM Trans. Web 9 (3) (2015) 1–40.
[11] C. Cappiello, M. Matera, M. Picozzi, F. Daniel, A. Fernandez, Quality-aware mashup composition: issues, techniques and tools, In: Proc. of the International Conference on the Quality of Information and Communications Technology (QUATIC '12), 2012, pp. 10–19, 3-6 Sept. 2012.
[12] C. Cappiello, M. Matera, M. Picozzi, G. Sprega, D. Barbagallo, C. Francalanci, DashMash: a mashup environment for end user development (Lecture Notes in Computer Science), in: S. Auer, O. Díaz, G. Papadopoulos (Eds.), Web Engineering - ICWE 2011, 6757, Springer, Berlin Heidelberg, 2011, pp. 152–166.
[13] F. Daniel, Live, personal data integration through UI-oriented computing (Lecture Notes in Computer Science), in: P. Cimiano, F. Frasincar, G.-J. Houben, D. Schwabe (Eds.), Engineering the Web in the Big Data Era, 9114, Springer International Publishing, 2015, pp. 479–497.
[14] F. Daniel, M. Matera, Mashups: Concepts, Models and Architectures, Springer, 2014.
[15] F. Daniel, M. Matera, J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, Understanding UI integration: a survey of problems, technologies, and opportunities, Internet Comput. IEEE 11 (3) (2007) 59–66.
[16] G. Desolda, Enhancing workspace composition by exploiting linked open data as a polymorphic data source (Smart Innovation, Systems and Technologies), in: E. Damiani, J.R. Howlett, C.L. Jain, L. Gallo, G. De Pietro (Eds.), Intelligent Interactive Multimedia Systems and Services (KES-IIMSS '15), 40, Springer International Publishing, Cham, 2015, pp. 97–108.
[17] G. Desolda, C. Ardito, M. Matera, EFESTO: a platform for the end-user development of interactive workspaces for data exploration (Communications in Computer and Information Science), in: F. Daniel, C. Pautasso (Eds.), Rapid Mashup Development Tools - Rapid Mashup Challenge in ICWE 2015, 591, Springer Verlag, Berlin Heidelberg, 2015, pp. 63–81.
[18] G. Desolda, C. Ardito, M. Matera, EFESTO: a platform for the end-user development of interactive workspaces for data exploration (Communications in Computer and Information Science), in: F. Daniel, C. Pautasso (Eds.), Rapid Mashup Development Tools - ICWE '15, 591, Springer International Publishing, 2016, pp. 63–81.
[19] G. Fischer, End-User development and meta-design: foundations for cultures of participation (Lecture Notes in Computer Science), in: V. Pipek, M.B. Rosson, B. de Ruyter, V. Wulf (Eds.), International Symposium on End-User Development - Is-EUD 2009, 5435, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 3–14.
[20] G. Ghiani, F. Paternò, L.D. Spano, G. Pintori, An environment for end-user development of web mashups, Int. J. Hum. Comput. Stud. 87 (2016) 38–64.
[21] Spacebrew. Retrieved from http://docs.spacebrew.cc/. Last Access May 9.
[22] P. Hamilton, D.J. Wigdor, Conductor: enabling and understanding cross-device interaction, In: Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14). Toronto, Ontario (Canada), ACM, New York, NY, USA, 2014, pp. 2773–2782, April 18 - 23.
[23] B. Hartmann, L. Wu, K. Collins, S.R. Klemmer, Programming by a sample: rapidly creating web applications with d.mix, In: Proc. of the Symposium on User Interface Software and Technology (UIST '07). Newport, Rhode Island, ACM, USANew York, NY, USA, 2007, pp. 241–250.

[24] P. Hirmer, B. Mitschang, FlexMash–flexible data mashups based on pattern-based model transformation (Communications in Computer and Information Science), in: F. Daniel, C. Pautasso (Eds.), Rapid Mashup Development Tools - Rapid Mashup Challenge in ICWE 2015, 591, Springer Verlag, 2016, pp. 12–30.
[25] D.D. Hoang, H.-y. Paik, B. Benatallah, An analysis of spreadsheet-based services mashup, In: Proc. of the Conference on Database Technologies (ADC '10). Brisbane, Australia, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 2010, pp. 141–150.
[26] Presto Enterprise Mashup Platform. Retrieved from http://mdc.jackbe.com/prestodocs/v3.7/raql/cacheStore.html. Last Access Nov 26th.
[27] Node-RED. Retrieved from http://nodered.org/. Last Access May 9.
[28] W. Kongdenfha, B. Benatallah, J. Vayssière, R. Saint-Paul, F. Casati, Rapid development of spreadsheet-based web mashups, In: Proc. of the International Conference on World Wide Web (WWW '09). Madrid, Spain, ACM, New York, NY, USA, 2009, pp. 851–860.
[29] R. Krummenacher, B. Norton, E. Simperl, C. Pedrinaci, SOA4All: enabling web-scale service economies, In: Proc. of the International Conference on Semantic Computing (ICSC '09), 1679938, IEEE Computer Society, Berkeley, CA (USA), 2009, pp. 535–542, 14-16 September.
[30] M. Latzina, J. Beringer, Transformative user experience: beyond packaged design, Interactions 19 (2) (2012) 30–33.
[31] H. Lieberman, F. Paternò, V. Wulf, End User Development, Springer, 2006.
[32] E.M. Maximilien, H. Wilkinson, N. Desai, S. Tai, A Domain-Specific Language For Web Apis and Services Mashups, Springer, 2007.
[33] B.A. Nardi, A Small Matter of programming: Perspectives On End User Computing, MIT Press, 1993.
[34] M.A. Paredes-Valverde, G. Alor-Hernández, A. Rodríguez-González, R. Valencia-García, E. Jiménez-Domingo, A systematic review of tools, languages, and methodologies for mashup development, Software 45 (3) (2015) 365–397.
[35] The Knowledge Workspace for the Digital Enterprise. Retrieved from https://icn.sap.com/projects/knowledge-workspace.html. Last Access March 10.
[36] J. Spillner, M. Feldmann, I. Braun, T. Springer, A. Schill, Ad-hoc usage of web services with dynvoker (Lecture Notes in Computer Science), in: P. Mähönen, K. Pohl, T. Priol (Eds.), Towards a Service-Based Internet - ServiceWave 2008, 5377, Springer, Berlin Heidelberg, 2008, pp. 208–219.
[37] S.L. Tanimoto, VIVA: a visual language for image processing, J. Vis. Lang. Comput. 1 (2) (1990) 127–139.
[38] R. Tuchinda, C.A. Knoblock, P. Szekely, Building mashups by demonstration, ACM Trans. Web 5 (3) (2011) 1–45.
[39] A. Viswanathan, Mashups and the enterprise mashup markup language (EMML), Dr. Dobbs J. (2010).
[40] S. Wright, D. Bakmand-Mikalski, R. bin Rais, D. Bishop, M. Eddinger, B. Farnhill, E. Hild, J. Krause, C. Loriot, S. Malik, Designing mashups with excel and vision, Expert SharePoint 2010 Practices, Springer, 2011513–539.
[41] J. Yu, B. Benatallah, F. Casati, F. Daniel, Understanding mashup development, IEEE Internet Comput. 12 (5) (2008) 44–52.
[42] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, M. Matera, A framework for rapid integration of presentation components, In: Proc. of the International Conference on World Wide Web (WWW '07), ACM, Banff, AlbertaCanada, 2007, pp. 923–932, May 8-12.