EFESTO: A Platform for the End-User Development of Interactive Workspaces for Data Exploration

Giuseppe Desolda[1(✉)], Carmelo Ardito[1], and Maristella Matera[2]

[1] Dipartimento di Informatica, Università degli Studi di Bari Aldo Moro, Via Orabona, 4, 70125 Bari, Italy
{giuseppe.desolda,carmelo.ardito}@uniba.it
[2] Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20134 Milan, Italy
maristella.matera@polimi.it

Abstract. This paper illustrates EFESTO, a mashup platform designed to enable end users to explore information by creating interactive workspaces. Within a Web composition environment, end users dynamically create “live mashups” where relevant information, extracted from heterogeneous data sources - including the Linked Open Data – is integrated according to visually defined queries. Visualizations of the resulting data sets can be flexibly shaped-up at runtime. Functions, exposed by local or remote services, also allow users to manipulate the resulting data depending on their situational needs. With respect to other mashup platforms, EFESTO privileges visual composition paradigms that accommodate the end-user mental model for a lightweight data integration within Web workspaces.

Keywords: Mashups · Web composition environments · Data integration

1 Introduction

Mashups are data-centric applications that can be created by composing heterogeneous resources [1]. They are considered a solution for supporting data exploration processes that exceed one-time interactions and allow users to progressively seek for information. As studied in [2], typically users invoke general-purpose search engines and/or specialized verticals, and then use “their brain” (or suitable cognitive aids, e.g., annotations or clipboards) for remembering 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 wider use of the mashup paradigm in real contexts, especially by users who are not experts in programming. While mashups have been identified as a useful mean for application development by the end users [1], so far the research on mashups has largely focused on the enabling integration technologies and standards, with limited attention on easing the mashup development process - in many cases mashup creation still involves the manual programming of

DOI: 10.1007/978-3-319-28727-0_5

service integration. Some user-centric studies [3] 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 [4]. Besides the complexity of the composition paradigm [5], the active interaction with the retrieved data, by means of exploration and manipulation actions, is hardly supported.

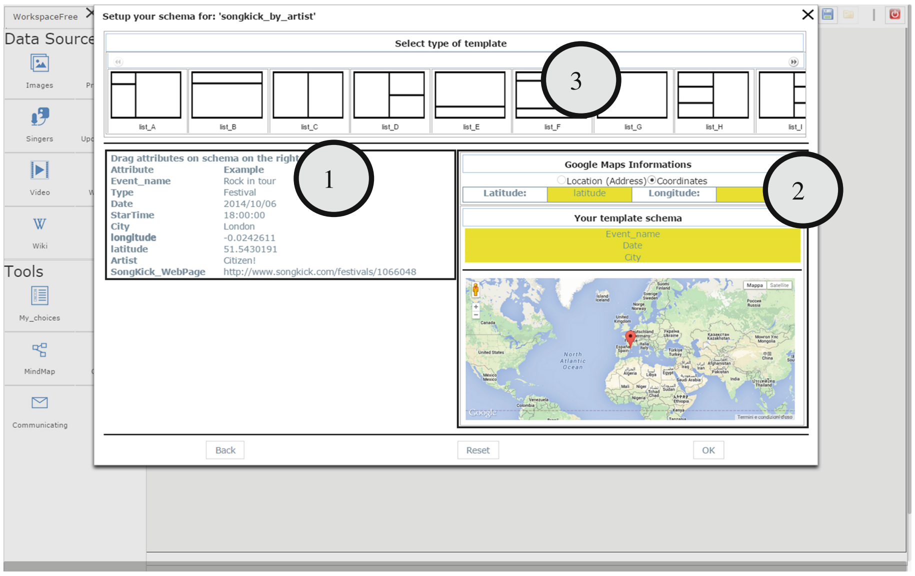
With the intent of overcoming the limitations identified in literature, we defined EFESTO (EFesto End uSer composition plaTfOrm), a platform for the End-User Development of mashups. Efesto was a god of the Greek mythology, who realized magnificent magic arms for other Greek gods and heroes. Analogously, the EFESTO platform aims to put in the hands of 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 Web workspaces. The paradigm was designed taking into account the results of some elicitation studies aimed to identify the end-user mental model for service composition [5, 6]. It was also validated during two field studies in specific application domains, namely Cultural Heritage [5] and Technology Enhanced Learning [7]. Besides helping us assess the elicited mental model, these studies also highlighted new (unexpected) requirements. Among the most important ones, the users expressed the need to manipulate, in a more powerful way, data extracted from services, and the possibility to satisfy more complex information needs by gathering data from the entire Web - not only from pre-packaged components. To overcome these drawbacks, the most recent version of EFESTO offers: *(i)* visual mechanisms to integrate data retrieved from different data sources; *(ii)* a new "polymorphic" data source model that, by exploiting the Linked Open Data (LOD) cloud, enables the access to "mutable" information depending on the situational needs expressed in the mashup under construction; *(iii)* a set of tools to organize, visualize and manipulate extracted data according to specific functions. This new version of the platform is available online at the address: http://efesto.ddns.net/.

