A Web-based Platform for Dynamic Integration of Heterogeneous Data

Tuan-Dat Trinh
Vienna University of Technology, Vienna, Austria
dat@ifs.tuwien.ac.at

Peter Wetz
Vienna University of Technology, Vienna, Austria
peter.wetz@tuwien.ac.at

Ba-Lam Do
Vienna University of Technology, Vienna, Austria
lam@ifs.tuwien.ac.at

Amin Anjomshoaa
Vienna University of Technology, Vienna, Austria
anjomshoaa@ifs.tuwien.ac.at

Elmar Kiesling
Vienna University of Technology, Vienna, Austria
elmar.kiesling@tuwien.ac.at

A Min Tjoa
Vienna University of Technology, Vienna, Austria
amin@ifs.tuwien.ac.at

ABSTRACT

Whereas the number of open and accessible data sources on the web is growing rapidly, data becomes more heterogeneous and is difficult to use or reuse. Even though many national and international organizations have published their data according to the Linked Data principles, a considerable number of data sources is still available in traditional formats, e.g., HTML, XML, CSV, JSON. This results in a challenge of data aggregation and integration. To address this issue, we apply the visual programming paradigm to develop an open web platform. The platform is based on Semantic Web technologies and aims at encouraging and facilitating use of heterogeneous Open Data sources. We define Linked Widgets as user-driven modules which support users in accessing, processing, integrating, and visualizing different kinds of data. By connecting Linked Widgets from different developers, users without programming skills can compose and share ad-hoc applications that combine Open Data sources in a creative manner.

Categories and Subject Descriptors

D.1.7 [**Visual Programming**]; H.3.5 [**Online Information Services**]: Web-based services; E.2 [**Data Storage Representations**]: Linked representations

General Terms

Management, Design, Standardization

Keywords

Open Data, Heterogeneous Data, Mashup, Widget, Service, Data Integration, Platform, End User

1. INTRODUCTION

In recent years, organizations and governments have made large volumes of useful Open Data available on the web. Publishers frequently release their data under a license that allows anyone to use, reuse and redistribute it. This allows interested stakeholders to analyze the data, put it in a new context, gain insights, and create innovative services. Using *Vienna Open Government Data*[1], for example, we can build a service which allows citizens to find Points of Interest (POI) such as parks, public barbecue areas or bathing areas. Open Data, particularly if published as Linked Open Data (LOD) in a well-defined, machine-understandable, and interlinked manner, has large potential to support informed decisions.

However, end users are not able to directly access, explore, and combine different sources due to a number of technological barriers: (i) Data is stored in heterogeneous formats, e.g., XML, CSV, PDF, JSON, and HTML; (ii) users do not know where to find required data sources; (iii) provided that they are aware of appropriate sources, they frequently do not have the means and skills to access them; and (iv) if users are able of collecting raw data from various sources, they are typically not capable of performing necessary data processing and data integration tasks.

Therefore, end users can not, yet, tap the full potential of Open Data, but rather have to rely on applications built by others. Research towards End User Programming aims to emancipate users from this dependency on programmers. It allows them to build up an application with limited upfront learning time investment [14]. In this research field, widget-based mashups serve as an implementation of the visual programming paradigm which allows end users to compose ad-hoc applications by combining available widgets. Such applications use "*content from more than one source to create a single new service displayed in a single graphical interface*" [5], thereby increasing the value of existing data.

Following the widget-based mashup approach, we have developed a Linked Widgets platform[2] that aims to (i) pro-

[1] `https://open.wien.at/site/`
[2] `http://linkedwidgets.org/`

vide universal and practical advantages without restrictions on domain or data sources, (ii) allow users to combine/integrate multiple Open Data sources and leverage their joint value, and (iii) allow novice users to analyze, integrate and visualize data.

The platform is built upon Semantic Web technologies and its design follows three guiding principles: *openness*, *connectedness*, and *reusability*. *Openness* distinguishes the platform from similar approaches and is the key for achieving our first objective, i.e., the capability to deal with disparate data sources. It means developers can implement and directly add their new widgets to the platform. *Connectedness* means users can combine data from different sources by connecting widgets of distinct developers. Finally, we foster *reusability* by allowing users to make use of the same widget in a dynamic and creative manner to compose various applications.

We use a graph-based model to semantically describe the input and output of a widget so that the platform can make use of the annotated models to provide *semantic search*, *data model matching*, and *auto composition*. *Semantic search* is a mechanism for the discovery of widgets that helps to solve an information need. *Data model matching* allows the platform to highlight compatible widgets (i.e., signal the user which widgets can be connected). *Auto composition* is an innovative approach to compose complete applications automatically from a set of widgets. Research on the latter is still in an early stage of development and beyond the scope of this paper.

The remainder of this paper is organized as follows. In Section 2, we introduce the most basic element of the platform, i.e., Linked Widgets with the responsibility to collect, process and present data. Section 3 illustrates the potential of the platform by means of a sample use case. Section 4 presents our uniform data type and data format to deal with heterogeneous Open Data. After introducing the Linked Widget model in Section 5, we present the most basic functions of the platform built on top of this model in Section 6. Section 7 provides pointers to related work and we conclude in Section 8 with an outlook on future research.

