Linked Widgets Platform for Rapid Collaborative Semantic Mashup Development

Tuan-Dat Trinh(✉), Peter Wetz, Ba-Lam Do, Elmar Kiesling, and A. Min Tjoa

TU Wien, Vienna, Austria
{tuan.trinh,peter.wetz,ba.do,elmar.kiesling,a.tjoa}@tuwien.ac.at

Abstract. In recent years, data has become vital in supporting our everyday lives. Along with large volumes of open data available on the web, various types of public, private, and enterprise data are stored in the cloud or distributed over multiple devices. The value of this data would increase drastically if we were able to integrate it. This would enable more sophisticated presentation and analysis of previously disparate data. So far, however, it is challenging for non-expert users to efficiently make use of such data because (i) *data heterogeneity* hampers integration of different kinds of data that are stored in various formats and spread among storage infrastructures; (ii) manual data integration processes are typically neither *reproducible*, nor *reusable*; and (iii) the lack of support for *exploration* does not allow for the integration of *arbitrary data sources*. This paper tackles these challenges by introducing a mashup platform that combines semantic web and mashup concepts to help users obtain insights and make informed decisions. To this end, we leverage a semantic model of mashup components for automated techniques that support the user in exploring available data. Moreover, we introduce a collaborative and distributed model to create and execute mashups. This facilitates distributed ad-hoc integration of heterogeneous data contributed by multiple stakeholders.

1 Context and Goals

Due to the evolution of the web, services, and a large number of smart devices, we can now access and make use of various kinds of data to support everyday decision making. On the one hand, large volumes of open data have been made publicly available covering many topics and aspects. Open data has the potential to create new insights and support informed decisions. Adopted by the G8 in 2013, the Open Data Charter[1] reflects the growing importance of open government data. The charter stipulates that open data must be discoverable, accessible, and usable by all people. On the other hand, we may possess private data which should not be seen by the public. Both open and private data can be stored in the cloud or on our own devices, such as mobile phones or desktop computers.

[1] https://www.gov.uk/government/publications/open-data-charter/g8-open-data-charter-and-technical-annex (accessed October 7, 2016).

F. Daniel and M. Gaedke (Eds.): RMC 2016, CCIS 696, pp. 51–73, 2017.
DOI: 10.1007/978-3-319-53174-8_4

Data integration offers a new view on data and helps us to explore and reveal useful information hidden in and spread among multiple data sources. There are many scenarios that demonstrate the value of data integration. However, it is challenging for end users to make effective use of available isolated datasets, because (i) *data heterogeneity* hampers the integration of different kinds of data that are stored in various formats such as CSV, XML, JSON, or RDF and spread among various storage infrastructures (e.g., databases, files, cloud, personal computers, mobile phones); (ii) manual data integration processes that users perform to collect, clean, enrich, integrate, and visualize data are typically neither *reproducible*, nor *reusable*; (iii) the lack of support for *exploration* does not allow for the integration of *arbitrary data sources*; (iv) there is a lack of means for the *identification* of relevant data sources and meaningful ways to *automatically integrate them.*

Gathering data from multiple sources and performing data analysis, integration, and visualization tasks is hence a cumbersome process. End users cannot, yet, tap the full potential of available datasets, but rather have to rely on custom applications tailored to specific use cases or domains. This further inhibits integrated use of these data. Our goal is to address the discussed issues and enable non-expert users to collaboratively integrate data and obtain new insights.

To this end, this paper introduces a mashup platform in which semantic web and mashup concepts are combined to facilitate data integration for non-expert users in a flexible and efficient manner. We separate complex data integration tasks into reusable modular functions, which are encapsulated in high-level user interface blocks, i.e., the so called Linked Widgets. Based on that, users lacking programming skills can visually connect widgets to create mashup-based data integration applications. We lift non-semantic data to a semantic level at runtime and add explicit semantics to the input and output data of Linked Widgets. Thus we enable users to link disparate data sources, address data heterogeneity, and enrich data from one source with data from other sources to foster new insights.

An innovative aspect of our work is the new model of *semantic*, *distributed*, and *collaborative* mashups. There is already a body of work related to semantic and collaborative mashups; however, to the best of our knowledge, there is no research on *mashups* assembled from components that are *distributed* among different nodes (e.g., sensors, embedded devices, mobile phones, desktop computers, servers) to collect and integrate data. In our approach, mashup applications can be composed of both *client* and *server* Linked Widgets. *Client widgets* are executed in the local context of a web browser environment. *Server widgets* can be executed as native applications on various platforms, including personal computers, cloud servers, mobile devices, or embedded systems. *Server widgets* can be used to contribute data from the node they are deployed on to one or multiple mashups. They can also make use of the computing resources of its node to continuously process data in the background. This architecture allows stakeholders to expose their private data in a controlled manner by contributing *server widgets* as functional black boxes. This efficiently facilitates collaborative ad-hoc data integration involving multiple stakeholders that contribute data and computing resources.

We have implemented our concepts in a prototype platform, which is available at http://linkedwidgets.org. The data including mashups and semantic models of all widgets is published into the Linked Open Data cloud. It can be accessed via the SPARQL endpoint at http://ogd.ifs.tuwien.ac.at/sparql.

The remainder of this paper is organized as follows. Section 2 discusses related work; Sect. 3 introduces our platform for semantic, distributed, and collaborative mashups; Sect. 4 illustrates the applicability by means of an example use case and Sect. 5 introduces five mashup patterns (i.e., collaborative, persistent, distributed, streaming, and complex mashup pattern) applicable for various use cases. Section 6 concludes the paper with a discussion of findings.

2 Related Work

To facilitate data integration, researchers have been developing mashup-based tools and frameworks for years. Examples include mashArt [2], Intel Mash Maker [4], Microsoft Popfly [7], Exhibit [9], ResEval Mash [10], Apatar[2], MashQL [11], DERI Pipes [13], Information Workbench[3], Husky[4], Vegemite [14], Super Stream Collider [15], Yahoo! Pipes [16], Damia [17], Presto[5], Google Mashup Editor [18], Mashroom [20], and Marmite [21]. Many of them are geared towards end users and allow them to efficiently create applications by connecting simple and light-weight components. Various surveys [1,3,5,6,8] have been conducted to categorize, evaluate, and identify the limitations of these mashup tools.

A limited number of frameworks, such as Super Stream Collider [15], DERI Pipes [13], and MashQL [11], aim at semantic data processing. These frameworks, however, do not leverage semantic web techniques to facilitate automatic data integration for non-expert users [8], neither do they provide mechanisms to integrate semantic with non-semantic data. Thus our objective is first to focus on semantic mashups and overcome the inherent limitations. We therefore design semantic models for mashup components and leverage the semantics to foster mashup-based data exploration and integration. The input data of mashup components can be available in different formats (e.g., CSV, XML, JSON, RDF), but the semantic models impose the semantic format on the output data and hence tackle data heterogeneity.