This paper illustrates EFESTO with a specific focus on the features presented and discussed during the ICWE 2015 Rapid Mashup Challenge. In particular, Sect. 2 presents, by means of a scenario, the composition paradigm implemented in our mashup platform and also illustrates the sequence of composition steps presented at the challenge. Section 3 describes the platform architecture and in particular the mechanisms supporting the visual construction of live mashups and the way the platform invokes and integrates heterogeneous services, including LOD data sources. Section 4 discusses the level of maturity of our platform, by illustrating how EFESTO has been customized, adopted and evaluated in specific application domains. Section 5 emphasizes the peculiarities of EFESTO by comparing it with other mashup platforms. Section 6 reports how we prepared for the Rapid Mashup Challenge. Section 7 concludes the paper and outlines our current and future work.

2 The EFESTO Composition Paradigm

To illustrate the main features of EFESTO, we now introduce a usage scenario that recalls the live demo given during the ICWE 2015 rapid mashup challenge[1].

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 classified 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* [8], 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.


Setup your schema for: 'songkick_by_artist'

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

Google Maps Informations

Location (Address) [✓ Coordinates]

Latitude: latitude Longitude:

Your template schema

Event_name
Date
City

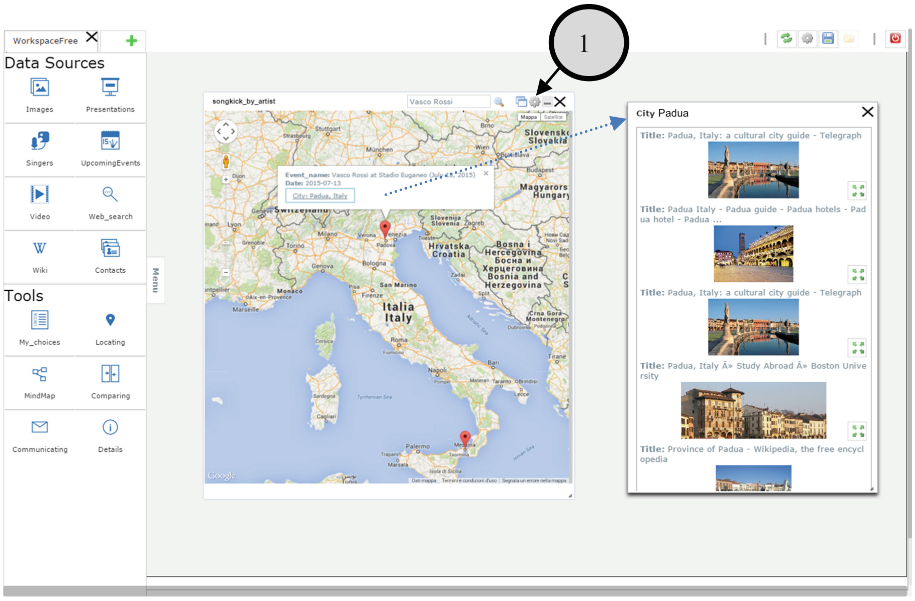
Fig. 1. Mapping between the SongKick data attributes and UI template fields (Color figure online)

As shown in Fig. 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 and drops the *latitude* and *longitude*

[1] The video that faithfully reports the live demo is available at https://youtu.be/bBG5O266y4g.

SongKick attributes into the related fields in the map UI template (Fig. 1, circle #2). Then he chooses a table UI template with three items in column (Fig. 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 Fig. 1, circle #2). These actions represent queries on the underlying data sources that will be successively executed to create the mashup data set.

After performing the mapping phase, Michael saves the mashup. Figure 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.



Event_name: Vasco Rossi at Stadio Euganeo (July 13, 2015)
Date: 2015-07-13
City: Padua, Italy

Title: Padua, Italy: a cultural city guide - Telegraph

Title: Padua Italy - Padua guide - Padua hotels - Padua hotel - Padua ...

Title: Padua, Italy: a cultural city guide - Telegraph

Title: Padua, Italy Â» Study Abroad Â» Boston University

Title: Province of Padua - Wikipedia, the free encyclopedia

Fig. 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 [8]) that he visually expresses through drag and 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 Fig. 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 Fig. 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 Fig. 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.

