Paggr: Linked Data widgets and dashboards
Benjamin Nowack
semsol, Bielefelder Str. 5, 40468 Düsseldorf, Germany

A B S T R A C T

The Linked Data effort has led to a large number of interconnected and reusable datasets with standar-ized RDF interfaces. Paggr is a novel Web system that simplifies personalized aggregation and interaction with these structured information sources through SPARQL-driven widgets and dashboards. Paggr utilizes a URI-based drag and drop mechanism for end-users, and a collaborative, browser-based development environment for widget creators.

Keywords:
Linked Data
User interaction
Widget
Drag and drop
Personalization
Semantic web

1. Introduction

A central objective of the Semantic Web initiative is simplified data integration and repurposing. Most of the necessary standards are in place, as well as the amount of interesting datasets that can be accessed natively or through RDF wrappers. At least two ongoing challenges remain, however. One is the creation of visual interfaces that are easy to get used to, the other is the efficient development of semantic web applications and ad hoc mashups.

This paper presents Paggr, a system that builds on the common pattern of aggregating and displaying chunks of web information in HTML-based widgets. While today’s web widgets are mostly based on proprietary API calls, Paggr uses SPARQL operations instead. Paggr widgets (“Sparqlets”) can be organized in Wiki-like dashboards, and a browser-based development tool is provided that allows developers to collaboratively create widgets and write SPARQL scripts.

2. Example use case

Technically, RDF enables the unrestricted combination of information from different sources. Browsing and visualization tools are generic and kept separate from the standardized data. This paradigm moves utilization decisions from the publisher to the consumer. Unfortunately, this freedom can lead to a non-satisfying user experience when either the generic tools are not optimized enough for the task at hand, or when specific end-user requests cannot be turned into tailored software solutions.

Paggr tries to offer a compromise between the creation of custom solutions and using standard tools. By splitting application functionality into web-accessible and reusable scripts, Paggr applications can benefit from the related network effect. Each widget can build on other (standard) widget components to accelerate the creation of individual solutions. The web-based widget editor allows the addition of new options at run-time, user feedback can be considered more quickly.

During the European Semantic Web Conference (ESWC) 2009, a first public Paggr system was launched.1 It was optimized for conference-specific tasks such as exploring upcoming sessions, finding speakers from a selected company or papers about a certain topic, or keeping track of ESWC-related discussions on the Twitter and Identi.ca services (see Fig. 1). The main RDF data source was the Semantic Web Conference Corpus.2

Due to connectivity issues at the conference venue, the application was mainly used by people who were not attending the conference and very few personal dashboards were created. Therefore, it was not possible to gather and evaluate the application development approach behind Paggr in terms of user satisfaction. What could be tracked, though, was the development time for individual widgets and public dashboards. It required several days to implement and test the core system scripts (graph management, keyword search, resource type detection, resource label retrieval,

---

2 http://data.semanticweb.org.
etc.), but after this one-time step, the development time of widgets for the actual use case (Authors, Papers, Sessions, Organizations, Topics, Posts) decreased with each new component that could reuse existing procedures (see Table 1). Not surprisingly, including non-RDF sources which needed inference scripts for efficient querying (e.g. for the integrated Twitter/Identi.ca widget) took generally longer than data that did not have to be preprocessed.

3. System components and architecture

A Paggr application consists of two main components: a widget server, and a dashboard server. Dashboard and widget servers are loosely coupled (see Fig. 2). They communicate through SPARQL or simple RDF retrieval, which means that a dashboard front-end can include and run widgets from any public widget server.

Although SPARQLScript is a simple procedural scripting language and can be written by hand, each widget server provides a web-based “Sparqlet Builder” that simplifies the creation and editing of Sparqlets and scripts. Paggr’s developer tool is described in Section 4.1.

Dashboard servers process user requests, store application data, and execute widget definition code loaded from the associated widget server. Dashboard servers also take care of caching, dynamic (sub-)script inclusion, and privacy preferences. Fig. 3 shows the general architecture of a dashboard server.