The surveys show that it is difficult for non-expert users to use the mashup frameworks for composition and integration. On the one hand, a high-level and problem-oriented framework is easier to use than a low-level one, because it does not require users to be familiar with special technological and programming concepts. On the other hand, we need a generic framework that can deal

[2] http://www.apatar.com/ (accessed October 7, 2016).
[3] http://www.fluidops.com/en/portfolio/information_workbench/ (accessed October 7, 2016).
[4] http://www.husky.fer.hr/ (accessed October 7, 2016).
[5] http://mdc.jackbe.com/enterprise-mashup (accessed October 7, 2016).

with the increasing number of heterogeneous web resources rather than one tailored towards specific problems and resources. These design objectives lead to a trade-off [1]. A versatile mashup framework typically consists of a large number of predefined components; users generally have a clear idea of what they are trying to achieve, but they do not know which components they need and how to correctly combine them in order to reach their goal. This "*simplicity and expressive power*" trade-off [1] is a challenging issue.

The two surveys [1,6] discuss the importance of the communities behind the frameworks in relation to the success of end-user development tools. An open and collaborative model – which ties together three stakeholders (i.e., data publishers, developers, and end users) – enables each stakeholder to contribute and share their work to the open data community. Based on available data sources provided by different data publishers, developers are encouraged to create and deploy mashup components that are free to use or reuse. By combining such components, users can create mashup applications to work with open data in a dynamic and creative manner. The applications finally can be shared or reconfigured among the communities fostering reusability. However, to the best of our knowledge, current research focuses on user communities only and allows them to share, comment, or rank their mashups. There is no mashup-based data integration framework that can facilitate collaborative work among users, data publishers, and developers in order to encourage widespread use of (linked) open data.

The survey [3] discusses the challenge to integrate data that is stored in different devices, but is not available on the web, yet. In the literature so far, there is no research on *mashups* assembled from components that are *distributed* among different nodes (e.g., sensors, embedded devices, mobile phones, desktop computers, servers) to collect and integrate data. Such *distributed mashups* facilitate *collaborative* data integration in which each stakeholder contributes their data and computing resources to the shared processing flow.

3 Proposed Mashup Approach

3.1 Architecture

The Linked Widgets platform architecture is illustrated in Fig. 1. Linked Widgets extend the concept of standard web widgets[6] with explicit semantics. In particular, the semantics of their inputs and outputs as well as of the transformations they perform are explicitly described. Linked Widgets consume and produce Linked Data and may integrate data from sources such as raw data in CSV, XML, JSON, or HTML, data collected from databases, and data fetched from APIs and cloud services.

Based on their execution mode, widgets can be classified as *client* or *server widgets*. From a functional point of view, we can furthermore categorize them into (i) *data widgets*, (ii) *processing widgets*, and (iii) *visualization widgets*. Linked

[6] http://www.w3.org/TR/widgets/ (accessed October 7, 2016).

Widgets are highly reusable and can be parameterized to create mashups that contain data, application logic, and presentation layers.

Client widgets are executed on the client side, i.e., they use client memory and processor resources; data is collected and processed at runtime in the browser. The server hosting the *client widgets* is not necessarily the platform server, i.e., a mashup can combine widgets hosted on different servers. This feature makes the platform flexible and allows external parties to host widgets on their own infrastructure. *Client widgets* are easy to develop and necessary for a lightweight and scalable mashup platform. However, their capabilities are restricted by the web browser execution environment.

Server widgets shift the execution function from the browser environment to standalone application environments on desktop computers, mobile phones, tablets, sensor, or embedded systems. *Server widgets* provide the following benefits: (i) They can act as a data connector to obtain and provide data on different services, devices, or systems for a mashup; (ii) they facilitate collaborative use cases where each participant contributes *data* or *processing widgets* to a shared mashup; (iii) because their computing tasks are performed within the hosting devices, similar to *client widgets*, they reduce the platform server load; (iv) they can run persistently in the background to collect or process data for data monitoring or data streaming applications; (v) *server widgets* deployed on powerful servers are capable of processing large volumes of data over extended time periods.

By creating a connection between an output terminal of a widget and an input terminal of another widget, users can model data flows and integrate data without any programming skills. Internally, we use JSON-LD[7] for the exchange of RDF data between widgets.

A web-based *collaborative mashup editor* forms the core of the data integration architecture. Multiple mashup users can compose and execute mashups simultaneously and collaboratively and integrate private data with publicly available data sources. Mashup creators can locate available widgets using *semantic widget search* and group them into collections. They then connect those widgets to build mashups. The platform supports them in this process by suggesting and enforcing valid connections by means of the *terminal matching* module. In addition, the *automatic mashup composition* module can automatically compose a complete mashup from a widget, or a complete branch that consumes or provides data for a specific input or output terminal. Built on top of the *automatic mashup composition* module, the *tag-based automatic composition* module allows users to compose mashups through structured text input.

The resulting mashups fall into three categories: (i) *local mashups*, which consist exclusively of client widgets, (ii) *distributed mashups*, which consist entirely of server widgets, except for the final visualization widget(s), (iii) and *hybrid mashups*, which make use of both client and server widgets. A *local mashup* does not use any resources of the platform server, because it is executed completely inside the client browser. This implies that intermediate and final data

[7] http://www.w3.org/TR/json-ld/ (accessed October 7, 2016).

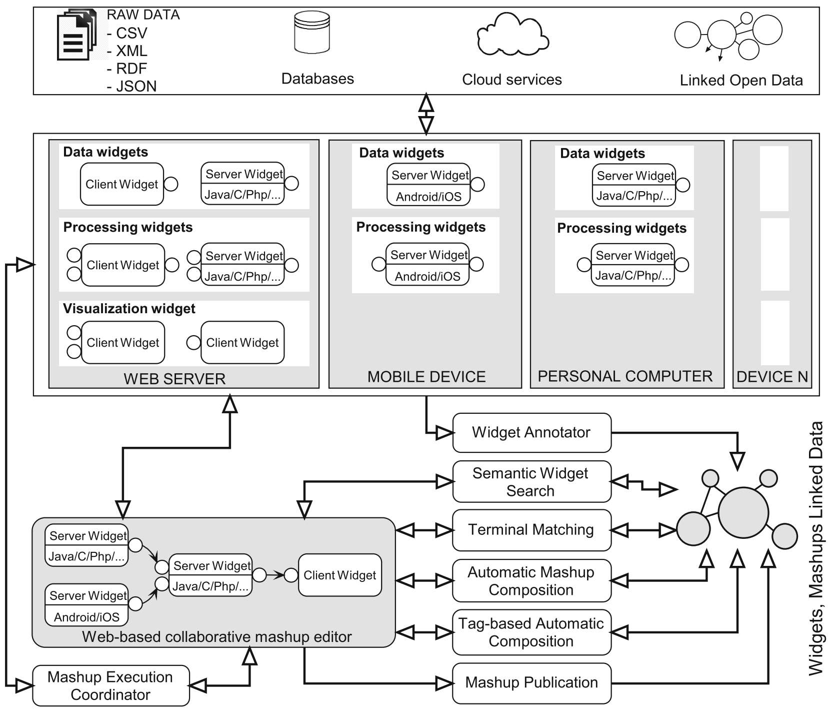


Fig. 1. Linked Widgets platform architecture

are lost as soon as the user closes the web browser. In contrast, widgets in *distributed mashups* are executed remotely as persistent applications; their output can hence be accessed at any time. *Hybrid* and *distributed mashups* can involve multiple nodes, each executing individual server widgets. This is highly useful, for instance, for streaming data scenarios where data must be collected and processed continuously over a longer period of time.