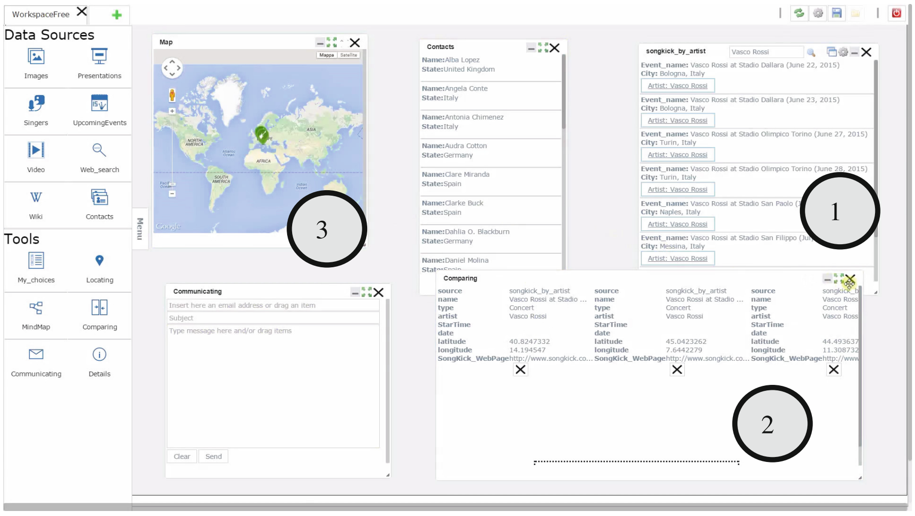
Let us suppose now that, during the interaction with EFESTO, Michael wants to get details about the artists of the music events, such as genre, starting year of activity and artist photo. He does not find any service, among those registered in the platform, that can satisfy this new information need. Thus, he decides to join the SongKick artist attribute with a DBpedia-based *polymorphic data source* [9]. The platform now shows a list of properties related to the musical artist class[2], and Michael creates a new data source based on the properties *genre*, *starting year of activity* and *artist photo*. Henceforward, Michael can find a list of upcoming events in the SongKick component and visualize the additional artist's information, retrieved through the new data source, when clicking on the artist name in SongKick. We call this data source "polymorphic" because, different from pre-registered data sources (e.g., Google Images) that only provide a pre-defined, invariable set of properties, it can enable the access to different information (properties) depending on the attribute in the origin data source it is bound to. For example, if the Michael's join starting point is the SongKick *city* attribute, properties like *borough*, *census*, *year*, *demographics* would be proposed.

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 Fig. 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 and drop the SongKick attributes onto the UI template, as already performed during the SongKick creation.

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 [1]. Our field studies, however, revealed that mashups generally lack *data manipulation functions* that can be instead useful to support common tasks [5, 7] and can empower the users to play a more active role than just consuming the finally visualized information. We thus extended EFESTO 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 and save favourites,

[2] When a service is registered in the platform, each attribute is automatically annotated with a DBpedia class that is semantically close to the attribute meaning [9]; for example the SongKick *Artist* attribute is annotated with the DBpedia *Musical Artist* class.

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 [10]. Coming back to our scenario, as shown in Fig. 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 (Fig. 3, circle #3), this item is automatically 'translated' as pin on the map. Another example is the *Comparing* tool (Fig. 3, circle #2) that assists the user in comparing items retrieved by one or more services (SongKick events in Fig. 3). In general, item transitions across different tools determine different organizations and visualizations of data and progressively enable different functions.



Name:Alba Lopez
State:United Kingdom

Name:Angela Conte
State:Italy

Name:Antonia Chimenez
State:Italy

Name:Audra Cotton
State:Germany

Name:Clare Miranda
State:Spain

Name:Clarke Buck
State:Spain

Name:Dahlia O. Blackburn
State:Germany

Name:Daniel Molina

Event_name: Vasco Rossi at Stadio Dallara (June 22, 2015)
City: Bologna, Italy
Artist: Vasco Rossi

Event_name: Vasco Rossi at Stadio Dallara (June 23, 2015)
City: Bologna, Italy
Artist: Vasco Rossi

Event_name: Vasco Rossi at Stadio Olimpico Torino (June 27, 2015)
City: Turin, Italy
Artist: Vasco Rossi

Event_name: Vasco Rossi at Stadio Olimpico Torino (June 28, 2015)
City: Turin, Italy
Artist: Vasco Rossi

Event_name: Vasco Rossi at Stadio San Paolo (J
City: Naples, Italy
Artist: Vasco Rossi

Event_name: Vasco Rossi at Stadio San Filippo (Ju
City: Messina, Italy
Artist: Vasco Rossi

source	songkick_by_artist	**source**	songkick_by_artist	**source**	songkick_b
name	Vasco Rossi at Stadio ...	**name**	Vasco Rossi at Stadio ...	**name**	Vasco Ross
type	Concert	**type**	Concert	**type**	Concert
artist	Vasco Rossi	**artist**	Vasco Rossi	**artist**	Vasco Ross
StarTime		**StarTime**		**StarTime**	
date		**date**		**date**	
latitude	40.8247332	**latitude**	45.0423262	**latitude**	44.493637
longitude	14.194547	**longitude**	7.6442279	**longitude**	11.308732
SongKick_WebPage	http://www.songkick.co...	**SongKick_WebPage**	http://www.songkick.co...	**SongKick_WebPage**	http://www

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

3 Architecture and Feature Checklist

Figure 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 services, mainly to create the so-called **UI components**. A **UI synchronization logic** then 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.