2. LINKED WIDGETS

A widget is an "*interactive single purpose application for displaying and updating local data or data on the web, packaged in a way to allow a single download and installation on a user's machine or mobile device*".[3] We introduced Linked Widgets [23] as the extension of standard widgets with a semantic model following Linked Data principles [3]. The semantic model describes data input/output and metadata such as provenance and license.

Linked Widgets are the key concept that our platform is based upon. To be able to connect with each other, they have input or output terminals. Connecting an input terminal of a widget to an output terminal of another means the former widget takes the output data of the latter as its input data. Similar to functional programming or web services, a Linked Widget can have multiple input terminals, but only a single output terminal.

We distinguish three types of Linked Widgets: *data widget*, *process widget*, and *visualization widget*. A *data widget* retrieves data from a data source and provides the collected data to other widgets. Hence, it has no input terminals. A *process widget* takes input data from other widgets, applies operations on the data, and provides the result to other widgets. It has both input and output terminals. A *visualization widget* has at least one input terminal and presents the data from another widget in a particular manner (e.g., textually or visually). It has no output terminals.

These three types of Linked Widgets are respectively organized into three layers, i.e., *data layer*, *business layer*, and *presentation layer* so that end users can easily make use of them to compose Open Data-consuming applications. Two mandatory components for a complete application are *data* and *visualization* widgets. Inside an application, more than one *visualization* block can be applied, because there can be multiple ways to display the same data.

The idea of Linked Widgets combines Web Services and Service-Oriented Architecture concepts; whereas services target developers, we transform them into widgets aimed at end users. To them, Linked Widgets are "black boxes" with adjustable parameters that control the underlying data collecting and data processing tasks. As we can either perform those tasks locally at the client application or remotely at the server side, the two corresponding types of widgets are named client and server widgets.

2.1 Client Widgets

Since the server of a mashup platform can easily become overloaded with intensive data processing requests from clients, we should make use of client resources whenever possible. An example is Linked Data Fragments [25] in which the client itself will execute complex SPARQL[4] queries after receiving the necessary data fragments from the server. In the Linked Widgets platform, the data processing of client widgets is similarly done on the fly in the client's browser.

The model of client widgets consists of four main components: (i) *input terminals*, (ii) *output terminals* (iii) *options*, and (iv) *a processing function*. Input/output terminals are used to connect widgets in a mashup and represent the data flow. Options are HTML elements inside a widget. They provide a mechanism for users to control a widget's processing function. The processing function defines how widgets receive input and return their output.

Everybody can use arbitrary web languages to easily develop a client widget by following three steps: (i) inject a JavaScript file[5] from the platform into the widget to equip it with the capability of cooperating with others, (ii) define the input and output configuration, and (iii) implement the JavaScript *run(data)* function which defines how the widget processes input data. If a widget has no input terminal, the corresponding *data* object is *null*. Otherwise, during runtime, the platform collects data from all relevant out-

[3] `http://www.w3.org/TR/widgets/`

[4] `http://www.w3.org/TR/rdf-sparql-query/`

[5] `http://linkedwidgets.org/widgets/WidgetHub.js`

put terminals to build the *data* object and pass it to the *run* function as a parameter. The *run* function is executed directly in the browser at the client side to process data.

The server hosting the client widget does not necessarily have to be the same as the platform server. Technically, a client widget is an HTML iframe wrapped in a widget interface. The platform automatically creates the interface, regarding to the input and output configuration, to provide additional functionality for the widget such as *create input and output terminals*, *run, cache, view output data*, and *resize widgets*.

2.2 Server Widgets

Although client widgets are crucial to reduce server load, we should - at least partially - replace them with server widgets to process special types of data, e.g., stream data, real-time data, and long-time processing data; for example, it is inconvenient, if users have to keep a mashup page open for a whole day for it to be able to continuously collect sensor data. Server widgets run, and therefore process, data on the server. They are also of high priority for mobile devices with restricted resources. Clients only act as an interface to the data visualization and modification of the processing, but do not perform the actual processing.

Server widgets, structurally, consist of two services, i.e., *execution service* and *output service*, and a *web page*. The *web page* is the user interface to control the parameters of the *execution service*. Similar to client widgets, inside the *web page*, we have to inject the communication-enabled JavaScript file as well as define the input and output configuration; but we replace the *run(data)* function with the *execution service* to perform the data collecting and processing tasks.

Because a widget can take the output of another one as its input data, the *execution service* of the former should be able to obtain data of the *execution service* of the latter. This is done by the *output service*; when an *execution service* is successfully run, its result data is stored and accessed via the *output service*. As different mashups use distinct instances of the same server widget, a widget instance should be identified by a token, which is used as a parameter to call the *output service* to retrieve the processed data. To summarize, an *execution service* includes following parameters: (i) the widget token generated by the platform, (ii) information of *output services* of other widgets from which it can take input data, and (iii) parameters defined in HTML inputs of the widget interface – the *web page*. An *output service*, meanwhile, takes a widget token, which points to a specific widget instance, as its only parameter.