The *mashup execution coordinator* is a key component that enables widgets to communicate. We design *local*, *remote*, and *hybrid* protocols for the three types of mashups. Based on the publish/subscribe model, the protocols facilitate efficient communication between independently developed widgets while minimizing the platform server load.

Complete mashups can be published by users on their website by means of the *mashup publication* module. A published mashup shows the final *visualization widget* only and hides all previous data processing steps from the viewer. The mashup itself can also be encapsulated as a new *data widget*.

For detailed information on client widgets, server widgets, their semantic models, and three protocols that facilitate widget communication during execution please refer to our previous work [19].

3.2 Rapid Mashup Creation

A large and fast-growing number of ready-to-use datasets and services are available; the Linked Widgets platform is not tailored towards particular datasets, but integrates data from arbitrary sources. Available widgets can also be combined with widgets that may become available in the future; currently available data can be merged with future data without modifying existing widgets. The platform facilitates data integration in a rapid and flexible manner as we can simply add or remove a data source to enable new use cases.

First of all, to simplify the mashup development process, based on the explicit input and output semantic model of Linked Widgets, we develop a model matching algorithm (which is implemented in the *Terminal Matching* module) to validate the links between widgets. This allows non-expert users to discover all widgets that can provide data to or consume data from input and output terminals of a widget. Let i and o denote the root nodes of the input and output tree models (cf. [19]), respectively. There are three conditions for matching input and output models: (i) the RDF classes of i and o must be identical, or the RDF class of i must be a subclass of that of o; (ii) any child of i must correspond to a child of o (i.e., the set of properties required by the input must be a subset of properties provided by the output); (iii) recursively, the data model of the input object property must match with the data model of the corresponding output object property.

Leveraging the model matching algorithm, we design an automatic mashup composition algorithm to identify all possible *complete* mashups from a given set of Linked Widgets. A *complete* mashup is a set of widgets and the links between their input and output terminals, such that (i) all links are valid, (ii) all terminals must be wired, and (iii) each output terminal is linked to exactly one input terminal. To this end, in the following, we reduce the mashup composition algorithm to a *find all cycles* algorithm.

From a given set of widgets (e.g., $\{W_1, ...W_7\}$), we construct a directed graph illustrated in Fig. 2. For every processing widget that has n input terminals, we add $(n-1)$ virtual terminals (e.g., V_1^5 is the only virtual terminal of W_5). The vertex set then consists of all input, output, and virtual terminals of all widgets. We need to create virtual terminals to be able to apply the Johnson algorithm which is described in the following.

The edge set consists of four subsets: (i) a set of all valid links E_1 between input and output terminals, which are discovered by the *Terminal Matching* mechanism (e.g., $E_1 = \{(I_1^2,O^1), (I_2^5,O^2), (I_1^5,O^3), (I_1^5,O^4), (I_1^6,O^5), (I_1^7,O^5)\}$); (ii) a set of internal edges E_2 that link input, output, and virtual terminals of every single processing widget (e.g., $E_2 = \{(O^2,I_1^2), (O^4,I_1^4), (O^5,I_1^5), (V_1^5,I_2^5), (O^7,I_1^7)\}$); (iii) a set of edges E_3 that link all output terminals of all data widgets to all virtual terminals (e.g., $E_3 = \{(O^1,V_1^5), (O^3,V_1^5)\}$); and (iv) a set of edges E_4 that link all output terminals of data widgets to all input terminals of all visualization widgets (e.g., $E_4 = \{(O^1,I_1^6), (O^3,I_1^6)\}$).

Next, we can use the Johnson algorithm [12] to identify all cycles of the constructed graph. Its time complexity is bounded by $O((n+e)(c+1))$ and its

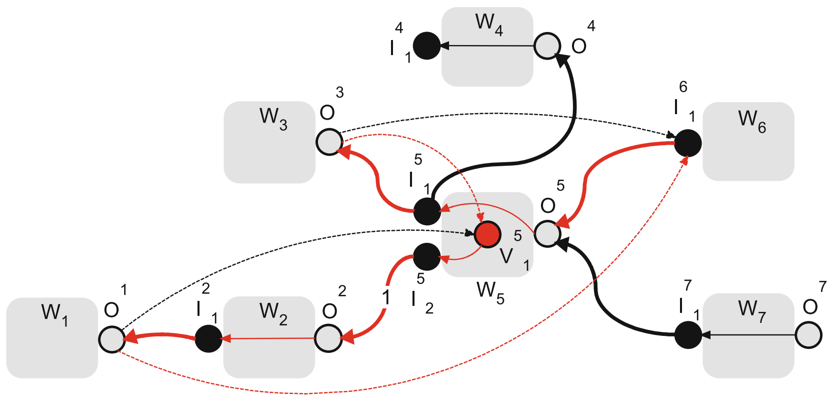


Fig. 2. Example graph corresponds to a set of widgets

space complexity is bounded by $O(n+e)$, where there are n vertices, e edges, and c elementary circuits in the graph. Finally, we apply post-processing steps and return a list of *complete* mashups indentified by the algorithm. For example, the cycle $\{I_1^6, O^5, I_1^5, O^3, V_1^5, I_2^5, O^2, I_1^2, O^1, I_1^6\}$ leads to a complete mashup of five widgets $\{W_1, W_2, W_3, W_5, W_6\}$ by removing all virtual terminals and keeping all edges in the first subset of edges E_1 only. However, the cycle $\{I_1^6, O^5, I_1^5, O^3, I_1^6\}$ does not yield a mashup, because I_2^5 of the involved widget W_5 is unlinked. It will therefore not appear in the final results.

4 Mashup Demo

4.1 Description of the Mashup Demo

For the mashup demonstration, consider the need to integrate data from different Excel and Google spreadsheets in an enterprise context. The typical process to achieve this goal is to download all spreadsheets then copy, delete columns, and create formulas to aggregate the data. These tedious tasks take up a lot of time and have to be repeated whenever the source data changes. To address this issue, we show that the Linked Widgets platform is capable of performing these tasks rapidly, flexibly, collaboratively, and in a distributed manner across groups of users and their multiple devices. The example collects, combines, and visualizes ice-cream sales data from a series of points of sale (POS) and relates it with the weather conditions obtained from the Wunderground API services[8].

To this end, we developed the following Linked Widgets: (i) *Google Sheet:* this client and data widget analyzes a sheet stored in the Google Drive cloud and transforms it into a W3C data cube[9], which is the semantic representation of the input sheet. (ii) *Local Spreadsheet:* this server and data widget runs on a

[8] https://www.wunderground.com/ (accessed October 7, 2016).
[9] https://www.w3.org/TR/vocab-data-cube (accessed October 7, 2016).

mobile phone or a desktop computer to collect local data. Similar to the *Google Sheet* widget, it receives a sheet and returns a W3C data cube. It allows users to directly add data from their devices to a mashup without the need to upload data to the web. (iii) *Cube Merger:* this server and processing widget receives a number of W3C data cubes as input, merges them, and outputs a data cube which includes the combined input data. It requires that all input cubes have the same dimensions and measures. (iv) *Aggregation:* based on the semantics of the input cube data, this client and processing widget aggregates data along the dimensions of the input data cube. It supports the aggregation functions *sum* and *average*. It is a generic widget in the sense that its interface is automatically generated based on the input data. (v) *Filter:* this client and processing widget calculates and returns a slice of the input data cube; to this end, users set the fixed values on one or multiple dimensions of the cube. (vi) *Temperature Enrichment:* this client and processing widget uses weather data to automatically enrich its input data cube if the cube contains time-dependent data. The output data is a new cube with temperature added as a new measure for each input date. (vii) *Google Chart:* this visualization widget analyzes the input data cube and displays it in a chart.