As already highlighted in the previous section, with respect to other mashup platforms 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* as described later in the paper), in a model-driven engineering (MDE) fashion. Models, expressed according to a Domain-Specific Language [5, 8], 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

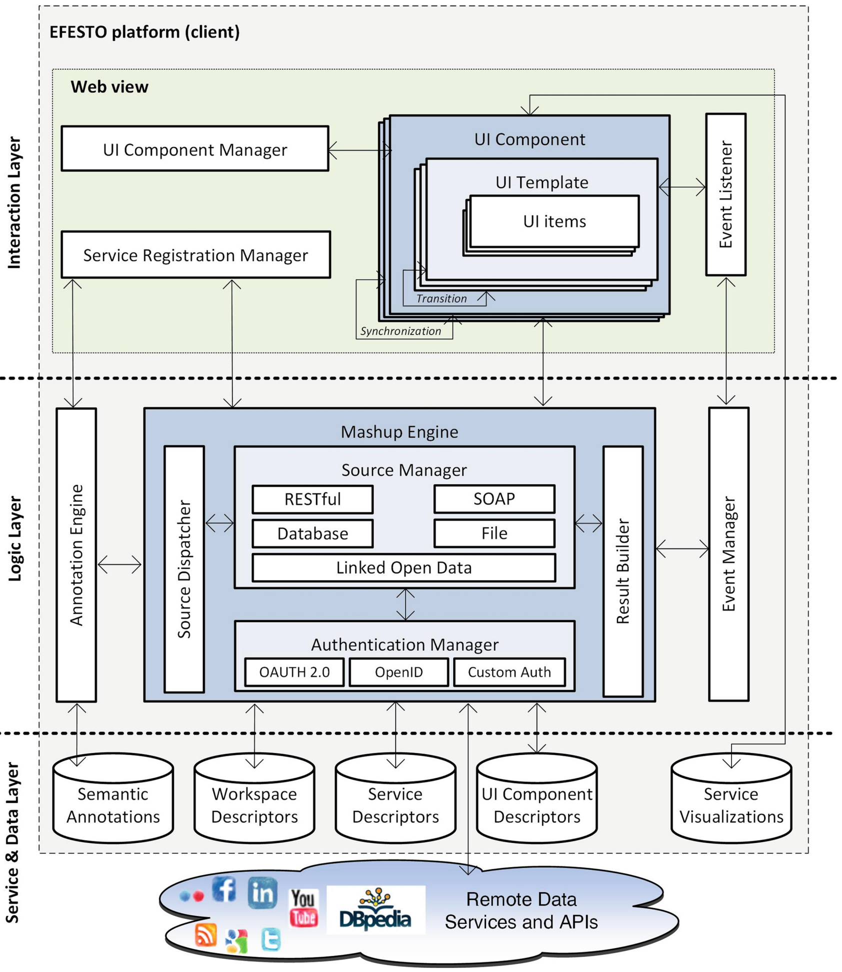


Fig. 4. The EFESTO Three-layers architecture

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.

3.1 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 define 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 [3, 6], is an essential prerequisite to foster EUD of mashups, is made possible by some front-end modules. As represented in Fig. 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 [5, 8]. 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 [8].

As represented in Fig. 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 operations 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.

3.2 Logic Layer

The Logic Layer provides for modules and mechanisms that translate the user composition actions operated at the Interaction Layer into the mashup executing logic. We here describe the different modules 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**, as we already discussed in some previous papers [7].

3.2.1 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 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 Fig. 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>
```

Fig. 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 6 represents an example of JSON array produced by querying the mashup shown in Fig. 5. Finally, the UI Component Manager builds the UI view to render the JSON data according to the layout of the component UI template.

3.2.2 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 Fig. 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.

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).

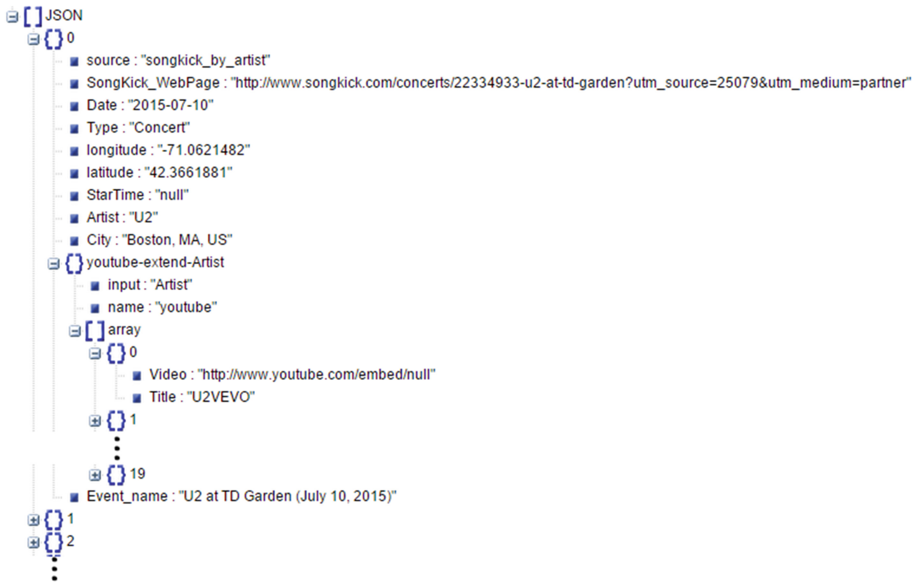