4. Developer tools and widget interaction

Paggr dashboards are essentially mashups that aggregate and integrate information from distributed and interlinked semantic web sources. Due to the nature of Linked Data that usually combine a multitude of schemas and resource identifiers, widget scripts may have to normalize incoming data first, or need simple inference operations before information can be efficiently displayed to users. Sparqlets also have to work with non-predefined or non-stable datasets, and should adapt depending on user input. To make this possible, Paggr relies heavily on parameterized queries where placeholders can be filled with context-specific values, either resulting from user interaction, or from dashboard settings.

Table 1
Development time of a Paggr application (from scratch, graph management and system scripts will be reusable in other projects).

<table>
<thead>
<tr>
<th>Sparqllets</th>
<th>Time needed (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph management (load, delete, refresh, isExpired, etc.)</td>
<td>15</td>
</tr>
<tr>
<td>System scripts (type detection, query filters, etc.)</td>
<td>10</td>
</tr>
<tr>
<td>MicroblogSearch (search, infer, getMicrofeedHTML, etc.)</td>
<td>10</td>
</tr>
<tr>
<td>PhotoSearch (search, infer, getPhotofeedHTML, etc.)</td>
<td>6</td>
</tr>
<tr>
<td>Papers (query, list, summary)</td>
<td>5</td>
</tr>
<tr>
<td>Authors (query, list, summary)</td>
<td>2</td>
</tr>
<tr>
<td>Topics (query, list, summary)</td>
<td>0.5</td>
</tr>
<tr>
<td>Organizations (query, list, summary)</td>
<td>1</td>
</tr>
<tr>
<td>Sessions (query, list, summary)</td>
<td>1.5</td>
</tr>
<tr>
<td>Custom content (WikiMarkup + CSS)</td>
<td>0.5</td>
</tr>
<tr>
<td>Dashboard: Home</td>
<td>2</td>
</tr>
<tr>
<td>Dashboard: ESWC live</td>
<td>0.5</td>
</tr>
</tbody>
</table>
The following sections describe how developers can define parameterized Sparqlets, and how dashboard users pass parameters to widget scripts.

4.1. Sparqlet Builder

The Sparqlet Builder is a browser-based tool that lets developers create chained data processes without the need to learn a programming language. Fig. 4 depicts an example script that first calls a remote sub-routine, then sets an endpoint URL, followed by a query against the defined endpoint. The script builder provides a set of standard blocks, such as endpoint declarations, variable assignments, queries, function calls, control structures, and output templates. It is also possible to write raw SPARQLScript code. Widget scripts generally can

- retrieve RDF or other data via HTTP,
- read from and write to local or remote SPARQL endpoints,
- iterate through result sets,
- load and include other SPARQLScript routines from the web, and
- generate HTML from SPARQL results and RDF resource descriptions.

SPARQLScript function calls return textual results or structured values that can be post-processed or used as placeholder values. Functions can be retrieved from remote servers, but they will be processed locally. For contextual behavior, widgets can access certain local RDF stores which are made available by the widget container (i.e. the dashboard), usually the dashboard’s configuration and data stores, a local cache, and a store for shared information.

Any script and block has a URI and can be retrieved as RDF, which lets script creators easily import and combine blocks from different processes. Copying an existing block’s identifier and pasting it to a block connector will load and inject the referenced block at the respective position.

Widget scripts are instantiated with a set of default parameters. These include list navigation arguments (LIMIT and OFFSET), the value of a keyword search string, and a “Drop-URI” when a resource was dragged on the widget. Additionally, script authors can define custom variables that will appear in the widget’s settings screen and which can be then used by the script, as shown in Fig. 5.

4.2. Dashboards and widgets

Dashboards are the user front-ends of Paggr applications. The essential elements and interaction points are depicted in Fig. 1. Dashboards are created similar to pages in a Wiki. Each user gets a personal URI space where any path can be used to create a new dashboard. Using the options bar, new widgets can be added from the widget server, and the dashboard can be saved and made public by the owner.