4.2 Description of the Preparation Needed to Make Approach Ready for Demo

The mashup demo engages a group of users, that is, the branch managers of the respective POSs. They need a mobile phone, tablet or desktop computer and a web browser to create, access, and run mashups from the platform.

For users who would like to contribute a private spreadsheet to the shared mashup, they need to download the *Local Spreadsheet* widget[10]. In our demo, it is realized as a desktop application. It extends an abstract desktop widget that implements the Linked Widgets protocol to communicate with other widgets. To develop the *Local Spreadsheet* widget, we need to implement the *execution* function only.

4.3 Description of the Demo Flow

The life cycle of a new mashup is initiated by a *host user*, who can then choose from a number of available Linked Widgets (listed in Sect. 4.1) to include in the palette of the mashup. To add a private widget, the user can provide the name and the URL of the widget. Then, after dragging and dropping a widget, she (or other users) can query the platform to (i) list all complete mashups that contain the widget, (ii) construct all possible complete *mashup branches* for a particular input or output terminal of the widget, and (iii) construct all complete *mashup branches* for all input terminals of the widget. A *mashup branch* is a part of a complete mashup that consumes (provides) data for a specific output (input) terminal. These features are available in the user interface via a click on the

[10] http://linkedwidgets.org/serverwidgets/ExcelSheet.jar (accessed October 7, 2016).

question mark symbol in the widget bar, and the symbol that appears when we hover a terminal.

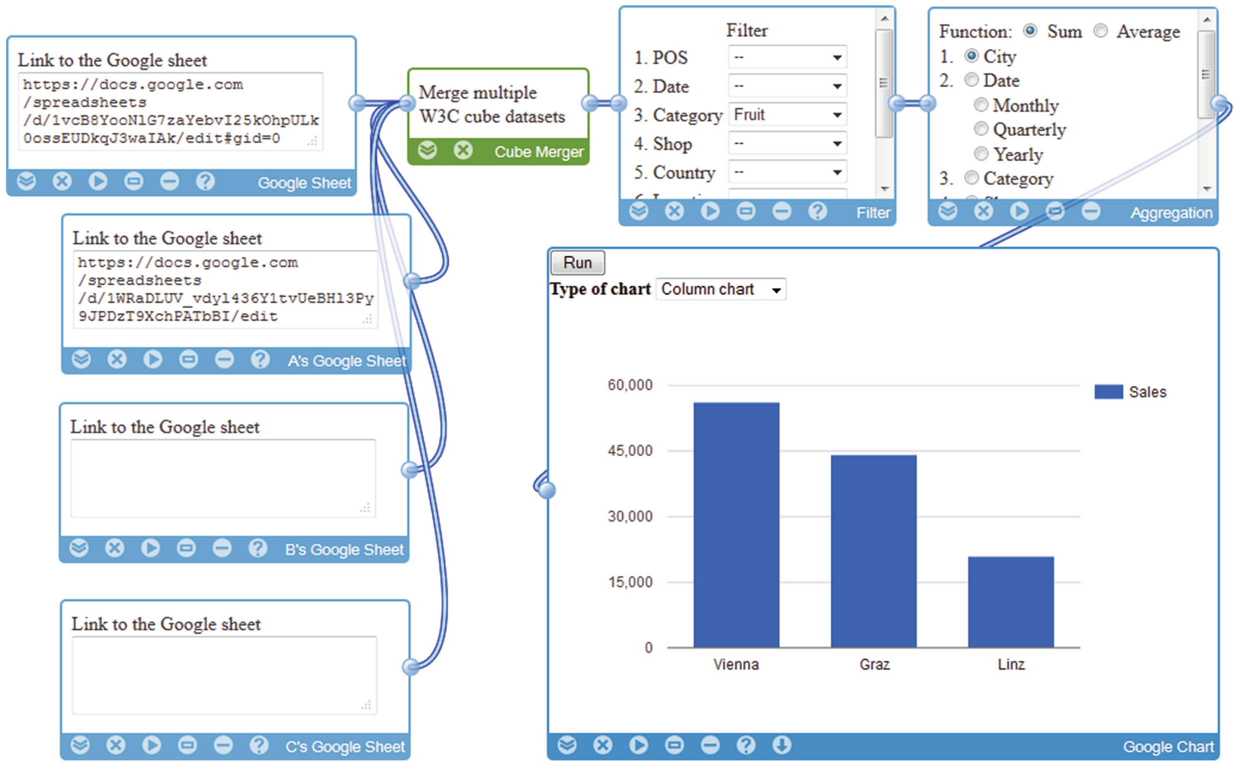
The first example mashup involves three Austrian POSs, which are located in Vienna, Graz, and Linz, respectively. Data are stored in two types of Google spreadsheet: (i) a point of sale spreadsheet that contains id, name, latitude, longitude, city, country of the POS; and (ii) three sales spreadsheets, each listing the number of items sold per day per category at that point of sale. Whereas the point of sale spreadsheet is provided by the *host user*, the three sale spreadsheets of POS A, B, C are provided by the local branch managers, who update the data every day by adding new rows into the spreadsheet.

The *host user* first adds a *Google Sheet*, a *Cube Merger*, a *Filter*, an *Aggregation*, and a *Google Chart* widget and links their input and output terminals. She then sends an automatically generated token that identifies the mashup instance to everyone in the group. Then, the recipients enter the token to load the respective mashup instance and start working collaboratively. All operations such as adding (removing) a widget to (from) the mashup, connecting widgets, resizing a widget, etc. are propagated and synchronized into all editors.

As soon as every local branch manager drags and drops their *Google Sheet* widget and connects it with the *Cube Merger* widget, the *host user* runs the last widget of the mashup, which is the *Google Chart* widget in our example. The three managers now enter the link to their *Google Sheet* widget; Fig. 3 illustrates the mashup shown on the screen of the manager of POS A. As soon as she executes her *Google Sheet* widget, everyone in the group can immediately see the updated data in the chart. An arbitrary participant can remove her widget and leave the collaborative mashup group, or the host user can invite additional users to join the group.

In this example mashup, the options inside the *Filter* and *Aggregation* widgets are generated automatically based on the input data cube. By changing the automatically generated options inside the *Filter* and *Aggregation* widgets, different types of analyses can be quickly performed. For example, collaborators can "compare all-time sales of all POS", "compare sales of all POS in 2014", "compare sales of fruit, milk, and chocolate items of POS A in 2014", "compare aggregate sales in different countries or cities", or "compare sales of fruit items in different cities" as shown in Fig. 3.