```
[ ] JSON
  { } 0
      source : "songkick_by_artist"
      SongKick_WebPage : "http://www.songkick.com/concerts/22334933-u2-at-td-garden?utm_source=25079&utm_medium=partner"
      Date : "2015-07-10"
      Type : "Concert"
      longitude : "-71.0621482"
      latitude : "42.3661881"
      StarTime : "null"
      Artist : "U2"
      City : "Boston, MA, US"
    { } youtube-extend-Artist
        input : "Artist"
        name : "youtube"
      [ ] array
        { } 0
            Video : "http://www.youtube.com/embed/null"
            Title : "U2VEVO"
        { } 1
          ⋮
        { } 19
      Event_name : "U2 at TD Garden (July 10, 2015)"
  { } 1
  { } 2
  ⋮
```

Fig. 6. JSON array produced by the Mashup Engine invoked on the *UI Component* descriptor shown in Fig. 5 with the *"U2"* query

3.2.3 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, we introduced a new polymorphic data source built upon the LOD cloud, and in particular exploiting the DBpedia knowledge base.

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 [9], which are then stored in the *Semantic Annotation* repository.

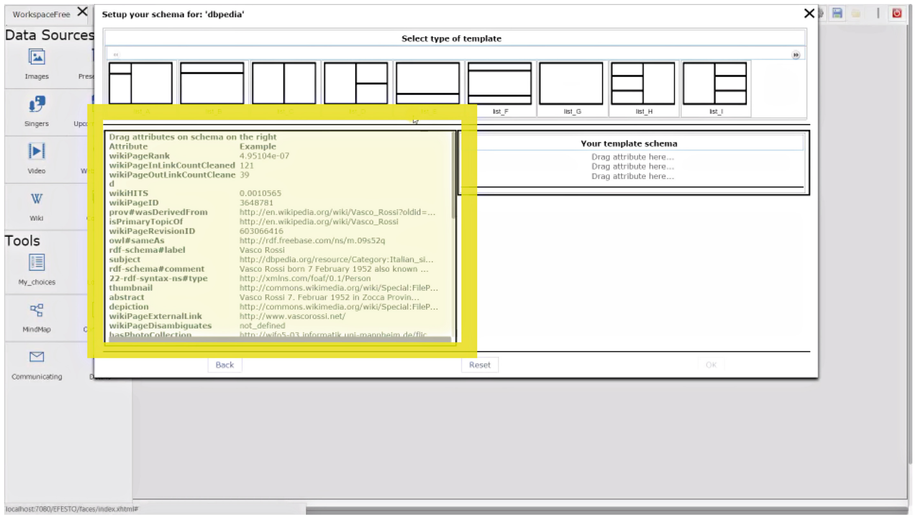
Now let us come back to the Michael scenario and suppose that he wants to join the SongKick Artist attribute with DBpedia. After he decides to use DBpedia as extension data source, the Event Manager triggers the retrieval, by the source manager, of the XML file with the annotations associated with SongKick. 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 Fig. 7). After the drag and drop of a sub-set 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 (Fig. 7) and (ii) during the execution of a SPARQL query, to disambiguate multiple retrieved instances.

3.3 Service and Data Layer

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



Drag attributes on schema on the right

Attribute	Example
wikiPageRank	4.95104e-07
wikiPageInLinkCountCleaned	121
wikiPageOutLinkCountCleaned	39
wikiHITS	0.0010565
wikiPageID	3648781
prov#wasDerivedFrom	http://en.wikipedia.org/wiki/Vasco_Rossi?oldid=...
isPrimaryTopicOf	http://en.wikipedia.org/wiki/Vasco_Rossi
wikiPageRevisionID	603066416
owl#sameAs	http://rdf.freebase.com/ns/m.09s52q
rdf-schema#label	Vasco Rossi
subject	http://dbpedia.org/resource/Category:Italian_si...
rdf-schema#comment	Vasco Rossi born 7 February 1952 also known ...
22-rdf-syntax-ns#type	http://xmlns.com/foaf/0.1/Person
thumbnail	http://commons.wikimedia.org/wiki/Special:FileP...
abstract	Vasco Rossi 7. Februar 1952 in Zocca Provin...
depiction	http://commons.wikimedia.org/wiki/Special:FileP...
wikiPageExternalLink	http://www.vascorossi.net/
wikiPageDisambiguates	not_defined
hasPhotoCollection	http://wifo5-03.informatik.uni-mannheim.de/flic

Fig. 7. Mapping step between the DBpedia-based polymorphic data source properties and the list UI template

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 Fig. 5), and the specification of the component 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)[3] 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 we opted for a custom XML grammar, which is anyway inspired to EMML, but it is simpler.

4 Level of Maturity

The current version of the EFESTO platform is the results of a 4-years research. During this period, we adopted a user-centred approach with the main goal of identifying how a mashup composition paradigm could really help the users themselves. Our research is indeed strongly inspired by and oriented towards End-User Development principles [11]. We therefore conducted several user studies to elicit end-user requirements with respect to the composition of mashups [6] and to validate our design choices and their consequent implementation in the platform [7, 8]. These studies show that EFESTO can be adopted by users without specific expertise in programming with a good level of effectiveness, efficiency, and satisfaction.

These findings are true for several application domains. The EFESTO architecture and the composition paradigm have been indeed designed having in mind customization as a mean to ease the adoption of the platform by different communities of end users, each one featuring specific requirements, background and expertise. In particular, we have investigated meta-design approaches, where different stakeholders (e.g., developers, graphic designers, domain experts) customize different elements of the platform (e.g., UI components, UI templates, visual composition mechanisms) and create artifacts that the end users can exploit profitably [5, 7]. In the end, thanks to this methodological framework, end users can use or customize the platform in different modalities, depending on their expertise and willingness to be engaged in the creation of artifacts: from the visual composition of ready-to-use UI components created by other and more

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

expert stakeholders, to the definition of their own components by means of the mashup operations illustrated above in this paper, to the development of new UI templates in casw a specific application domain requires for different types of data visualizations.