As an example for how the platform facilitates service composition at runtime, consider a mashup with three server widgets $A \rightarrow B \rightarrow C$. Typically, when a user triggers an action to run a widget, e.g., widget C, this action requires all widgets that provide input to this widget to run first. Because widget C requires the output of widget B, which in turn, requires output of widget A, widgets A and B need to run first. Therefore, as the first step, the platform calls A's *execution service* to request A to collect data from its corresponding sources. When it finishes, the platform uses information of A's *output service* to call B's *execution service*. This service then calls A's *output service* to receive the processed data from A as its input data. Similarly, C can take the output data of B by calling B's *output service* and display the final data to users.

During the service composition process, the platform just calls the *execution services* of involved widgets. It receives no output data of any widgets and performs no data processing task. Data are transferred between and processed inside the widget servers themselves. This contrasts with client widget communication where data is collected and processed directly inside the browser.

3. DATA INTEGRATION EXAMPLE

In this section we present a motivational scenario that illustrates how the platform allows users to handle Open Data sources in an innovative fashion and fosters (re)use of data.

We organize widgets into *widget collections* addressing different problem domains. Each collection might use different Open Data sources. As an example use case, consider a *tour guide* collection of seven widgets that combine data from *Google Maps*, *Last.fm*, *Flickr*, and *LinkedGeoData*.

- **Map Pointer:** Users can define a point on a map. The point's latitude and longitude is then returned as output.
- **Music Artist Search:** Via the last.fm API[6] this widget accepts an artist's name as an input and returns the corresponding URL.
- **Music Event by Artist:** Based on an artist URL this widget returns events this artist participates in while providing a time and event name filter. This is also done with the last.fm API.
- **Point of Interest Search:** This widget leverages the LinkedGeoData[7] repository to find semantically encoded POIs. Users can influence the output by providing parameters. Users can select the type of POI and the radius of retrieved POIs with respect to the incoming location.
- **Flickr Geo Image Search:** By using the Flickr Image Search API[8] this widget enriches location data with images. Users may specify a radius and result limit.
- **Google Map:** This visualization widget displays points on a map. It is typically used to display the final results of a mashup.
- **Geo Merge:** This widget merges two lists of point data into a single list of pairs based on their distance. Users can specify a minimum and a maximum distance between points. The Geo Merge widget therefore serves two purposes, i.e., merging of two inputs into one output and filtering based on distance constraints.

[6] `http://last.fm/api`
[7] `http://linkedgeodata.org/`
[8] `https://flickr.com/services/api/`

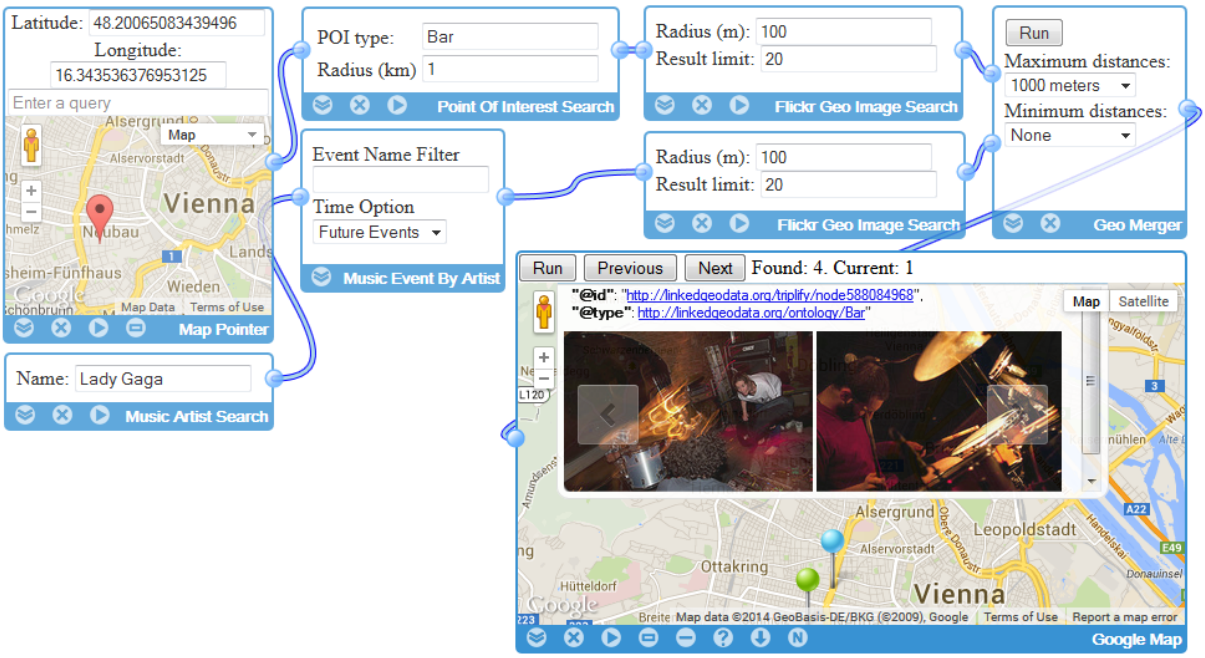


Figure 1: A mashup example

Fig. 1 shows one of the mashups in this collection. It covers the following scenario: *We are traveling to a city X and want to know whether our favorite music artist will give a concert there. After the concert, we want to go get a drink at a bar near the venue. Is there any combination of music events and bars which satisfy these conditions?*

Google Map displays the enriched result. In our example, we get four pairings of a bar and a nearby music event that the artist is involved in. Each bar and music event combination is enriched with illustrative Flickr images and a URI pointing to the corresponding entities at LinkedGeoData and Last.fm.