Each widget has a little semantic web icon in the upper left corner which can be left-clicked for widget options, or right-clicked to copy a widgets URI. The icon also works as a drop area for other resource identifiers. Dropping a URI on a widget will trigger a server request with the URI passed as parameter. Widget authors can define a default action and directly reload the now parameterized widget, or, in case of multiple possibilities, they may first present a list of options (e.g. “Filter”, “Add”, “Remove”) and continue after receiving a user selection.

All hyperlinks in widgets can be dragged and dropped. Ideally, they represent a Linked Data entity, as in the example use case, where topic identifiers from a conference can be used to filter a session widget, or when person URIs enable semantic filtering of an “Organizations” widget. When a URI is not dropped on a widget but on the dashboard background instead, a basic inline RDF viewer is launched that will retrieve and display information about the dropped resource.

5. Implementation and deployment

Paggr is written in PHP. It uses the ARC RDF toolkit3 for generic functionality such as parsing, storing, or querying. The actual application and user-facing parts are created with Trice,4 an ARC-based web application framework and publishing system. Any piece of information, from sessions to user accounts and cached HTML snip-
Fig. 4. Sparqlet Builder.

Fig. 5. Custom widget parameters.
pets is stored as RDF and accessed through SPARQL (or SPARQL+ [1], in case of update operations).

