ABSTRACT
The Linked Open Data (LOD) cloud has grown significantly in the past years. However, the integration of LOD into applications has not yet become mainstream because managing LOD for applications can be a challenging task and adapting (existing) applications for use with LOD is complicated. We have build a Linked Data Framework (LDF) that makes it easy for application administrators to integrate and maintain LOD sources in their application. The framework enables the integration of LOD in existing applications that make use of web services by providing a way to create a customized API wrapper for the application to interact with LOD, using existing web service description languages such as WSDL to define the needs of the application. The framework allows for easy editing and testing of the used queries and processing of query result data by the means of plug-ins. We have performed evaluation of the framework to demonstrate its usability. We have found that the first prototype of our framework simplifies the development, deployment and maintenance of Linked Data applications. Its modular structure and support for standards are important properties. It can therefore contribute to the increase of Linked Data usage in mobile and web applications.

KEYWORDS
RDF, Linked Open Data, LOD Cloud, Framework, Web, Data mapping, Semantics, WSDL, Web Services, PHP, MySQL

1. INTRODUCTION
Linked Open Data (LOD) is data that is openly available on the Internet in a machine readable form, and linked to other data using Resource Description Framework (RDF) triples1; a framework for describing data items and their relations on the web. An RDF triple is a simple data relation consisting of a subject, a predicate and an object, e.g. banana – is of type – fruit. These concepts are represented by Uniform Resource Identifiers (URIs), which effectively are addresses on the Internet that can be linked or resolved using HTTP2. A bundle of interconnected triples together form an RDF Graph, that in turn could form a data set in the LOD Cloud. More and more of these data sets are being linked to each other, and together they form the LOD Cloud. The LOD Cloud has grown rapidly in the past years. It has grown from only a few data sources in 2006 to more than 300 data sources at the end of 2011, which together contain more than 30 billion triples (Bizer, Jentzsch & Cyganiak, 2011). While some of these data sources are generic in nature (e.g. DBPedia3), others are more specific. The data in the LOD cloud would be of little use if one could not query it. SPARQL4, a recursive acronym for SPARQL Protocol and RDF Query Language, is a query language for databases that consist of data stored in RDF format. Such database systems are called RDF stores. SPARQL is considered as one of the key technologies of the semantic web and is also an official W3C recommendation. Most RDF data stores provide what is called a ‘SPARQL endpoint’; an interface for sending SPARQL queries to be executed on the data in the RDF store, with the results of the query being sent back to the requestor. With the amount of data available in the LOD Cloud growing rapidly, the value of the LOD Cloud increases; the combination of data in generic and more specific data sets holds potential for creating numerous applications that serve various demands and domains.

1.1 Problem Statement
Despite the growing possibilities for the creation of LOD applications, the integration of LOD into applications has not yet become mainstream. As an illustration, a quick survey along development related websites shows that RDF is not yet seen as an important skill by the majority of developers.

1.1.1 Quick Survey along Development Related Sites
The website vWorker.com5 (formerly known as rentacoder.com) has a detailed classification system for freelancers, but does not include categories for either RDF, LOD, Linked (Open) Data or SPARQL. The website rentacoder.com6 also does not list these categories, and search queries on the website guru.com7, which has more than 350.000 registered freelancers, return only 38 freelancers that mention RDF as a primary or secondary skill and only 11 freelancers that do the same for SPARQL (results of June 2012). A very popular developer-help site at the moment is Stack Overflow8. On this site developers can ask questions that other developers can answer, while a voting system ensures the best answers to be listed first. As of July 2012, 39,045 questions have been asked on their site that contain the tag ‘database’ and 17,598 questions contain the

1 http://www.w3.org/TR/rdf-concepts/
2 Hyper Text Transfer Protocol: http://www.w3.org/Protocols/HTTP/
3 http://www.dbpedia.org/
4 http://www.w3.org/TR/rdf-sparql-query/
5 http://www.vworker.com/
6 http://www.rent-acoder.com/
7 http://www.guru.com/
8 http://stackoverflow.com/
tag ‘query’. On the other hand, only 604 questions contain the tag ‘query’, while only 440 questions contain the tag ‘sparql’. According to the StackOverflow site, all four of these tags were created 1 year ago.

1.1.2 Complicating Factors in Using Linked Data
To be able to harvest the potential value of the LOD Cloud in terms of useful applications, we need ways to create such applications in a relatively simple and straightforward manner. However, there are factors that compromise the rapid development, deployment and adjustment of LOD applications. In this section we will address these factors.

1. The LOD Cloud has grown considerably (Bizer, Jentzsch & Cyganiak, 2011) and is now at a size that makes it nontrivial to oversee which elements are useful for a certain application and which are not. As a result, considerable effort must be taken to clearly investigate and define what data is to be used in a newly to develop application, i.e. which RDF graphs will be used, how they are linked to each other, etc. The outcome of this process will be different for each distinct application. As a consequence, the data consuming components of the application will either have to be purpose built, or a solution has to be found to abstract these differences away from the application. Data complexity also plays a role in application maintenance. With the LOD cloud growing rapidly, it is desirable to be able to easily change the way Linked Data is used by a LOD application in order to involve more or other data. Being it either data that was added to the LOD Cloud after the application was first designed, or data from sources that were not considered during the initial development phase. In both cases this data could provide additional value for the application. In order to be more flexible with respect to the adjustment of data used by the application, this aspect needs to be configurable without or with very little modification of the underlying application code. Moreover, to simplify and speed up application development, it would be beneficial if we could change the way the application deals with Linked Data to an extent that makes it possible to reuse the application, or parts of it (modules), using a different data selection.