Many other combinations of widgets are possible. The platform ensures the semantic validity of widget combinations. For example, we can identify (i) all past or future music events of the artist on the map by wiring *Music Event By Artist* directly to *Google Map*, (ii) all POIs near a defined place by connecting *Point of Interest Search* to *Google Map* etc.

4. SEMANTIC DATA TRANSMISSION

Even though the number of interesting data sources on the web is increasing steadily, it is difficult to combine or integrate those sources; this is because data is stored in heterogeneous formats and often lacks structure or semantics. To deal with this situation and foster data integration, we decide to lift all kinds of data to a semantic level and use a uniform format to represent it; this will also facilitate discovery of desired data. A data widget can collect data from an HTML page, or obtain XML result data from a web service, but it always outputs semantic data for other widgets.

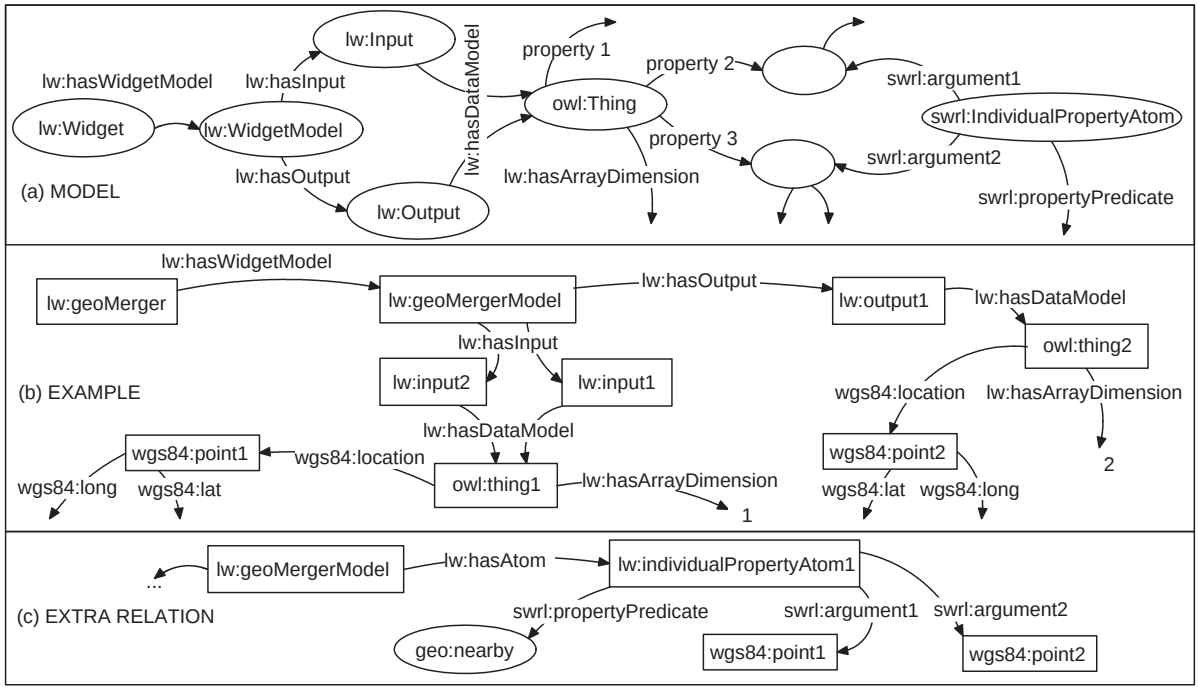
In lightweight applications and applications which include on-the-fly data processing and data transmission, it is important to choose an appropriate data format. A deliberate decision will save a considerable amount of resources, i.e., CPU power, memory, and time. We evaluated RDF, OWL, XML, JSON, and JSON-LD as potential data formats for semantic data transmission in our mashup platform. Among those, we found that JSON-LD, which "*combines the simplicity, power, and web ubiquity of JSON with the concepts of Linked Data*"[9] is the most appropriate for our needs. Since January 2014 it has been an official web standard recommended by the W3C.

Compared to RDF, JSON-LD is more human-readable and takes less memory to present the same information. Additionally, in simple cases of Linked Widget interaction, where the output data model of a widget fits the input data model of another widget exactly (i.e., they have exactly the same structure or the output is a subset of the input), due to the JSON format, widgets can directly receive data from others without further processing tasks. In more complex cases, the output of a widget needs to be modified to be compatible with the input of another widget. In these cases, JSON-LD enables the platform to create a SPARQL query to perform this additional data adaption task.