5.1. Sparqlets

Sparqlets, the RDF-enabled widgets used by Paggr, and related server-side routines are programmed in SPARQL+ and SPARQLScript [1]. SPARQL+ extends SPARQL's standard syntax with query aggregates and basic update functionality. SPARQLScript combines SPARQL+ with placeholders, variable assignments, control structures, queries across multiple endpoints, and result templating, as illustrated in Code Example 1.

```sparql
ENDPOINT <sparql/public>

# feed still fresh?
$fresh = ASK FROM <graph-updates> WHERE {
  FILTER (?date > "$\{NOW-60min\}\")
}

IF (!$fresh) {
  # refresh feed, update graph log (SPARQL+)
  LOAD <http://planetrdf.com/index.rdf>
  DELETE FROM <graph-updates> {
  }
  INSERT INTO <graph-updates> {
    <http://planetrdf.com/index.rdf> dc:date "$\{NOW\}\"}

  # retrieve latest items
  $items = SELECT * WHERE {
    ?item a rss:item ;
    rss:title ?title ;
    dc:date ?date .
  } ORDER BY DESC(?date) LIMIT 10

  # output template
  $size = $\{\$items.size\}
  IF ($size) {
    "I found $\{\$items.size\} items: <ul>"
    FOR ($item in $items) {
      "<li>$\{\$item.title\}</li>"
    }
    "</ul>"
  } ELSE { "sorry, no items found"}

  Code example 2: A SPARQLScript routine that auto-reloads a graph and renders a list of feed items.

Dashboard interactivity is implemented in JavaScript. Paggr registers custom handlers for common events like “mouse-up”, “mouse-down”, or “click”, which then makes it possible to detect if a widget is going to be re-positioned, or if a link was dropped on a widget.

5.2. Deployment

A Paggr application can be installed on standard LAMP servers without the need for administration privileges; an FTP account is usually enough for deployment. It is, however, possible to use a more advanced, distributed setup to improve performance and scalability. User RDF stores, for example, can be spread and hosted on a server cloud. Likewise, Trice supports multiple session stores and can work with load-balanced front-end servers. So far, individual user stores and a central user account registry have to scale vertically, but the remaining storage components can scale horizontally. As Sparqlets mainly work with remote SPARQL endpoints, the overall user experience relies less on local user store performance than on the response times of external services. A lightweight Paggr system could be used as front-end for large-scale back-end systems.

Sparqlets are run in separate SPARQLScript processor instances for better script control (e.g. each script can only perform a limited number of operations) and also to reduce security risks (the scope of variables is limited to the running script, function calls require explicit arguments).

While Sparqlets can be adjusted and optimized for a certain application context like Paggr’s personal dashboards, this is not a requirement. The widget scripts can be kept application-independent and at least their HTML results could therefore be embedded in third-party web sites. An open-source SPARQLScript processor is part of the ARC toolkit. So far, however, Sparqlets are not directly compatible with existing widget containers such as the OS X Desktop or AJAX Startpages like Netvibes. This sort of integration would require a wrapper. Likewise, enabling Paggr’s level of widget interaction on arbitrary HTML sites would need an additional JavaScript library.

---

5 LAMP stands for servers that use Linux, Apache, MySQL, and PHP.

6 http://netvibes.com/
6. Related work

Paggr was inspired by Berners-Lee’s idea and concept of a Semantic Clipboard [2] that suggested to use RDF’s typed links and self-describing resources for semantics-aware applications. Paggr tries to bring the idea of such a clipboard to web sites and is thus limited to web-based containers whereas Berners-Lee’s idea considers both desktop and web applications.

The success of AJAX widgets and dashboard applications like Netvibes gave Paggr its visual direction. Paggr is different in that each widget is based on typed, URI-identified objects which enable the interaction and linking between individual widgets and widget items.

Paggr widgets use a technical trick to enable native copy and paste in web pages that was made popular by Live Clipboard[7]: Visual overlays are combined with transparent form elements so that copy or paste actions in these visual areas can reuse operating system functionality.

Paggr’s Sparqlet Builder is comparable with solutions such as DERI Pipes [3] (which were inspired by Yahoo! Pipes[8] but also process RDF) or SPARQLMotion[9], a commercial solution for building RDF- and SPARQL-driven mashups. The main difference of Paggr (and Sparqlets in general) is the ability to use this type of semantic development tool and processor in widely deployed web server environments. The low installation requirements (neither administration privileges nor application servers) can potentially bring semantic scripting to a larger developer community than Java-based or desktop-oriented products. For this new target audience, the procedural approach of SPARQLScript reduces the immediate need for a visual editor or having to remember a difficult, fully declarative syntax. Developer feedback from a related project[10] suggested that average SQL and programming skills, together with a few code examples are sufficient to get started with SPARQL and SPARQLScript. The latter also provides a templating module which is similar to front-end templating mechanisms. Sparqlets offer both presentation and data-level functionality for the creation of simple front-end-facing applications as well as back-end services in a single development environment. In general, however, the Sparqlet Builder is less feature-rich and mature than SPARQLMotion or DERI Pipes.

An interesting aspect on the semantic web market is that different solutions are not necessarily competitive in a sense that one product excludes the use of another one. While, for example, mashups built with Popfly[11] are not interoperable with MashMaker[12], the common RDF standards lead to more complementary products. SPARQLMotion can theoretically access DERI Pipes results and produce output that can be post-processed by Sparqlets, etc., even if it is not possible yet to directly reuse different tool procedures natively.

7. Conclusion and future work

Paggr is one of the first applications that combines the wide range of possibilities enabled by Linked Data with simple, task-optimized user interfaces. By breaking use cases down into dashboards, widgets, and widget actions, application providers can offer a rich set of features and still let their users define the level of complexity they are able to handle. With Sparqlets being RDF-based, Paggr not only enables repurposing and reuse of machine-readable data, but also of distributed application components that adapt to the dashboard and information context they are run in.

After deploying the first public Paggr applications, it has become clear that certain interaction methods still require more user feedback and improvements before they will be sufficiently intuitive. The Sparqlet Builder and the inline RDF viewer are also just first releases and will have to be extended. Set-based filtering is currently discussed and explored in the Semantic Web community, it could be interesting to apply the concept to Paggr as well, e.g. by enabling drag and drop for complete lists, not just individual resource identifiers.

References