Because Linked Widgets within a mashup can be contributed and executed by multiple independent actors, it is important to log the contribution of each to the resulting integrated data. We therefore store provenance data when a mashup is executed. The provenance data contains information about each widget's contribution to the final result. The generated provenance trail includes information on who executed a particular widget, the timestamps when processing started and finished, etc. The provenance is shown when users press the download button in the widget bar. For example, part of provenance of the *Google Sheet* widget contributed by the host user (e.g., Alice) is shown in Listing 1; she acts on the behalf of the developer of the widget to produce a data cube; the data is associated with a signature for reproducibility checking.


Link to the Google sheet
https://docs.google.com/spreadsheets/d/1vcB8YooN1G7zaYebvI25kOhpULk0ossEUDkqJ3waIAk/edit#gid=0
Google Sheet

Link to the Google sheet
https://docs.google.com/spreadsheets/d/1WRaDLUV_vdyl436Y1tvUeBH13Py9JPDzT9XchPATbBI/edit
A's Google Sheet

Link to the Google sheet
B's Google Sheet

Link to the Google sheet
C's Google Sheet

Merge multiple W3C cube datasets
Cube Merger

Filter
1. POS --
2. Date --
3. Category Fruit
4. Shop --
5. Country --
Filter

Function: [✓ Sum] Average
1. [✓ City]
2. Date
Monthly
Quarterly
Yearly
3. Category
Aggregation

Run
Type of chart Column chart
Google Chart

Fig. 3. Collaborative mashup shown on the screen of the manager of Point of Sale *A*

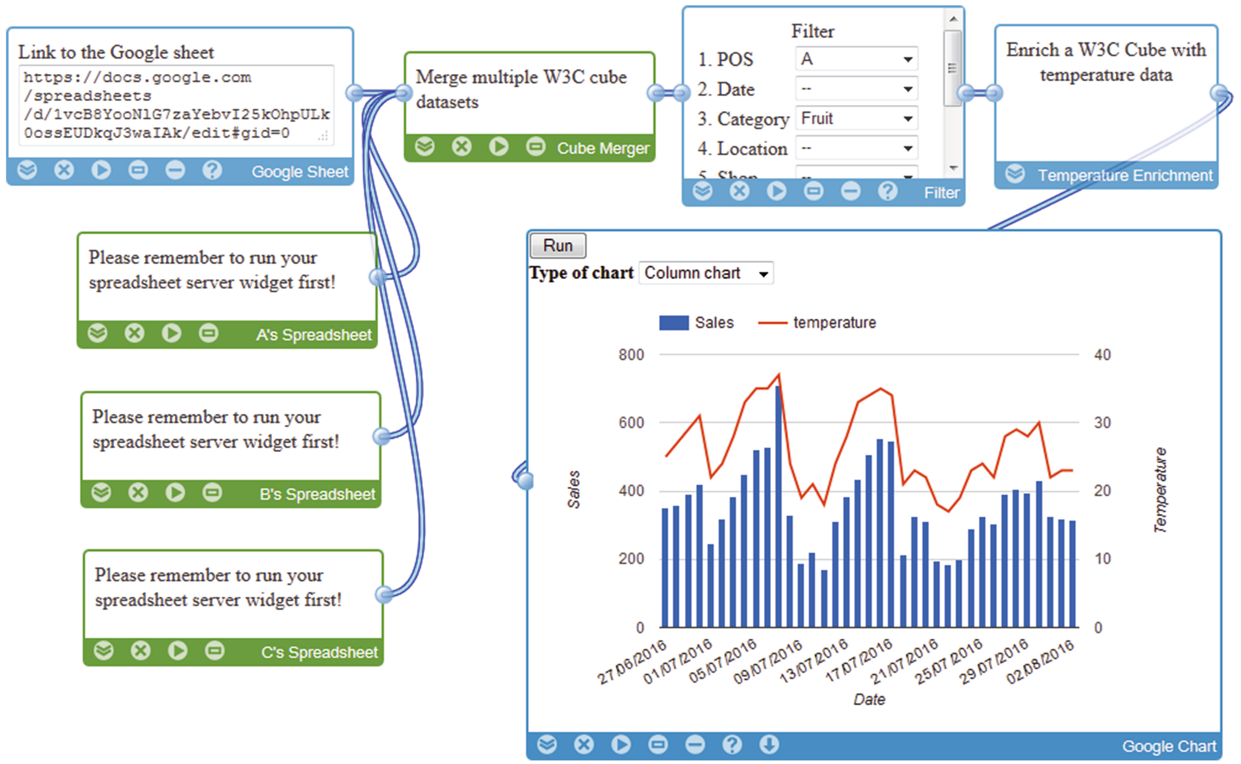
```
{
   "@context": {...},
   "@type": ["prov:Entity", "void:Dataset"],
   "@id": "http://.../ontology/resource/62e2ba4f9abc75a2b7f1f87cdfeac7b3",
   "signature": "62e2ba4f9abc75a2b7f1f87cdfeac7b3",
   "wasAttributedTo": {
      "@id": "http://linkedwidgets.org/ontology/resource/userAlice",
      "actedOnBehalfOf": {"@id": "http://.../resource/unitProviderDat"}
   },
   "wasGeneratedBy": {
      "@id": "http://.../Execution-nicOtMWOHs",
      "@type": ["prov:Activity", "http://.../ProcessingUnitExecution"],
      "startedAtTime": "Sun August 21 2016 02:06:22 GMT+0200",
      "endedAtTime": "Sun August 21 2016 02:06:23 GMT+0200",
      "hasParameter": [
         {
            "@type": "http://linkedwidgets.org/ontology/Parameter",
            "@id": "_:n1",
            "name": "txtLink",
            "value": "https://docs.google.com/spreadsheets/..."
         }
      ],
      ...
   }
}
```

Listing 1. Provenance for execution of the host user's Google Sheet widget

The sales data can be obtained either from a cloud storage service (e.g. Google Drive), or it originates from a diverse set of devices. For example, in Fig. 4, rather than using Google sheets, the three managers now add local spreadsheets stored on their local computers since they do not want to upload the files to the web (e.g., as sales data is sensitive). Next, each manager selects the file and runs his *Local Spreadsheet* widget (cf. Fig. 5) as a standalone application on his desktop computer. This server widget then generates a URL of his private widget so that the user can add it to the collaborative mashup. Whenever the user modifies his spreadsheet, he re-executes his *Local Spreadsheet* widget to update the result of the whole mashup. Moreover, in this example, we use a *Temperature Enrichment* widget to combine sales data with weather data and deduce the impact of temperature on sales figures for each POS; Fig. 4 illustrates this setup fruit items of POS A.

To facilitate collaborative data integration, we offer the *Delegating* widget (cf. Fig. 6) that can persistently be connected with an instance of a server widget. When we run this delegating widget, it subscribes to the "*returning output*" event of the server widget instance whose token is specified in the input box. The delegating widget acts as an agent for the server widget instance, meaning that as soon as the server widget returns its new output data, the delegating widget receives the data and immediately returns the same result.

With the delegating widget, the output data of multiple groups can dynamically be integrated with each other. To this end, each group contributes a *Delegating* widget to a new mashup. Each *Delegating* widget is connected with the last server widget of each group's mashup (i.e., the *Cube Merger* widget in our demo). For example, in Fig. 6, we have three groups (Austria, France, and Italy). The group Austria is described in the previous mashup; it includes three local branch managers from Vienna, Graz, and Linz. As soon as a participant of an arbitrary group (e.g., the Vienna manager) updates his data, the final integrated data collected from all groups in all countries is updated immediately, too.