5. LINKED WIDGET MODEL

We enrich the widget's input and output data based on semantic models. These semantic I/O models are essential for the subsequent search and composition processes. Furthermore, they are crucial for the effective sharing of widgets. For example, even when the number of widgets available is limited (e.g. 43 for Yahoo! Pipes [19] and 300+ for Microsoft Popfly[10]), finding appropriate widgets needed to build a particular mashup solution is already a difficult task. Existing

[9] `http://www.w3.org/TR/json-ld/`
[10] `http://en.wikipedia.org/wiki/Microsoft_Popfly`



lw: http://linkedwidgets.org/ontologies
swrl: http://www.w3.org/2003/11/swrl
geo: http://www.geonames.org/ontology#
wgs84: http://www.w3.org/2003/01/geo/wgs84_pos
owl: http://www.w3.org/2002/07/owl

Figure 2: General Linked Widget model and Geo Merger model

mashup platforms usually employ a text-based approach for widget search, which is not particularly helpful for advanced widget exploration and widget composition tasks.

Fig. 2 presents a part of our ontology for the modeling of Linked Widgets. Using Semantic Web technologies to describe mashups and their components is not by itself a novel approach (cf. [16, 18]). However, rather than capturing the functional semantics and focusing on input and output parameters like SAWSDL [11], OWL-S[11], or WSMO[12], we use a graph-based model [22, 21, 24] to formally annotate the input and output components as well as their relations. We reuse the *SWRL* vocabulary to define the semantic relation between two nodes in the input and output graphs.

Fig. 2 also shows the detailed model of the *Geo Merger* widget. The widget takes two arrays of arbitrary objects containing the *wgs84:location* property as input. Its domain is the *Point* class with two literal properties, i.e. *lat* and *long*. The widget output is a two-dimensional array in which each row represents two objects from two input arrays, respectively. Those objects include locations satisfying the distance filter of the *Geo Merger* widget.

To specify that input/output is an array of objects, we use the literal property *hasArrayDimension* (0: single element; $n > 0$: n-dimensional array). Because the input of *Geo Merger* is an "arbitrary" object, we apply the *owl:Thing* class to represent it in the data model.

[11] `http://www.w3.org/Submission/OWL-S/`
[12] `http://www.w3.org/Submission/WSMO/`

The *point*, *location*, *lat* and *long* terms are available in different vocabularies. However, since a well-established ontology facilitates data exchange between widgets, we chose *wgs84*. The *widget annotator* module interactively recommends frequently used terms of the most popular vocabularies to developers. This eases the annotation process and fosters consistency by diminishing the use of varying terms to describe the same concepts.

With SPARQL queries, we can find a widget that receives or outputs an object containing, for instance, geographic information. Moreover, based on an input terminal, e.g., the first input of *Geo Merger*, we can find all output terminals that can be connected (cf. Listing 1). Conditions that have to be satisfied for the terminals are: (i) matching class and array dimension; and (ii) matching attributes, i.e., the set of attributes required by the input terminal must be a subset of the attributes provided by the output terminal. In the *tour guide* collection, the output of *Map Pointer*, *Point of Interest Search*, *Flickr Geo Image Search*, and *Music Event by Artist* satisfy these conditions.

Similarly, we can model a more advanced widget. Input and output can consist of object attributes and one can model relations between those objects. For example, when modeling the *Geo Merger*, if required, we introduce the *nearby* relation between the two input points as shown in Fig. 2. Due to the graph-based description, the platform can answer questions such as "find all widgets containing the *nearby* relation between two locations".

Listing 1: A SPARQL query for terminal matching

```
PREFIX ifs: <http://ifs.tuwien.ac.at/>
SELECT DISTINCT ?oTerminalName ?oWidget WHERE {
  <http://ifs.tuwien.ac.at/WidgetGeoMerge>  ifs:hasWidgetModel  ?iWModel.
  ?iWModel ifs:hasInput  [ifs:hasName "input1"^^xsd:string;  ifs:hasDataModel ?iDataModel
      ].
  ?iDataModel a ?type.
  ?iDataModel ifs:hasArrayDimension ?listLevel.

  ?oWidget ifs:hasWidgetModel  ?oWModel.
  ?oWModel  ifs:hasOutput [ifs:hasName ?oTerminalName; ifs:hasDataModel ?oDataModel]
  ?oDataModel a [rdfs:subClassOf ?type].
  ?oDataModel ifs:hasArrayDimension ?listLevel.

  FILTER NOT EXISTS{
    ?iDataModel ?property ?iValue.
    FILTER NOT EXISTS {?oDataModel ?property ?oValue.}
  }
}
```

6. PROTOTYPE SYSTEM

6.1 Architectural Design Considerations

When defining the architecture [23] for our mashup platform, we set out to follow three essential design principles, (i) *openness*, (ii)*connectedness*, and (iii) *reusability*. First, Open Data applications cannot tap their full potential, if they are not open. They should follow an open architecture that enables arbitrary developers and end users to contribute and share their work with the Open Data community. An example for the benefits of openness is LinkedGeoData[13], which uses information collected by the OpenStreetMap[14] project and makes it available as an RDF knowledge base according to the Linked Data principles. There is a community behind the project and everyone can contribute geographical data. Users are able to directly edit displayed resources in the map view, which simplifies the process of extending data quantity and improving data quality. Similarly, we encourage developers to contribute Linked Widgets to our mashup platform to enable new combinations of Open Data sources. Furthermore, even end users can create new widgets from composed applications without any programming. Users of the community can finally share, reconfigure, or edit the created Open Data applications.