2. Another aspect is the deployment of LOD web applications. In order to target a large area of web application developers, the technology needed to build LOD web applications should be available on commonly used hosting platforms. The majority of these services are so called ‘shared web hosting services’ or ‘virtual web hosting services’, where multiple sites run on a single shared server. Each on their own ‘partition’, but sharing resources such as RAM9 and CPU10 power. These hosting services offer a standardised package of supported services and technologies, and in many cases do not allow the user to install exotic software. A common configuration is the so called ‘LAMP’ configuration, which stands for the combination of Linux11 (as operating system), Apache httpd12 (as HTTP server), MySQL13 (as relational database system) and PHP14 (as server side scripting language). The configuration of these hosting environments and the restrictions that apply make RDF Store solutions such as ClioPatria15 and Jena16 unsuitable, because they rely on Prolog17 and Java18, respectively. The ARC19 implementation on the other hand is based purely on MySQL and PHP, thus making it suitable for use in LAMP environments. However, as for now (July 2012) development of ARC(2) has ceased due to lack of funds.

1.1.3 Recapitulation and Requirements
To summarize the above, for LOD applications to become mainstream, we need a solution that makes it easier to develop, deploy and maintain LOD based applications. The solution should:

- Help developers to arrive at ‘interesting’ applications more quickly and efficiently;
- Use techniques that are widely available and affordable such as PHP, MySQL, Javascript, etc.
- Make it easier to fine tune the operational behaviour of such applications in order to ease application maintenance in the dynamic data environment of the LOD Cloud;

To accomplish these goals, we need a solution that separates the application logic from dealing with domain specific knowledge and data selection. In other words, the solution needs to offer the possibility to map functionality-oriented semantics to domain-oriented semantics in order to specify what data from the LOD Cloud is required for the various application functionality, without affecting design choices in the application logic.

In this paper, we describe our efforts to build a prototype version of an application framework that facilitates Linked Data based applications. In order to be a solution to the problems as described in the previous section, the main requirements for this framework are as follows:

1. The application framework has to have the ability to map application specific semantics to domain specific semantics, and therefore serve as a middle layer between Linked Data sources and Linked Data based applications.

2. The framework has to extract the data selection process away from the application.

9 Random Access Memory
10 Central Processing Unit
11 http://en.wikipedia.org/wiki/Linux
12 http://httpd.apache.org/
13 http://www.mysql.com/
14 http://www.php.net/
15 http://cliopatria.swi-prolog.org/home/
16 http://incubator.apache.org/jena/
17 http://www.swi-prolog.org/
18 http://www.java.com/
19 http://arc.semsol.org/
3. The settings available to the user of the framework must be devised in such a way that its is easy to change the data selection process.
4. The framework must contribute to the reusability of applications and application components, to allow for faster development and deployment of new Linked Data applications.
5. The framework must be easy to use and install on average shared hosting environments.

2. CONTEXT
In this project we used the efforts of the VU Significant Sights project to create a dynamic LOD based tour application as a use case in order to develop the Linked Data Framework, evaluate it and illustrate our findings. This project has a clear focus on the cultural heritage domain. Before explaining this project further, we will shortly introduce some related and previous projects to sketch the scene of open data usage in cultural heritage applications.

2.1 Open Data in Cultural Heritage Applications
Many cultural heritage organizations have been working on the digitization of their collections in the past years, while the efforts of others are underway. In recent years several applications have been developed that exploit these digitized collections. Examples of such applications include the CHIP Project and the MultimediaN N9C Eculture project. The CHIP Project aims to demonstrate the deployment of novel Semantic Web technologies by means of a semantics-driven browsing application. With this application the visitor can embark on a virtual (and physical) tour through the Rijksmuseum, composed out of semantic recommendations using a personalised user model. The MultimediaN N9C Eculture project aims to provide a platform for cross-collection searching and browsing of cultural heritage collections. The purpose is to demonstrate the use of semantic interoperability, information access and visualization, and context-specific presentation generation. The Europeana project presents itself as “a single access point to millions of books, paintings, films, museum objects and archival records that have been digitised throughout Europe”. To reach this goal Europeana collects data on cultural heritage objects from institutions across Europe. For these institutions, Europeana is an opportunity to reach out to more users and enhance their experience. In order to have a common standard, the Europeana Data Model (EDM) was developed. The EDM is described in detail by Doerr et al. (2010). The main goal of Europeana is not just to collect all this data, but to enrich the data and make it available to the general public. To make the data available as Linked Open Data, the Europeana Linked Open Data Pilot was initiated (Haslhofer & Isaac, 2011). These efforts show that the Cultural Heritage domain is actively involved in exploring the use of Linked Open Data, which is why we chose this domain as context for our project.