Link to the Google sheet

https://docs.google.com/spreadsheets/d/1vcB8YooN1G7zaYebvI25kOhpULk0ossEUDkqJ3waIAk/edit#gid=0

Merge multiple W3C cube datasets

Filter
1. POS: A
2. Date: --
3. Category: Fruit
4. Location: --

Enrich a W3C Cube with temperature data

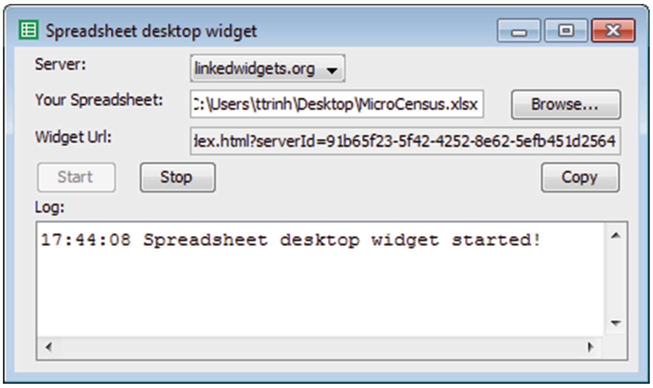
Please remember to run your spreadsheet server widget first!

Please remember to run your spreadsheet server widget first!

Please remember to run your spreadsheet server widget first!

Type of chart: Column chart

Fig. 4. A mashup composition showing sales data from desktop spreadsheets combined with weather conditions


Spreadsheet desktop widget
Server:
linkedwidgets.org
Your Spreadsheet:
:\Users\ttrinh\Desktop\MicroCensus.xlsx
Browse...
Widget Url:
iex.html?serverId=91b65f23-5f42-4252-8e62-5efb451d2564
Start
Stop
Copy
Log:

```
17:44:08 Spreadsheet desktop widget started!
```

Fig. 5. Server widget runs as standalone application on a desktop

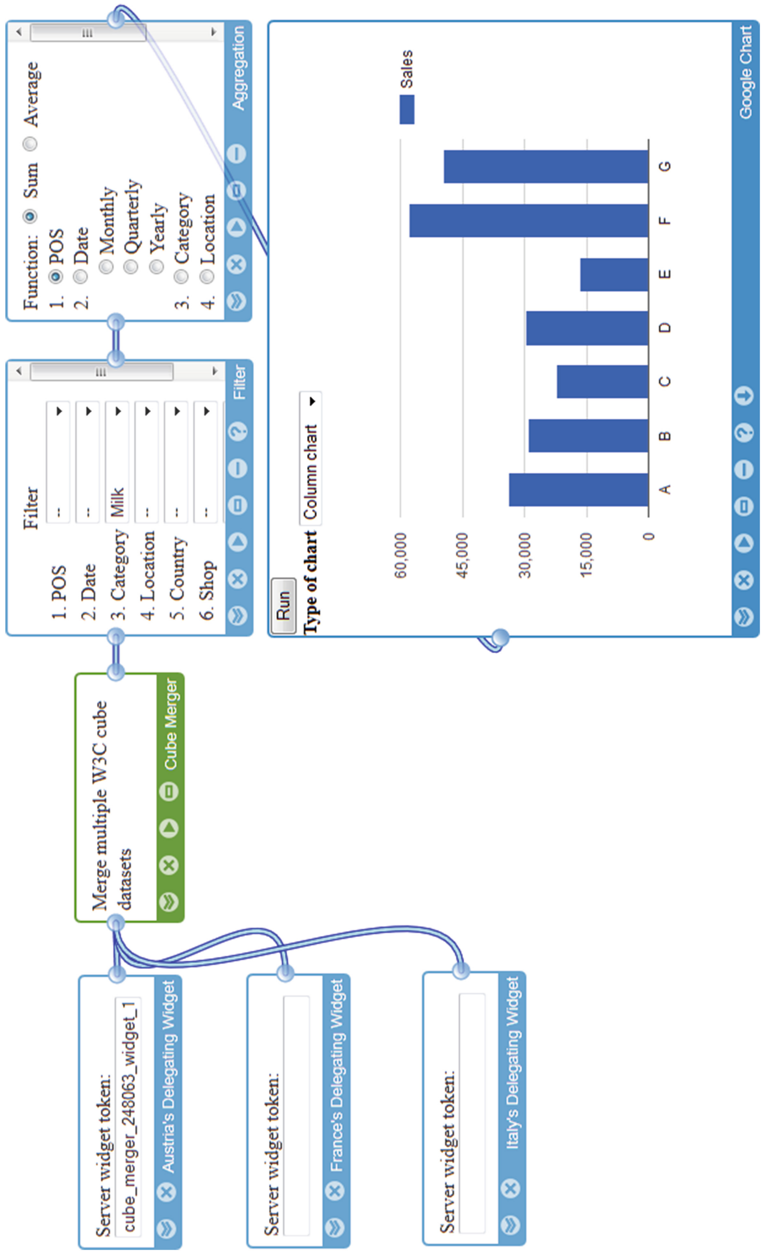
The *Delegating* widget is designed for collaborative mashups as follows: (i) It first allows a participant to prepare his own data for a collaborative mashup. Rather than exposing a large volume of data, he can extract a relevant part of the data only and perform pre-processing tasks in a private mashup. He then contributes the data to the collaborative group by using a delegating widget. This not only makes data integration more secure (as his data may be sensitive it should not be visible to others), but also speeds up the execution of the whole mashup (as irrelevant parts of data are already removed). The private mashup is not a part of the collaborative mashup, but its final output data is used in the collaborative mashup via the delegating widget. (ii) The delegating widget allows a participant to hide a branch of the collaborative mashup. The Linked Widgets platform enables anyone to develop his own widget; he then can contribute it to the public community or keep it private. A private widget can be used in a collaborative mashup; it is visible to everyone in the shared mashup screen. The hiding feature is hence useful if a participant does not want to expose his private widget used in the collaborative mashup to others. Moreover, it simplifies the overall mashup; each should only see the relevant part of the mashup rather than the part that he can ignore or cannot control.

There are many ways to combine *client widgets* and *server widgets* to develop mashups that are useful for a variety of scenarios. In the next section, we introduce five patterns to compose collaborative, persistent, distributed, streaming, and complex mashups, respectively.

5 Mashup Patterns

5.1 Collaborative Mashups

Definition. A collaborative mashup is a type of mashup application that is edited and/or operated by more than one user at the same time.


Server widget token: cube_merger_248063_widget_1

Server widget token:

Server widget token:

Merge multiple W3C cube datasets

Filter

1. POS	--
2. Date	--
3. Category	Milk
4. Location	--
5. Country	--
6. Shop	--

Function: [✓ Sum] Average

1. [✓ POS]
2. Date
 Monthly
 Quarterly
 Yearly
3. Category
4. Location

Type of chart Column chart

Fig. 6. A Mashup integrating data from three groups

Use Case. Collaborative mashups are useful for a group of users. They can collaboratively define a data processing flow and each participant can supply their input data independently, while all participants immediately get a live representation of the combined data. In this model, tedious and repetitive manual data integration processes (e.g., data cleansing, data uploading, update notifications) are encapsulated in widgets.