The *openness* feature improves the way users can store and exploit data of the available Open Data applications. This means that data valuable for the community should be accessible for anyone in a well-structured format. The input and output models of Linked Widgets as well as the metadata of widgets, widget collections, and composed Open Data applications of the mashup platform are published via a SPARQL endpoint. Third parties can develop additional functionality for the platform, e.g., *widget search*, or *input/output model matching* which enables users, for example, to find – given input terminal A – all output terminals B from all other widgets such that the connection between A and B is valid. Valid means that the output of terminal B consists of a semantic structure which makes it attachable terminal A. Users then can discover all possible connections between widgets to continuously build more applications.

[13]http://linkedgeodata.org/About
[14]http://www.openstreetmap.org/

Next, we design the architecture around the idea of *connectedness*, which is implemented via two concepts, i.e., *data connection* and *functionality connection*. Since we, except for domain-specific applications, have many related Open Data sources, the architecture and its implementation should not restrict themselves to a small number of data sources. Instead, combining two or more data sources and enriching the data with additional value creates exciting opportunities.

From the *openness* feature we can derive the *functionality connection* and the *reusability* feature. Anyone can contribute new functionality to an application, however, these should not be separated from each other. It should be possible to connect and reuse them in an effective and efficient manner. Our mashup platform supports reuse in four ways: (i) Users can creatively combine Linked Widgets from different developers to compose Open Data applications, (ii) they can reuse Open Data applications from others, but change the parameters of the constituted widgets, (iii) they can reuse a composed Open Data application as a new widget, and (iv) based on available widgets, developers can implement new widgets to support new use cases.

6.2 Linked Widget Annotator

To allow developers to create and annotate widgets correctly and efficiently, we provide a *Widget Annotator* tool. Developers simply drag, drop and then configure three components, i.e., *WidgetModel*, *Object*, and *Relation* to visually define their widget models. After that, the system automatically generates the OWL description file for the model as well as the corresponding HTML widget file. The latter includes the injected Java Script code snippet served for the widget communication protocol and a sample *JSON-LD* input/output of the widget according to the defined model. Based on that, developers can implement the widget's processing function of the client widgets or the *execution service* of the server widgets, which receives input from widgets and returns output to others.

Developers can also rewrite/improve widgets using server-side scripting languages. Finally, as soon as they have deployed their widgets, developers can submit their work to the platform where it is listed and can be reused with other

available widgets; in particular, widget annotations are published into the LOD repository of widgets which can be accessed from the graph *<http://linkedwidgets.org>* of the `http://ogd.ifs.tuwien.ac.at/sparql` SPARQL endpoint.

6.3 Semantic Widget Search

In line with the growth of Open Data sources, the number of available widgets can be also expected to grow rapidly. In this case, to ensure that users can find widgets on the platform, we provide a *semantic search* feature in addition to conventional search methods which are based on keywords, category, tags, etc.

Because the widgets' RDF metadata is openly available via the SPARQL endpoint, other third parties, if necessary, can also develop their own widget-search tool. Our search tool is similar to the annotator tool, but it is much simpler and directed at end users. By defining the constraints for input/output, they, for example, can find widgets which return *Films* with particular properties for each *Film*, or even find widgets which consist of a relationship between *Films* and *Actors*.

6.4 Mashup Panel

The *Mashup Panel* is a crucial part of the platform; it allows users to compose, publish and share their applications.

The list of widgets that belong to the user's selected collection is placed at the left-hand side of the *Mashup Panel*. Users simply drag and drop widgets into the mashup area at the right-hand side. For each chosen widget, available operations are *resize*, *run*, *view/cache output data*, and *get detailed information* about the widget based on its URI. In the next step, users can wire the input of a widget to the output of another one and thus build up a data-processing flow. The connected widgets, under the coordination of the platform, will communicate and transmit data to each other. Finally, if the whole mashup is saved, parameters set in HTML form inputs from each widget will be automatically detected and stored so that users can publish the final result displayed inside the visualization widget onto their websites. Furthermore, the combined applications are semantically annotated and can be shared between users via their URIs.

We implemented two algorithms, i.e., *auto-matching* and *auto-composition*, to help users acquaint themselves with their new widgets. The *auto-matching* algorithm enables users to find – given input/output terminal A – all terminals B from all other widgets such that connection between A and B is valid. The *auto-composition* algorithm is a more advanced approach in that it can automatically compose a complete application from a widget, or a complete branch that consumes/provides data for a specific output/input terminal. "*Complete*" in this context means that all terminals must be wired. This as well as the semantic search feature distinguish our platform from similar contributions.

7. RELATED WORK

Researchers have been developing mashup-based tools for years. Many of them are geared towards end users and aim to allow them to efficiently create applications by connecting simple and lightweight entities.