We validated our approach to customization in different contexts. One extensive experimentation was conducted in the Cultural Heritage domain, when our platform was customized to support professional tourist guides. The emerging need in this scenario was to enhance the visits lead by the guides in an archeological park with the possibility to create flexibly multi-device mashups to show to the visitors complementary multimedia material retrieved by different (both public and private) online sources [5, 7, 12].

Another customization experience is then related to the adoption of EFESTO in a Technology-Enhanced Learning (TEL) scenario. In this context students learn about a topic presented in class by their teacher, then complement the teacher's lesson by searching information on the Web, and communicate and share the results of their search with the teacher and other students [7]. Nowadays, schools are provided with different computing devices, not only desktop but also tablets and interactive whiteboards. Teachers and students are increasingly using such devices and various software tools in their daily activities.

A further interesting scenario in which we have customized and we are experimenting EFESTO is the living labs of the VINCENTE (A Virtual collective INtelligenCe ENvironment to develop sustainable Technology Entrepreneurship ecosystems) research project. The aim of the project is to design, implement and test methodological and technological platforms that use services to create ecosystems for sustainable entrepreneurship, which optimize the use of resources, enhance the knowledge, respect the environment and ethical values and ensure the social inclusion. Our current work is devoted to the customization of EFESTO to the specific requirements related to the establishment of collaborative entrepreneurial ecosystems.

5 Related Work

The problem of facilitating the access to Web services and APIs through mashup tools has been attracting the attention of several researchers, who in the last years focused on different issues. From an HCI perspective, empowering a larger class of users to create their own applications requires intuitive abstraction mechanisms, easy development tools and a high level of assistance. Therefore, some research projects have been dealing with the problem of enabling the creation of effective presentations on top of Web services and APIs, to provide a direct channel between the user and the service (e.g. [13]). They focused on the notion of Web Service Graphical User Interfaces (WSGUIs) [14], i.e., on a set of mechanisms to enrich the Web service specifications with annotations that could make the definition of visual interfaces easy.

The previous approaches do not allow the composition of multiple services in an integrated application. In some cases, building a complete Web application equipped with a user interface requires the adoption of additional tools or technologies. Recently, different approaches have been proposed to blend design and execution environments while exploiting intuitive mechanisms to define mashups. For example, NaturalMash allows one to express in natural language what service(s) the users want to use and how to synchronize them [15]. 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). If on the one hand the users have only to type assisted queries to mashup services, on the other hand this paradigm inherits all the natural language processing problems and limitations.

A completely different approach is described in [16] where the authors propose a new perspective on the problem of data integration on the Web, the 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 can also append a list of Vimeo videos. This data integration can also be improved by means of filtering and ordering mechanisms. These operations can be performed, 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). As highlighted by the authors, despite this approach is very promising, some limitations still affect this solution, for example, low performance (UIs need to be instantiated locally), the missing support for more advanced use cases beyond data integration and heterogeneity of structured data in the Web.

As we have illustrated in this paper, in our approach the integration of different services is guided by UI templates, which implicitly provide an integration schema and therefore do not require the users to specify the mapping of service attributes with a global integration schema. Mechanisms similar to UI templates are adopted also in other approaches for the composition of service-based interactive applications, but from a different perspective. For example, in the mashup composition approach presented in [13], a so-called service front end is a form-based UI module that gives a representation of the technical interface of a Web service and provides the users with the list of parameters expected by the service. The user can specify values for such parameters, depending on the needed content. The resulting application is thus able at runtime to query the service and visualize the results in a tabular template. Our UI templates also offer support to query services, but through a paradigm that seamlessly allows the user to define integrated views over different services. Our UI templates then introduce additional abstractions, which go beyond pure service querying as they guide the users in a data integration process resulting into integrated visualizations.