Consider, for example, the simple task of scheduling a meeting between users whose calendars are spread across computers, mobile phones, Cloud services, etc. A widget-based collaborative workflow would allow participants to contribute their calendar using *server widgets* such as locally executed Apps or Cloud-based calendar widgets. They can then simply merge their calendar widgets in a collaborative mashup to identify available timeslots.

Pattern. The Linked Widgets collaborative mashup pattern is presented in Fig. 7. It involves at least one server widget (e.g., W) and two users; the two users are responsible for their two widgets (e.g., W_1 and W_2), which can be either client or server widgets.

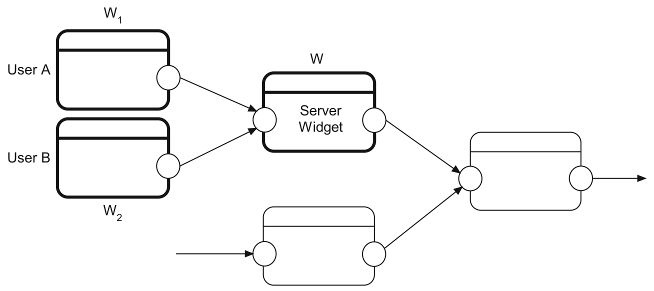


Fig. 7. Collaborative mashup pattern (pattern widgets highlighted with thick borders)

To execute the collaborative mashup, first, an arbitrary participant triggers the run action in the visualization widget. This will also trigger the execution of all preceding widgets (except W_1 and W_2), one after the other. The server widget W, however, cannot run, yet, because it still waits for the output data of W_1 and W_2.

In the next step, the two users set the values of their widget's parameters and run them. As soon as both widgets return output data, the server widget W performs the data integration task and returns the result to its succeeding widget. Whenever W_1 or W_2 submits new output data, the final result is recalculated and immediately presented to every participant in the synchronized visualization widget.

A participant can leave the collaborative mashup at any time. To this end, she removes her widget from the mashup so that the input data of the server widget W is reset, and her private data is removed from the mashup. On the other hand, a new participant can easily join the collaborative mashup by entering the collaborative token to load the mashup and adding her private widget into the collaborative mashup editor.

5.2 Persistent Mashups

Definition. A persistent mashup is a type of mashup application that can continuously run in the background and maintain its status and intermediate data.

Use Case. A persistent mashup can be used for data integration tasks that are typically time-consuming (e.g., statistical analysis, data analysis, and data reporting). To this end, the mashup creator composes a mashup and submits her input data. At any time, she can reopen the mashup to check the current status and result. As she does not host and run the mashup on her device, she can manage a mashup that performs expensive calculation tasks even with a slow client device.

Pattern. Figure 8 shows the Linked Widgets persistent mashup pattern. It contains a server widget placed before a visualization widget, which is a client widget. Because the server widget performs the processing tasks in the respective server rather than the browser environment, it can persistently maintain the calculation data and the intermediate mashup data.

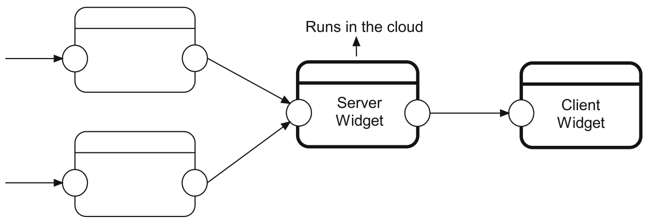


Fig. 8. Persistent mashup pattern

As soon as the mashup is reopened, the client widget requests the latest output data from the server widget for visualization. The browser hence acts as a front-end tool that shows up-to-date data from the back-end processing.

5.3 Distributed Mashups

Definition. A distributed mashup is a type of mashup application in which the involved widgets are hosted in distributed nodes and devices.

Use Case. Distributed mashups facilitate, for instance, integration of sensor data from embedded devices. The data collector tasks run pervasively among the server widgets of distributed nodes. To facilitate data integration, those server widgets can clean, formalize, and convert the data before sending it to a central node where data is aggregated before finally being visualized in a client widget. The distributed model typically involves three types of nodes: (i) embedded devices (which provide input data), (ii) a powerful server (which processes data), and (iii) a personal device such as a mobile phone, tablet, or laptop for visualization. Distributed mashups, moreover, allow us to integrate data from different devices without the necessity to upload the data to a central point.

Pattern. Figure 9 depicts the widget combination pattern of distributed mashups. The involved server widgets are placed in an arbitrary position of the mashup. Based on the available programming language (e.g., Java, Python, Erlang, C++, etc.) of the hosting device, we implement the respective versions of the server widgets. By adding (removing) a widget into (from) a mashup, we can add (remove) the node into (from) the ad-hoc architecture. To this end, it is necessary that the device is connected to the internet so that widgets can communicate using the *remote protocol* (cf. [19]).

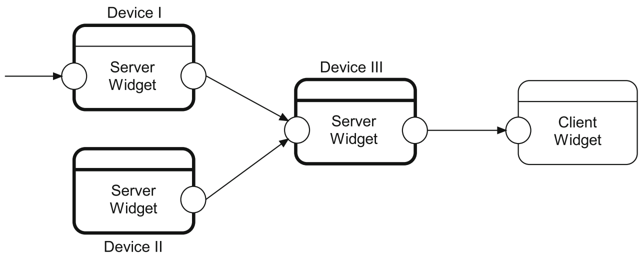


Fig. 9. Distributed mashup pattern

5.4 Streaming Mashups

Definition. A streaming mashup is a type of mashup application in which data flows continuously between two or more widgets.

Use Case. Streaming mashups can, for instance, be used for data monitoring applications. While current data is presented to users, new data is constantly generated, delivered, and aggregated for presentation. Due to the variety of real-time and streaming data sources available on the web (e.g., weather data, public transportation data, data on stock quotes), each can build her own application to process daily data and support decision making. For example, users can compose a mashup that collects the temperature, pressure, and wind speed at different places (e.g., her home, her work place) every minute, based on the weather conditions of the nearest stations. Users can then visualize the aggregated data in a chart, which is updated every minute.

Pattern. The Linked Widgets' streaming mashup pattern is illustrated in Fig. 10. At least one widget continuously runs and returns its output data. We use the term *streaming widget*, which can be either a client or a server widget. A streaming client widget can be used, but the streaming data flow of the mashup will be stopped once we close the browser; a streaming server widget should be used if we intend to make our streaming mashup persistent.

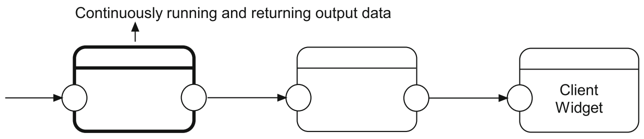


Fig. 10. Streaming mashup pattern

5.5 Complex Mashups

The purpose of classifying hybrid mashup patterns is to clarify and emphasize different aspects of mashups; there is no strict boundary between *collaborative*, *persistent*, *distributed*, and *streaming* mashups. We can combine these types in several ways. For example, we can construct mashups that continuously integrate streaming data from distributed sensors of multiple stakeholders.

6 Conclusions

A large number of data sources, APIs, services, and data visualizations are publicly available. However, non-expert users are not able to directly access, explore, and combine different sources due to their lack of skills and knowledge of data processing.