Aghaee and Pautasso [1] provide a detailed overview of mashup approaches. They discuss open research challenges which we – at least partly – address with our platform. For instance, we address the *Simplicity* and *Expressive Power Tradeoff* challenges through a semantic model. They also evaluate Yahoo! Pipes [19], IBM Mashup Center[15], Presto Cloud[16], and ServFace [15]. A common limitation they identify for all these platforms is that the wiring paradigm is hard to grasp for non-expert end users. We aim to overcome this barrier, for instance, by recommending valid wiring options to the user.

Other surveys of the mashup literature [2] have developed a number of evaluation criteria and identified shortcomings of existing approaches. Among these shortcomings are lacks of support for (i) event-based behavior, (ii) component discovery features, and (iii) language-dependent mashup components. Computer scientists addressed some of these shortcomings in more recent contributions, but some remain an open challenge.

Grammel and Storey [7] review six different approaches and identify potential areas of improvement and future research. For instance, they argue that context-specific suggestions could support learning of how to build and find mashups. Regarding user interface improvements, they note that designing mechanisms such as automatic mashup generation to provide starting points to end users would enhance usability drastically. This feature is also provided by the platform presented in this paper. However, detecting invalid mashups still remains a challenge that requires appropriate debugging mechanisms for non-programmers.

Giusy et al. [4] analyzed the strengths and weaknesses of popular Mashup tools, i.e., Damia [20], Yahoo Pipes, Microsoft Popfly, Google Mashup Editor[17], Exhibit [9], Apatar[18], MashMaker [6], with respect to the data integration aspect. They are all server side applications, which means that mashups and the data involved are both hosted on the server of the application provider. This may result in problems due to communication overload when a mashup creates too many requests to the servers. Most importantly, the survey claims that each tool requires a considerable level of programming effort by the user to build a mashup even though they are supposed to target "non-expert" users.

Super Stream Collider (SSC) [8], MashQL [10], and DERI Pipes [12] are three platforms aimed at semantic data processing. Whereas SSC consumes live stream data only, MashQL allows users to easily create a SPARQL query, using its custom query-by-diagram language. MashQL cannot aggregate data from different sources and its output visualization only supports text and table formats. DERI Pipes requires users to be familiar with Semantic Web technologies, SPARQL queries, and programming to perform semantic data processing tasks from different data sources. There are multiple other platforms which we only want to point at, such as Vegemite [13], Paggr [17], or Marmite [26]. They

[15] `http://pic.dhe.ibm.com/infocenter/mashhelp/v3/index.jsp`
[16] `http://mdc.jackbe.com/enterprise-mashup`
[17] `http://code.google.com/gme/index.html`
[18] `http://www.apatar.com/`

all follow a mashup-based approach to ease users' access to data sources, but unlike our approach, they do not make use of semantic models.

To sum up, following are the most apparent differences between the Linked Widget platform and similar approaches. (i) Other solutions are low-level data processing oriented whereas the Linked Widget platform acts on a higher, more problem-oriented level. Users working with other tools have to be familiar with special technological and programming concepts, e.g., conditional statements, and conditional loops, to perform their data integration tasks. In using our platform, users first define a goal, e.g., search for POIs near a place, then can discover appropriate widgets, and finally arrange them in a mashup. To ease this process, we organize widgets in a taxonomy tree and in domain-specific collections. Additionally, we provide keyword and semantic search features based on the widget model. (ii) Linked Widgets are backed by a semantic model to facilitate input-output model matching and widget auto-composition features. (iii) Linked Widgets can obtain and process both semantic and non-semantic data. They always lift non-semantic data to a semantic level and produce semantic output data according to its predefined model. (iv) The platform makes use of both client and server widgets to reduce server load. Server widgets allow for data processing without the need to leave a mashup open in a client browser. (v) Finally, the platform is open and everybody can contribute new widgets to extend its functionality.

8. CONCLUSION AND FUTURE WORK

This paper presents an overview of a mashup platform for the dynamic integration of heterogeneous Open Data. The platform aims at end users without knowledge of data integration or programming concepts. We encapsulate semantic, graph-based models inside Linked Widgets and provide mechanisms to annotate their input and output. Leveraging these annotations, the platform combines and searches widgets based on defined semantics.

Because it is open for developers to contribute their widgets to the platform and the widget development is language-independent, we expect to have a large number of versatile widgets in future. As a consequence, users can explore more Open Data sources and easily collect and combine desired information.

In future, this system should serve as a universal data platform and bring together both mashup developers and mashup users. This will allow users to work with different kinds of data sources (e.g., governmental, financial, environmental data) and types of data (e.g., open, linked, tabular data), without technical barriers. From a data perspective, the vision is to support people in their everyday decision-making.

From a technical perspective, we aim to provide a mashup platform for the exploration of Open Data following Semantic Web design principles. We also want to push these ideas a step further by lifting non-semantic data on a semantic level and leverage its full potential.

Future research will focus on using the semantic models, especially for the widget auto-composition feature. We also need to improve the model-matching algorithm by utilizing ontology alignment techniques, since two models can use different resources from different ontologies. Another interesting direction for future research is the automatic creation of new widgets able to handle dynamic web data as an input source.