2.2 Use case: The AmsterTour application
The VU Significant Sights project combines insights gained in earlier projects such as CHIP and Eculture. On the one hand it aims to develop a dynamic tour through Amsterdam that adapts to the user’s context, e.g. location, time and preferences, as well as to the user’s interests. On the other hand, it uses four integrated Linked Data collections: (1) Amsterdam Museum, (2) ‘Stadsarchief’ Amsterdam (Amsterdam City Archives), (3) ‘Rijksmonumenten’ (monuments appointed by the Dutch State department of Cultural Heritage) and (4) ‘Beeld & Geluid, Open Beelden’ (Dutch audiovisual archive ‘Sound & Vision’, Open Images) project. By examining these four data collections of known quality, interesting links can be found between data from the various sources. These links can be exploited in applications such as the mobile tour guide the project aims to realize. The project focuses on different aspects that play a role in the realization of this mobile tour application:

1. How the use of Linked Open Data influences the user interface design of such applications;
2. How the development, deployment and maintenance of Linked Data based applications can be simplified.
3. How the use of Linked Open Data influences the consumer application, but a proof of concept for using Linked Open Data in such applications.

This paper deals with the second aspect. As a ‘runner up’ to both research issues, a sample application called ‘AmsterTour’ was developed to serve as a starting point and to gain ‘hands on’ experience in building such applications. AmsterTour is a mobile city tour guide that generates personalized tours through the city of Amsterdam and provides its users with interesting cultural and historical information. The application is described in detail by Schootemeijer et al. (2012). The application creates a tour along monumental buildings in the city center, provides the user with a route to walk between these buildings and provides pictures and information on the selected buildings. The application is not a ‘market ready’ consumer application, but a proof of concept for using Linked Open Data in such applications.

In the following section we will discuss related work. In section four we will describe the conceptual model, the GUI and the implementation of the Linked Data Framework. Section five discusses the evaluation of the framework, followed by the discussion and conclusion. We will end this paper with an indication of future work.

3. RELATED WORK
In this section we will take a look at related work. This includes work that has similarities with the Linked Data

---

26 http://beeldbank.amsterdam.nl/
27 http://monumentenregister.cultureelerfgoed.nl/
28 http://www.openbeelden.nl/
29 http://www.amstertelden.nl/
30 Graphical User Interface
Framework we developed, as well as work that can complement or facilitate the Linked Data Framework (LDF).

### 3.1 The Callimachus project

Another interesting project that aims at the use of Linked Data in applications is the Callimachus project\(^3^1\). The Callimachus project is a framework for data-driven applications based on Linked Data. It is sometimes also referred to as a ‘Linked Data Management system’ as opposed to traditional ‘Content Management Systems’ that deal mostly with unstructured data. Callimachus manages mostly structured data, with textual pages to explain, navigate and visualize that data. The system can also be compared with data warehouses. Human readable pages can be created that use the structured data. This is accomplished with the use of ‘view templates’, that consist of classes associated with the data to be displayed. Using these templates, display to the end users can be controlled and rich and dynamic web pages can be created over the data. The Callimachus project has in common with the LDF that it is a tool to ‘manage Linked Data’ and that it aims to make the use of Linked Data in web applications easier. It differs from the LDF in the sense that the LDF is purely a ‘mediator’ or ‘middle layer’ between the data source and the application, whereas the Callimachus project actually stores the data itself. Furthermore, the Callimachus project uses a system of template based pages to visualize the data and therefore seems to be more oriented at publishing, whereas the LDF supplies raw data according to a specified schema to applications, being it web applications or mobile and native applications that use HTTP requests to gather data.

### 3.2 The LarKC project

With the semantic web growing bigger and bigger, new cutting edge technology is needed to query all this data and reason with it in an efficient way. For large scale reasoning on this semantic web data, the resources of a single machine are no longer sufficient. To this end, the Large Knowledge Collider (LarKC)\(^3^2\) was developed. LarKC is a platform for massive distributed incomplete reasoning, that aims to remove scalability barriers in semantic web reasoning. Fensel et al. (2008) explain how this can be achieved: “This is achieved by (i) enriching the current logic-based Semantic Web reasoning methods, (ii) employing cognitively inspired approaches and techniques, and (iii) building a distributed reasoning platform and realizing it both on a high-performance computing cluster and via “computing at home”. As part of the LarKC platform, WebPIE\(^3^3\) was developed; The Web-scale Inference Engine. WebPIE uses distributed techniques to perform reasoning on RDFS and OWL semantics. As such, it is based on the MapReduce programming model (Dean & Ghemawat, 2004). WebPIE is explained in detail, along with several example algorithms, by Urbani et al. (2011). The LarKC project officially ended in September 2011 and a third, final release of the platform is now available. The LarKC platform is different from the LDF in the sense that it itself is a reasoning platform, capable of doing massive distributed incomplete reasoning, whereas the LDF relies on external RDF stores to perform SPARQL queries. The similarities are that they both include methods to query data, alter this data using plug-ins written for the platform, and allow to piece together a solution that fits the demands of the target application. The LarKC platform is far more advanced than the LDF, but also requires much more expertise to use. Furthermore, the LarKC platform can not be used in an average shared hosting environment, whereas the LDF can. The LarKC platform is designed for large, cutting edge semantic web projects dealing with billions of triples, whereas the LDF is designed for easy use in small to medium sized projects running on average hosting environments. The goal of the LarKC platform is to provide new cutting edge technology to challenge massive scale reasoning problems, whereas the goal of the LDF is to bring straightforward use of Linked Data to the masses.