Other recent approaches to perform mashup focus on distributed and/or multi-screen mashups. Among them, the SmartComposition approach [17] 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. Even though the development of distributed, multi-device mashups is not discussed in this paper, we also worked on extending EFESTO to allow multiple users to collaboratively construct and execute mashups across different devices. In fact, the workspaces created in EFESTO can be shared among different users so that they can synchronously collaborate in creating and manipulating new information. The available mechanisms for sharing and collaboration are inspired to the ones of Google Drive. Moreover, chat, annotations and offline messages also support asynchronous collaboration. We validated the devised extensions, and especially their usefulness for the end users, in a user study in the Technology Enhanced Learned domain [7]. The users were satisfied of the devised collaborative mechanisms and found them very useful. However, they expressed the need to further "manipulate" the collaboratively-created workspaces through functions that could allow them to accomplish collaboratively some situational tasks. It was this study that suggested us to move towards the notion of *actionable mashups* [10], i.e., interactive workspaces where users could also invoke tools to manipulate the integrated data across several dimensions. Such new features permit the transition of information between different *task containers*, i.e., dedicated, contextual task environments that, according to the recently proposed notion of Transformative User Experience [18], can support users in accomplishing in an elastic way their tasks. We believe this feature, scarcely explored in literature and not investigated in other mashup platforms, provides for a very innovative direction that could give value to mashups as tools to let users to make sense of data for accomplishing their tasks.

6 ICWE 2015 Rapid Mashup Challenge

During the Rapid Mashup Challenge, we illustrated the EFESTO characteristics described above by means of a demo that followed the same flow of actions as the reference scenario described in Sect. 2. The mashup built on the fly included the services SongKick, YouTube, Vimeo, Google Maps and Google Images, and allowed us to demonstrate: (1) how to define a union of the YouTube and Vimeo data sets; (2) how to join the SongKick Artist attribute (visualized in a Map UI Template) with YouTube; (3) how to shift from a map UI template to a list UI Template for the SongKick UI Component; and (4) how to synchronize at the UI level the new data set with Google Maps (showing the location of selected music events) and Google Images (showing images of the cities where the events take place). During the demo, we also showed how to extend the integrated information retrieved by this core set of services by navigating in the LOD.

Getting prepared for the challenge actually did not require additional efforts as the services mashed up during the demo were already registered in the platform. We only made sure that their descriptors and the adapters for invoking them were running correctly. A problem compromising the correct behaviour of the platform, which is anyway common to many mashup platforms, could be the change of APIs for the registered services, which could compromise their invocation by the platform.

During the demo everything worked perfectly; we wished we had more time to demonstrate some features that we recently introduced in EFESTO that, as described in the previous section, relate to the notion of actionable mashups. The readers interested in these extensions can find more details in [10], and watch the video available at: https://youtu.be/bBG5O266y4g (min 4:00–6:10).

7 Conclusions

In several application domains there is an increasing demand by end users to access, integrate, and use flexibly multiple resources available online. The EFESTO platform tries to respond to this need by letting users easily integrate, by means of an End-User Development paradigm, heterogeneous information that otherwise would be totally unrelated. This approach is very useful in all those situations where, due to varying information needs exposed by the end users, a pre-packaged application could not work properly. The modus operandi promoted by the EFESTO approach also facilitates the construction of new knowledge and its continuous enrichment in contexts where the establishment of communities implies the collaborative creation of knowledge.