References

[1] S. Aghaee and C. Pautasso. End-user programming for web mashups: Open research challenges. In *Proceedings of the 11th International Conference on Current Trends in Web Engineering*, ICWE '11, pages 347–351. Springer Berlin Heidelberg, 2012.

[2] S. Aghaee and C. Pautasso. An evaluation of mashup tools based on support for heterogeneous mashup components. In *Proceedings of the 11th International Conference on Current Trends in Web Engineering*, ICWE '11, pages 1–12. Springer Berlin Heidelberg, 2012.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[4] G. Di Lorenzo, H. Hacid, H.-y. Paik, and B. Benatallah. Data integration in mashups. *SIGMOD Rec.*, 38(1):59–66, June 2009.

[5] N. C. Engard. *Library mashups : exploring new ways to deliver library data*. Information Today, Medford, 2009.

[6] R. J. Ennals and M. N. Garofalakis. Mashmaker: Mashups for the masses. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 1116–1118, New York, NY, USA, 2007. ACM.

[7] L. Grammel and M.-A. Storey. A survey of mashup development environments. In M. Chignell, J. Cordy, J. Ng, and Y. Yesha, editors, *The Smart Internet*, volume 6400 of *Lecture Notes in Computer Science*, pages 137–151. Springer Berlin Heidelberg, 2010.

[8] D. L. P. Hoan Nguyen Mau Quoc, Martin Serrano and M. Hauswirth. Super stream collider–linked stream mashups for everyone. In *Proceedings of the Semantic Web Challenge co-located with ISWC 2012*, Boston, US, 2012.

[9] D. F. Huynh, D. R. Karger, and R. C. Miller. Exhibit: Lightweight structured data publishing. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 737–746, New York, NY, USA, 2007. ACM.

[10] M. Jarrar and M. D. Dikaiakos. Mashql: A query-by-diagram topping sparql. In *Proceedings of the 2nd International Workshop on Ontologies and Information Systems for the Semantic Web*, ONISW '08, pages 89–96, New York, NY, USA, 2008. ACM.

[11] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell. Sawsdl: Semantic annotations for WSDL and XML schema. *Internet Computing, IEEE*, 11(6):60–67, 2007.

[12] D. Le-Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, and C. Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 581–590, New York, NY, USA, 2009. ACM.

[13] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau. End-user programming of mashups with vegemite. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, IUI '09, pages 97–106, New York, NY, USA, 2009. ACM.

[14] B. A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, MA, USA, 1993.

[15] T. Nestler, M. Feldmann, G. Hübsch, A. Preußner, and U. Jugel. The servface builder - a wysiwyg approach for building service-based applications. In *Proceedings of the 10th International Conference on Web Engineering*, ICWE '10, pages 498–501, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] A. H. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H.-y. Paik. Semantic-based mashup of composite applications. *IEEE Trans. Serv. Comput.*, 3(1):2–15, 2010.

[17] B. Nowack. Paggr: Linked data widgets and dashboards. *Web Semant.*, 7(4):272–277, 2009.

[18] S. Pietschmann, C. Radeck, and K. Meißner. Semantics-based discovery, selection and mediation for presentation-oriented mashups. In *Proceedings of the 5th International Workshop on Web APIs and Service Mashups*, Mashups '11, pages 7:1–7:8, New York, NY, USA, 2011. ACM.

[19] M. Pruett. *Yahoo! Pipes*. O'Reilly, first edition, 2007.

[20] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, and A. Singh. Damia: Data mashups for intranet applications. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1171–1182, New York, NY, USA, 2008. ACM.

[21] M. Taheriyan, C. Knoblock, P. Szekely, and J. Ambite. A graph-based approach to learn semantic descriptions of data sources. In H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. Parreira, L. Aroyo, N. Noy, C. Welty, and K. Janowicz, editors, *The Semantic Web - ISWC 2013*, volume 8218 of *Lecture Notes in Computer Science*, pages 607–623. Springer Berlin Heidelberg, 2013.

[22] M. Taheriyan, C. A. Knoblock, P. Szekely, and J. L. Ambite. Rapidly integrating services into the linked data cloud. In *Proceedings of the 11th International Conference on The Semantic Web - Volume Part I*, Lecture Notes in Computer Science, pages 559–574. Springer Berlin Heidelberg, 2012.

[23] T.-D. Trinh, B.-L. Do, P. Wetz, A. Anjomshoaa, and A M. Tjoa. Linked widgets: An approach to exploit open government data. In *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, IIWAS '13, pages 438–442, New York, NY, USA, 2013. ACM.

[24] R. Verborgh, T. Steiner, D. Van Deursen, R. Van de Walle, and J. Gabarró Vallés. Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc. In *Proceedings of the 7th International Conference on Next Generation Web Services Practices*, pages 373–379, 2011.

[25] R. Verborgh, M. Vander Sande, P. Colpaert, S. Coppens, E. Mannens, and R. Van de Walle. Web-scale querying through Linked Data Fragments. In *Proceedings of the 7th Workshop on Linked Data on the Web*. CEUR-WS.org, Apr. 2014.

[26] J. Wong and J. I. Hong. Making mashups with marmite: Towards end-user programming for the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1435–1444, New York, NY, USA, 2007. ACM.