### 3.3 SWSE

SWSE\(^3^4\), which stands for Semantic Web Search Engine, is an application for searching and browsing through the LOD Cloud. Hogan et al. (2011) explain that “Following traditional search engine architecture, SWSE consists of crawling, data enhancing, indexing and a user interface for search, browsing and retrieval of information; unlike traditional search engines, SWSE operates over RDF Web data [...] which implies unique challenges for the system design, architecture, algorithms, implementation and user interface”. The interesting aspect of LOD search engines is that they could be used to really give answers to questions users ask the system. Something that current search engines for the human readable web cannot. According to Hogan et al. “Google does not typically produce direct answers to queries, but instead typically recommends a selection of related documents from the Web. We note that in more recent years, Google has begun to provide direct answers to prose queries matching certain common templates – for example, ‘population of china’ or ‘12 euro in dollars’ – but again, such functionality is limited to a small subset of popular user queries.” The research of Hogan et al. shows the possibilities in the direction of direct question answering. They conclude: “Based on the experiences collected, [we] have identified open research questions which [we] believe should be solved in order to – directly or indirectly – get closer to the vision of search over the Web of Data [...].” Even though a search engine is clearly very different from an application framework such as the LDF, the SWSE has in common with the LDF that it aims to make Linked Data more accessible to a general public. Furthermore, it would be interesting to have a semantic search engine embedded into the LDF, in order to select data to use in the application relying on the LDF.

### 3.4 RDF Stores

The data available in the LOD Cloud needs to be stored somewhere. The RDF triples that make up this data can be stored in so called RDF Stores. Storage of triples takes place in purpose built database systems. Furthermore, RDF

\(^3^1\) http://callimachusproject.org/

\(^3^2\) http://www.larkc.eu/

\(^3^3\) http://www.cs.vu.nl/webpie/

\(^3^4\) http://swse.deri.org/
stores must at least provide the interface to store and/or retrieve RDF data. Examples of RDF store implementations that can serve as a back-end for LOD web applications include ClioPatria and Jena. Both platforms are able to retrieve, read, process, write and store RDF data and both platforms provide a SPARQL endpoint. ClioPatria is an SWI-Prolog implementation and Jena is Java framework. Another implementation is called ARC. Now in its second release, ARC2 offers a PHP-based solution that offers RDF parsers and serializers, as well as RDF storage based on MySQL. ARC also offers a SPARQL endpoint. The ARC system is described in detail by Nowack (2008). Within the context of the Europeana project, Haslhofer et al. (2011) have done a qualitative and quantitative study of these and other existing RDF stores. Not included in this study is the RAP toolkit\(^3\). RAP stands for RDF API\(^4\) for PHP, and this toolkit offers features for parsing, manipulating, storing, querying, serving, and serializing RDF graphs. The RAP system is described by Oldakowski, Bizer & Westphal (2005). RDF stores are related to the LDF in the sense that the LDF relies on external RDF stores to perform SPARQL queries. The LDF can use RDF stores hosted at several different locations. It would be interesting to investigate the integration of an RDF store into the LDF, for example to facilitate local caching of RDF data.

4. THE LINKED DATA FRAMEWORK

In order to provide a solution as described in section 1.1.3, we have developed the ‘Linked Data Framework’: An application framework that facilitates easy data management for Linked Data applications.

4.1 General Description

The Linked Data Framework is a (web) application in its own right, that manifests itself as a ‘control panel’ - type application that can be installed on any PHP/MySQL hosting environment, in many cases being the hosting environment of the ‘target application’. By ‘target application’ we mean the application that will be using the Linked Data Framework to gather data. The target application can be of several types, e.g. a web application, a mobile app, or a native application that relies on HTTP requests to gather data.

4.1.1 Extracting the Data Selection Process

Using the Linked Data Framework, the data- and domain specific aspects of the application can be extracted out of the main application and put into the framework as a collection of settings instead of customized code. The data selection process no longer takes place in the target application, but in the framework. The Linked Data Framework has a modular setup, to allow for easy editing of the data selection process. The framework uses SPARQL to select data from remote sources, and has a plug-in system to allow for manipulation of the selected data. The devised queries can be tested immediately from within the framework, and the sequence of queries and plug-ins can easily be edited using a visual ‘data chain’ editor. With ‘data chain’ we mean something similar to a ‘plug-in chain’ (i.e. a collection of plug-ins that are chained together), the difference being that the ‘data chain’ can also contain queries or (in the future) other data gathering or manipulation components. The target application interacts with the framework using an API interface. The target application can do a ‘data request’, i.e. a request to the framework to provide it with data, optionally providing input parameters. The Linked Data Framework will handle the request by selecting data using the chain of queries and plug-ins as setup for the specific request.

4.1.2 Flexibility in application maintenance and reuse

With the data selection process handled by the Linked Data Framework, the way in which data is selected can be edited without the need to make changes to the application. This can include simple modifications e.g. to select additional data, as well as more drastic changes; By altering settings in the framework such as the used selection query and/or the used plug-ins, the main application (or modules thereof) can be reused with data from other Linked Data sources. These ‘application modules’ can be shared among users of the framework, while the same holds for plug-ins. A ‘marketplace’ can be instantiated that users of the framework can consult to find an application module or plugin that provides certain functionality. To make the distinction between these two terms more clear, an application module is a piece of code that is part of the target application and does something with the data received from the framework (after the complete request has been handled), while a plug-in does something with the data as part of the request handling within the framework.