This paper described how, in addition to what offered by other platforms, EFESTO also enables a seamless transition of the retrieved data across different organizations, visualizations and functionality. We believe this is a characterizing feature that can pave the way to a new conception of mashups as effective tools for supporting users' tasks and we are devoting several efforts to formalizing the new interaction model for characterizing the possible transitions across different data organizations. The potential of the interaction paradigm was also recognized at the Mashup Challenge. The comments of the jury and of the participants were very positive; and in the end we won the challenge! We also received very encouraging feedback on the idea of including LOD data sources. Our current work is devoted to consolidating LOD navigation and extending the current mechanisms by means of recommendations.

Acknowledgment. This work is partially supported by the Italian Ministry of University and Research (MIUR) under grants PON02_00563_3470993 "VINCENTE", PON04a2_B "EDOC@WORK3.0", and PON03PE_00136_1 "DSE" and by the Italian Ministry of Economic Development (MISE) under grant PON Industria 2015 MI01_00294 "LOGIN". We are also immensely grateful to Prof. Maria Francesca Costabile for her valuable and constant support.

References

1. Daniel, F., Matera, M.: Mashups: Concepts. Models and Architectures. Springer, Berlin (2014)
2. White, R.W., Roth, R.A.: Exploratory search: beyond the query-response paradigm. Synth. Lect. Inf. Concepts Retrieval Serv. **1**(1), 1–98 (2009)
3. Namoun, A., Nestler, T., De Angeli, A.: Conceptual and usability issues in the composable web of software services. In: Daniel, F., Facca, F.M. (eds.) ICWE 2010. LNCS, vol. 6385, pp. 396–407. Springer, Heidelberg (2010)
4. Casati, F.: How end-user development will save composition technologies from their continuing failures. In: Costabile, M.F., Dittrich, Y., Fischer, G., Piccinno, A. (eds.) IS-EUD 2011. LNCS, vol. 6654, pp. 4–6. Springer, Heidelberg (2011)
5. Ardito, C., Costabile, M.F., Desolda, G., Lanzilotti, R., Matera, M., Piccinno, A., Picozzi, M.: User-driven visual composition of service-based interactive spaces. J. Vis. Lang. Comput. **25**(4), 278–296 (2014)

6. Ardito, C., Costabile, M.F., Desolda, G., Lanzilotti, R., Matera, M., Picozzi, M.: Visual composition of data sources by end-users. In: Proceedings of International Working Conference on Advanced Visual Interfaces (AVI), pp. 257–260. Como, Italy, 28–30 May 2014
7. Ardito, C., Bottoni, P., Costabile, M.F., Desolda, G., Matera, M., Picozzi, M.: Creation and use of service-based distributed interactive workspaces. J. Vis. Lang. Comput. **25**(6), 717–726 (2014)
8. Cappiello, C., Matera, M., Picozzi, M.: A Ui-centric approach for the end-user development of multidevice mashups. ACM Trans. Web **9**(3), 1–40 (2015)
9. Desolda, G.: 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. (eds.) Intelligent Interactive Multimedia Systems and Services, vol. 40, pp. 97–108. Springer International Publishing (2015)
10. Ardito, C., Costabile, M.F., Desolda, G., Latzina, M., Matera, M.: Making mashups actionable through elastic design principles. In: Díaz, P., Pipek, V., Ardito, C., Jensen, C., Aedo, I., Boden, A. (eds.) IS-EUD 2015. LNCS, vol. 9083, pp. 236–241. Springer, Heidelberg (2015)
11. Costabile, M.F., Fogli, D., Mussio, P., Piccinno, A.: Visual interactive systems for end-user development: a model-based design methodology. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **37**(6), 1029–1046 (2007)
12. Ardito, C., Costabile, M.F., Desolda, G., Matera, M., Piccinno, A., Picozzi, M.: Composition of situational interactive spaces by end users: a case for cultural heritage. In: Proceedings of Nordic Conference on Human-Computer Interaction (NordiCHI), pp. 79–88, Copenhagen, Denmark, 15–18 October 2012
13. Krummenacher, R., Norton, B., Simperl, E., Pedrinaci, C.: Soa4all: enabling web-scale service economies. In: Proceedings of International Conference on Semantic Computing (ICSC), pp. 535–542, Berkeley, CA, USA, 14–16 September 2009
14. Wajid, U., Namoun, A., Mehandjiev, N.: Alternative representations for end user composition of service-based systems. In: Costabile, M.F., Dittrich, Y., Fischer, G., Piccinno, A. (eds.) IS-EUD 2011. LNCS, vol. 6654, pp. 53–66. Springer, Heidelberg (2011)
15. Aghaee, S., Pautasso, C.: End-user development of mashups with naturalmash. J. Vis. Lang. Comput. **25**(4), 414–432 (2014)
16. Daniel, F.: Live, personal data integration through UI-oriented computing. In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (eds.) ICWE 2015. LNCS, vol. 9114, pp. 479–497. Springer, Heidelberg (2015)
17. Krug, M., Wiedemann, F., Gaedke, M.: Smartcomposition: a component-based approach for creating multi-screen mashups. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 236–253. Springer, Heidelberg (2014)
18. Latzina, M., Beringer, J.: Transformative User Experience: Beyond Packaged Design. Interactions **19**(2), 30–33 (2012)