In this paper, we present a mashup platform for dynamic and automatic exploration and integration of heterogeneous data sources for non-expert users. To foster reusability and creativity, we modularize the functionality into Linked Widgets. Linked Widgets represent modules that users can recombine in order to create new applications. We make use of both client and server computing resources to create a powerful, extensible, and scalable data integration model. With server Linked Widgets, data processing tasks can be run persistently and be distributed among various devices. This is particularly useful for data streaming or data monitoring use cases.

The platform is built upon semantic web concepts. We impose a semantic format on widget outputs. This helps the platform to address data heterogeneity. Based on that, generic widgets such as *Aggregation*, *Filter*, and *Google Chart* can automatically adapt their interface by analyzing the semantics of their input.

We leverage the explicit semantics of Linked Widgets' input and output models to ease and simplify the mashup development process, e.g., by automatically composing meaningful mashups from a given set of available widgets.

Moreover, the platform (i) facilitates collaborative work among users, data publishers, and developers encouraging widespread use of data and (ii) integrates dispersed data stored on different devices and contributed by multiple stakeholders in a rapid manner. Hence, the platform is applicable to a variety of scenarios, such as scientific research, data journalism, enterprise data integration, or ad-hoc integration of web data by non-expert users.

The platform is still in its early stages of development and some limitations apply, which we intend to address in future work. First, we plan to extend the semantic model to include annotation of parameters that can be set in the user interface. Furthermore, we presently focus on collaborative, and distributed data integration scenarios in an open environment. A key issue for future work is to ensure privacy and secure data transfer in order to protect sensitive data. Finally, our research has focused exclusively on conceptual and technical aspects so far; a comprehensive user study as well as extensive performance and scalability testing will provide insights into important applied aspects.

A Mashup Feature Checklist

Mashup Type

- Data mashups
- Logic mashups
- User Interface (UI) mashups
- **Hybrid mashups**

Component Types

- **Data components**
- **Logic components**
- **UI components**

Runtime Location

- Client-side only
- Server-side only
- **Both Client and Server**

Integration Logic

- UI-based integration
- **Orchestrated integration (local and hybrid mashups)**
- **Choreographed integration (distributed server-side mashups)**

Instantiation Lifecycle

- Stateless
- **Short-living**
- **Long-living (persistent mashups)**

B Mashup Tool Feature Checklist

Targeted End-User

- Local Developers
- **Non Programmers**
- Expert Programmers

Automation Degree

- **Full Automation**
- **Semi-automation**
- **Manual**

Liveness Level

- Level 1 (Non-Executable Prototype Mockup)
- Level 2 (Explicit Compilation and Deployment Steps)
- **Level 3 (Automatic Compilation and Deployment, requires Re-initialization)**
- Level 4 (Dynamic Modification of Running Mashup)

Interaction Technique

- Editable Example
- Form-based
- Programming by Demonstration
- Spreadsheets
- Textual DSL
- Visual Language (Iconic)
- **Visual Language (Wiring, Implicit Control Flow)**
- Visual Language (Wiring, Explicit Control Flow)
- WYSIWYG
- Natural Language
- Other

Online User Community

- None
- Private
- **Public**

References

1. Aghaee, S., Pautasso, C.: End-user programming for web mashups. In: Harth, A., Koch, N. (eds.) ICWE 2011. LNCS, vol. 7059, pp. 347–351. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27997-3_38
2. Daniel, F., Casati, F., Benatallah, B., Shan, M.-C.: Hosted universal composition: models, languages and infrastructure in mashArt. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 428–443. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04840-1_32
3. Di Lorenzo, G., Hacid, H., Paik, H.Y., Benatallah, B.: Data integration in mashups. ACM Sigmod Rec. **38**(1), 59–66 (2009)
4. Ennals, R., Brewer, E., Garofalakis, M., Shadle, M., Gandhi, P.: Intel mash maker: join the web. ACM SIGMOD Rec. **36**(4), 27–33 (2007)
5. Fischer, T., Bakalov, F., Nauerz, A.: An overview of current approaches to mashup generation. In: Proceedings of the International Workshop on Knowledge Services and Mashups (KSM09), pp. 254–259. Citeseer (2009)
6. Grammel, L., Storey, M.A.: An End User Perspective on Mashup Makers. University of Victoria Technical Report DCS-324-IR (2008)
7. Griffin, E.: Foundations of Popfly: Rapid Mashup Development. Apress (2008)
8. Hendrik, A.A., Tjoa, A.M.: Towards semantic mashup tools for big data analysis. In: Linawati, M.M.S., Neuhold, E.J., Tjoa, A.M., You, I. (eds.) ICT-EurAsia 2014. LNCS, vol. 8407, pp. 29–138. Springer, Heidelberg (2014)
9. Huynh, D.F., Karger, D.R., Miller, R.C.: Exhibit: lightweight structured data publishing. In: Proceedings of the 16th International Conference on World Wide Web, pp. 737–746. ACM (2007)
10. Imran, M., Soi, S., Kling, F., Daniel, F., Casati, F., Marchese, M.: On the systematic development of domain-specific mashup tools for end users. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 291–298. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31753-8_22
11. Jarrar, M., Dikaiakos, M.D.: MashQL: a Query-by-diagram Topping SPARQL. In: Proceedings of the 2nd International Workshop on Ontologies and Information Systems for the Semantic Web, pp. 89–96. ACM (2008)
12. Johnson, D.B.: Finding all the elementary circuits of a directed graph. SIAM J. Comput. **4**(1), 77–84 (1975)
13. Le-Phuoc, D., Polleres, A., Hauswirth, M., Tummarello, G., Morbidoni, C.: Rapid prototyping of semantic mash-ups through semantic web pipes. In: Proceedings of the 18th International Conference on World Wide Web, pp. 581–590. ACM (2009)
14. Lin, J., Wong, J., Nichols, J., Cypher, A., Lau, T.A.: End-user programming of mashups with vegemite. In: Proceedings of the 14th International Conference on Intelligent User Interfaces, pp. 97–106. ACM (2009)
15. Nguyen, M.Q.H., Serrano, M., Le-Phuoc, D., Hauswirth, M.: Super stream collider-linked stream mashups for everyone. In: Proceedings of the Semantic Web Challenge co-located with ISWC 2012. Boston, US (2012)
16. Pruett, M.: Yahoo! Pipes, 1st edn. O'Reilly, Sebastopol (2007)
17. Simmen, D.E., Altinel, M., Markl, V., Padmanabhan, S., Singh, A.: Damia: data mashups for intranet applications. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1171–1182. ACM (2008)
18. Tony, L.: Creating Google Mashups with the Google Mashup Editor. Lotontech Limited (2008)

19. Trinh, T.D., Wetz, P., Do, B.L., Kiesling, E., Tjoa, A.M.: Distributed mashups: a collaborative approach to data integration. IJWIS **11**(3), 370–396 (2015)
20. Wang, G., Yang, S., Han, Y.: Mashroom: end-user mashup programming using nested tables. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009, pp. 861–870. ACM, New York (2009)
21. Wong, J., Hong, J.I.: Making mashups with marmite: towards end-user programming for the web. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 1435–1444. ACM (2007)