4.2 Conceptual model of the LDF

The Linked Data Framework can be used to specify a collection of settings that together determine how data is selected from the LOD Cloud to fulfill the target applications data needs. In order to be flexible and allow for reuse, we have chosen a modular setup for the Linked data Framework. The framework consists of ‘LOD Sources’ that can provide Linked Data (i.e., that provide an accessible endpoint), ‘Queries’ to select data from the specified LOD Sources, ‘Plug-ins’ to alter selected data, ‘Application Modules’ that group together different requests that can be made by the target application and ‘Data Requests’ that contain the settings to define how a specific request should be handled. In short, these components in the Linked Data Framework together define how to transform the request input of data requests done by the target application into the desired request output. In the following subsections we will give a more detailed description of each of the components in the Linked Data Framework (LDF).

4.2.1 LOD Sources

Within the context of the LDF, an ‘LOD Source’ is a collection of settings that together specify a particular data source; its name, description, location, and how to access it. For example, the type of source is specified (e.g. SPARQL using HTTP POST), the endpoint address and any required parameters.

---

\(^3\) http://www4.wiwiss.fu-berlin.de/bizer/rdfapi/
\(^4\) Application Programming Interface
4.2.2 Data Queries
A Data Query is a specification of a query statement that can be executed against a LOD Source. Apart from the actual query text, the LOD Source can be selected towards which the query needs to be executed. By separating the source and the query, a query can easily be tested against different sources, or different versions of the same source. The Data Query concept within the LDF also keeps track of the ingoing and outgoing data involved in executing the query. The ‘input data’ for the query is defined by replacement tags in the query as shown in Figure 1, while the output data consist of the data fields returned by the query.

![Figure 1. The query window showing the use of replacement tags for ‘input data’. The tag is replaced by the actual value at runtime. The value within the tag is used for testing purposes.](image)

4.2.3 Data Plug-ins
A Data Plug-in is a piece of code that is intended to alter data supplied to the plug-in or provide new data derived from the supplied data. A plugin can either be an external file that contains the effective code, or a piece of code that can be edited directly using the framework and is stored in the frameworks database. The first option aims to be practical for more complex plug-ins and to allow for the sharing of plug-ins among users of the framework, while the latter option is handier for quickly accomplishing small tasks. Depending on the chosen type, either the file location of the plug-in, or the actual code needs to be specified.

4.2.4 Application Modules
Within the context of the LDF, an Application Module is a collection of settings that together represent the ‘data needs’ of the target application, or a particular module of the target application. For example, if the target application can make three different ‘data requests’ (comparable to API calls), the Application Module in the LDF will list these three requests. Furthermore, it contains the settings needed to define how these three types of requests should be handled. Each application module consists of its General Settings and a collection of Data Requests.

- **General Settings:** The general settings for an Application Module consist of the file location of a config file which describes the Data Requests for the target application, or a particular module thereof. After a config file is selected, the LDF extracts a list of the Data Requests and the input and output variables (data documents) for each request.

  - **Data Requests:** The settings for a Data Request define the ‘data chain’ that will transform the data provided according to the request input specification into data required by the request output specification. They determine which Data Queries and Data Plug-ins are involved, in what order they are executed and how data is mapped between these components. Hence, these settings specify the flow of data from request input to request output.

4.2.5 Interplay of Components
These components together form the configurable set of tools that allow an application administrator to define and adjust the data selection strategy for an application without the need to modify the target application. The target application uses an API interface provided by the framework to request data needed for its operation. Figure 2 provides an overview of the various components and how they work together to fulfill a data request made by the target application.

![Figure 2. Conceptual model of the LDF. An Application Module groups together multiple Data Requests. The ‘Data Chain’ of a Data Request can consist of multiple queries and plug-ins.](image)

The target application can perform a data request by making an API call to the framework. This could either be a HTTP request or a direct function call in PHP. When the LDF receives a data request, it loads the settings for the particular request and executes the Data Queries and Plug-ins that together form the data chain for that request. The queries and plug-ins are executed in the order that was specified using the GUI of the LDF. The end result of the data chain is used as response to the original data request made by the target application.
4.3 Data management between objects in the data chain
An important aspect of the data chain in the LDF is how the flow of data between queries and plug-ins is handled. This aspect played a big role in realizing the modular setup of the LDF and the possibility to share and reuse application modules and plug-ins. Another aspect is that connecting different queries and plug-ins in the data chain should be relatively easy for the user and done be configuration, not by coding. Hence, in order for the LDF to meet its requirements, data handling between the components in the data chain (queries or plug-ins) needed to fulfill the following criteria:

- No restrictions may apply to the specification of input and output formats for components, as well as the data chain as a whole. The LDF or data chain therein may not require certain specifications in order to operate, other than the fact that each input and output document has a single root element.
- In order for components to be general purpose and shared among users of the framework, a component must be able to assume that data will be available according to its own input specification, no matter what format was used by a previous component. Similarly, it must be able to assume that any derived data can be written back using its own output specification, regardless the format expected by the next component.
- No undesirable behaviour such as naming conflicts, overwriting or deletion of data may occur.
- The solution should be scalable, e.g. when dealing with large amounts of data, copying and converting data back and forward into different formats is undesirable.
- All data available to, or provided by, earlier components in the data chain should be available to the next component. These criteria are important to allow for the easy sharing and reuse of queries, plug-ins and application modules. If these conditions can not be met, components have to be altered each time they will be re-used, to meet local demands. For the LDF to fulfill its requirements, easy sharing of components is a critical factor.

4.3.1 Data Mapping
To facilitate the flow of data between components in the data chain and meet the requirements, we created a mapping solution that allows the user of the LDF to map data members of one structure to data members in another. Consider the following example:

```
root.street[A]  >  root.address[A].street
root.house[nr][A]  >  root.address[A].number
root.city[A]  >  root.address[A].city
```

The above listing shows a mapping of address information from one structure to another. The structure on the left side of the “greater than” sign (which in this case stands for “Mapped to”) contains a single root element that in turn contains three separate listings (one might think of them as arrays) of streets, house numbers and cities. However, the structure on the right side contains only one listing, being a listing of complex elements (one might think of them as objects) that each contain a street, number, and city. The ‘A’ between the square brackets in both structures indicates a matching index. This means that on reading actual data, an index at position ‘A’ in the one structure, must be filled in on the location of the ‘A’ in the other structure. A mapping can contain multiple index indicators, as well as static numeric indexes to reach a certain part of a data structure:


In the latter case, the ‘3’ will not be matched and the mapping will always refer to the city member of the A’s address ‘object’ in the 3rd dataset. The mapping solution in the LDF allows such ‘mapping entries’ to be entered, after which a component can read the data using the mapped format, without the need to copy the data.

4.3.2 Central Data Structure
During execution of the data chain, the LDF manages one central data structure. This data structure starts out containing the initial request data, and expands after execution of each component in the data chain that produces new result data. To prevent conflicts, result data of a component in the data chain is always added to a new ‘node’ in the central data object. To allow components to read and write data according to their own specification, for each component (i.e. query or plug-in):

- A mapping object is created, that contains the mappings between the schema expected by the component and data in the central data object. Mappings can exist to all data in the central data object, including the original request data and the result data from previous components that was added to new nodes. The mapping object is passed to the component as ‘data-in’ object. The mapping object is discarded after execution of the component finishes.
- A data structure is created that matches the components output schema specification. This data structure is passed to the component as ‘data-out’ object. The structure is added to the central data structure on a new ‘node’ after execution of the component finishes.

After the execution of a data chain that consists of one query followed by one plug-in, the central data object will have the following structure:

```
CentralDataObject
→ Request Input
  → ID
  → Data
  → {Request input schema}
→ Query
  → ID
  → Data
  → {Query output schema}
→ Plug-in
  → ID
  → Data
  → {Plug-in output schema}
→ Request Output
  → XML Data (or other format)
```
Besides the central data structure, three mapping objects have been created in this example to function as ‘data-in’ object for the query, plug-in and request output component. These objects have been discarded after execution of the components finished. The data chain always contains a ‘Request Input’ and a ‘Request Output’ component. The first contains the request input data provided with the API call. It serves to make a data mapping between the request input data and the first component in the data chain, in most cases a query component. The ‘Request Output’ component in the data chain serves to make a mapping between the last component in the data chain and the required output schema specification for the API call. When the Request Output component is executed, it serializes the data specified in the mapping to XML according to the provided schema, or another specified format such as JSON.

4.4 The GUI of the LDF

The user can access the Linked Data Framework by entering a special sub-path to the URI on which the main application is hosted, or (if the target application does not utilize a URI) using any accessible web address that can be arranged for the purpose of accessing the framework. Typically, the address will end with '/ldf/', short for Linked Data Framework. The AmsterTour application for example, introduced in section 2.2, has the web address http://www.amstertour.com associated to it. If the AmsterTour application were to use the Linked Data Framework, the framework and the settings therein would be accessible through http://www.amstertour.com/ldf/.

The GUI of the LDF is set up to resemble a desktop application, more than a web site. The layout of the GUI is in IDE-Style (Integrated Development Environment), a sub-form of a Multiple Document Interface (MDI). It has panels that can be resized or collapsed and a central ‘work field’ in which setting windows will open, including code editors for queries and plug-ins. The GUI contains a toolbar at the top for actions such as adding components to the framework, a panel on the left called ‘Component browser’ to list components, and a panel called ‘Immediate’ at the bottom of the GUI to display test results. Figure 3 shows an overview of the GUI.

The component browser contains a ‘harmonica-view’ to organize the various components in the LDF. It has four separate panes to list the LOD Sources, Data Queries, Data Plug-ins and Application Modules that are loaded into the framework. The pane for LOD Sources, Data Queries and Data Plug-ins contain a list of the respective components. The pane for Application Modules contains a folder for each application module loaded into the framework. Each folder contains a link to the general settings window for that module, and a folder that contains the Data Requests for the particular module. When a component is double-clicked, a settings window for the component opens in the central work field. The setup of the GUI is such that it is easy to compare (copy and paste) settings and compare results. All settings windows open as a child window of the center panel so that they can be viewed next to each other. However, if the user wants more space while editing he/she

![Figure 3. An overview screenshot of the LDF GUI.](image-url)
can maximize a window to the size of the complete center panel, by pressing the maximize button on the window header. In addition, the Component browser (left) and Immediate pane (bottom) can be collapsed to create even more room for editing.

4.4.2. The Settings Windows of the Components
The settings window for LOD Sources allows the user to modify the settings for a LOD Source, such as the source type, the endpoint URI and parameters required for a request. The settings window for LOD Sources is shown in Figure 4.

![Image of LOD Source settings window]

Figure 4. The LOD Source settings window.

The Data Query settings window allows the user to edit the query and select the LOD Source that the query should be executed against. It also provides the possibility to test the query and see the result it generates. When the ‘Test query’ button is pressed, a new tab will be added to the Immediate panel at the bottom of the GUI. This tab will contain a grid view containing the results of the query. A new tab is added each time the ‘Test query’ button is pressed, to allow for easy comparison while editing the query. Tabs can be closed when no longer needed. The settings window also shows an overview of the input and output variables (or data-documents) for the query. This information is extracted from the query. The Data Query settings window is shown in Figure 1.

The settings window for Data Plug-ins allows the user to select the location of a plugin file containing the PHP code to be executed, or to enter and edit PHP code directly in the GUI. The settings window also shows an overview of the input and output variables (data-documents) for the query. The general settings screen for Application Modules lets the user select a WSDL file to specify the number of Data Requests and the schemas of their request input data and their request output data. The settings window for a Data Request contains the ‘visual data chain editor’ which allows the user to set up the data chain needed to fulfill the request. The user can add query components and/or plug-in components to an initial set of two components, being the Request Input and the Request Output component. By connecting the components and specifying a ‘data-mapping’ between each two connected components, the user can determine the flow of data from request input to request output. Figure 5 shows the settings window for a Data Request.

![Image of Data Request settings window]

Figure 5. The settings window for a Data Request showing the Visual Data Chain with two queries and one plug-in.

4.5 Implementation of the LDF
In order to meet the requirements set out for the framework, we limited ourselves to using only technology available at the majority of hosting providers. This led us to implement the LDF using:

- PHP, for the server side parts of the framework such as retrieving and storing settings and handling the actual data requests;
- MySQL, for storage of settings;
- JavaScript, for client side scripting;
- Ext JS\(^38\), a Javascript user interface library from Sencha;
- jQuery\(^39\), a general purpose JavaScript framework.

4.5.1 Environment
To simulate an average hosting environment, we set up our own web server for developing and testing the LDF. For this server, which we set up as a virtual machine running on VirtualBox, we chose CentOS\(^40\) (5.5) as operating system. CentOS is an open linux distribution based on Red Hat Enterprise Linux\(^41\), a commercial linux distribution used for many LAMP environments. The CentOS community is committed to keeping the distribution compatible with Red Hat Enterprise Linux. CentOS itself is also used by many LAMP based hosting providers and (virtual) private server services. Hence, it is very suitable to resemble a typical LAMP hosting environment.

4.5.2 Implementation of the GUI
For the implementation of the GUI we used the Ext JS library version 4.1 from Sencha. We chose to use this framework after considering several options such as jQuery

---

\(^{38}\) http://www.sencha.com/products/extjs
\(^{39}\) http://www.jquery.com
\(^{40}\) http://www.centos.org/
\(^{41}\) http://www.redhat.com/products/enterprise-linux/
Ut42, jQuery Tools43, DHTML Goodies44 and the Closure library from Google45. Interface Design was not the main topic for this project, and thus the amount of time to spend on it was limited. However, we realized that a clean consistent interface would be important for the LDF in order to make it easy to use and thus be able to fulfill its goals. The Ext JS library offered a complete and integrated approach for setting up a clean looking and consistent GUI. With Ext JS, the complete interface can be built in terms of configurable objects. Ext JS offers multiple approaches for building web applications. We used the Model View Controller (MVC) approach, because this allows for easy addition and/or modification of screens and functionality in the future. Something that comes in handy when building an experimental application. Using the MVC approach in this case means that the layout for screens and screen elements, the underlying code to perform actions such as performing a test request and the data models used to load settings etc. are completely separated entities which are stored in separate files.

4.5.3 Standards: XML Schema and WSDL
The structure of the input message and output message of the API call are defined using XML Schema46. We realized that using standards would make the difference between creating a framework that just works with parts specially created for it, and creating a more general solution that could also work with existing applications. Using XML Schema to describe the request input and output enabled us to extract this information from a WSDL47 file. WSDL stands for Web Service Description Language and is a format that describes how to invoke a web service. It describes the structure of the input and output documents using embedded XML Schema, as well as the method that should be used to invoke the service. Normally, this specification is used to tell an application how it should invoke a web service. By using this specification ‘the other way around’, it can tell the LDF how the target application is going to make its calls to the API, and the response it is expecting. This principle now allows us to use the LDF to specify a ‘LOD based replacement webservice’ for applications that use a web service to provide them with data. Using standards as XML Schema and WSDL provides the following benefits:

- Easy sharing of application modules and plug-ins designed to be used with the LDF, because of well understood specifications;
- Empower existing web service based application with Linked Data from the LOD Cloud;
- The LDF can be used by organizations to keep ‘legacy applications’ that expect a certain API interface running, while migrating to Linked Data. The LDF can serve as middleware between the legacy application and the new Linked Data environment.

Apart from the main request input and output documents, the input and output for queries and plug-ins within the data chain is specified in XML Schema as well. Within the limited scope of this project, support for XML Schema and WSDL was implemented. Because of the modular structure of the LDF, other specifications can be added in the future. Internally the LDF does not work with WSDL or XML Schema. These specifications are only used to transform the data to an internal object structure. Hence, to add support for an additional format, only a parser and (optionally) a serializer for that format need to be added. Besides XML as input/output format for Data Requests made by the target application, we also implemented a parser and serializer for JSON. With the possibilities that the use of standards has brought, the LDF has in effect become an API Wrapper around Linked Data.

4.5.4 Implementation of the Data Mapping Solution
A challenge in the development of the LDF was the implementation of the data mapping solution, as described in section 4.3.1, ‘Data mapping’. After determining the requirements of the mapping solution, three options were considered:

1. Use XML Schema specifications and serialized XML Data as input and output for the data request as a whole as well as for the individual components of the data chain. Use XSLT48 to translate the output document form one component to the input specification of the next component.
2. Use an internal PHP data structure to provide data to the components of the data chain and copy data to the new format expected by each component. Use a parser and serializer to do the transition from and back to the serialized XML data provided and expected by the data request as a whole.
3. Similar to the second option, but use a ‘mapping object’ to be able to read out data according to a schema different than the schema of the actual data.

The first option was attractive in the sense that it was -in principle- very easy to implement. Conversion of XML documents into other XML documents using XSLT is a task that can be accomplished using standard libraries. However, we identified several problems with this approach;

1. It is costly in terms of efficiency and scalability because a constant process of parsing and serializing will be going on between components in the data chain, in order for the components to use the data.
2. Result Data of previous components in the data chain cannot be passed on, because the input/output schema of the next component is not and cannot be aware of that data.
3. Adding additional standards in the future is difficult, because the internal process relies on XML/ XSLT.

48 XML Styling Language Transformations:
  http://www.w3.org/TR/xslt
The solution therefore becomes inefficient and does not meet the requirements.

The second solution does not suffer from these problems. The data can remain in an object structure or associative array, and a ‘central data structure’ can be used as explained in section 4.3.2, ‘Central Data Structure’. However, in order to still support WSDL and XML Schema to specify the various input and output documents, as well as to deliver the resulting request data back to the target application, several parsers and serializers need to be implemented to transform the XML (Schema) into the used internal object representations and, in case of the result data, back to XML. The added benefit is that in the future other (schema) languages can more easily be supported by adding additional parsers and serializers. The downside of the second option, is the fact that data would have to be converted and copied in memory to the ‘input’ object for each component, according to its specified input schema. A process that is undesirable when dealing with large datasets.

The third solution solves this issue by using a ‘mapping object’. Therefore, we started developing the third option, with the second option in mind as fallback option. To be able to represent the entire input schema specification, multiple ‘mapping objects’ are chained together to form the ‘skeleton’ of the data schema. Apart from this ‘skeleton’ structure, there is a single mapping object containing the actual mapping info. The mapping info is analysed and broken down into pieces when the mapping is set, to optimize performance when data is being read at execution time. When a certain data member in the mapping structure is dereferenced, the central mapping object is consulted, providing it an identifier to look up the correct mapping and an array of all indexes that were used in dereferencing the particular element. These indexes will be used to get the desired value in the central data structure. Which index needs to be used at which position in dereferencing the central data structure is determined by matching the [A], [B], etc. index indicators as found in the mapping string. This matching process (also) happens at the moment the mapping is set. Therefore, dereferencing a data member in the mapping structure boils down to the integration of two tiny prepared arrays; An array with sub-members in the central data structure, and an array containing the indexes that need to be used. The order in which the indexes are merged with the sub-members is determined when the mapping is set, and stored along with the sub-members. Consider the following simplified example:

**Mapping that was set:**

```
root.street[A] > root.address[A].street
```

**Dereferenced data member in the mapping structure:**

```
root.street[16]
```

‘Dereference array’ that is passed from the mapping structure to the central mapping object:

```
[0] → [mapping identifier]  
[1] → 16
```

The central mapping object contains a ‘construction array’ for the provided mapping identifier, that was created when the mapping was set, and is a ‘broken down’ version of the original mapping:

```
[0] → root  
[1] → address  
[a2] → 1  
[3] → street
```

The central mapping object now ‘constructs’ a path to the desired data member in the central data structure. This is done by doing a loop over the ‘construction array’ and assigning a reference of the next ‘sub-member’ to a local variable; each time the same local variable. If the key of the construct array starts with ‘a’, the next dereference step is a dynamic array dereference. In this case, the local variable is set to a member of the array it currently Beholds. Which member, is determined by the index that is to be used. That index can be found in the ‘dereference array’ provided by the mapping structure at the position that is given by the member of the construct array. For example, the value of the current element of the construct array is 1, and element 1 in the ‘dereference array’ returns 16. After the loop, the local variable contains a reference to exactly the desired data element in the central data structure, and that reference is returned. This mapping method is efficient in that it basically only dereferences data members in the central data structure. If this structure would be addressed directly, the same principle occurs. When dereferencing the data element `root.address[16].street` directly, internally the object (or associative array) ‘root’ is consulted and asked to return the child member ‘address’. Thereafter the returned array ‘address’ is asked to return the element at index 16, and finally the object at that position is asked to return the child element called ‘street’. Hence, almost the same thing happens in the central mapping object, the extra overhead being a function call to the central mapping object and the loop itself. The third option needs the same parsers and serializers as the second option, to be able to deal with XML data and XML Schema for the input and output specifications.

### 4.5.5 Implementation of the Data Chain Visualization

The library for the data chain visualization was created from scratch, apart from using the jQuery library for tasks such as adding elements to the DOM, changing CSS specifications and creating draggable objects.

- The ‘components’ of the data chain (queries, plug-ins, etc.) are `<div>` elements styled using the CSS3 ‘border radius’ property to create their distinctive shape, and made draggable using jQuery.
- The lines between the components are created using the CSS3 ‘transform’ property. That is, using the various browser specific alternatives since none of the major browsers supports the ‘transform’ property as of yet. The lines are actually horizontal lines positioned with their middle exactly at the middle between the two components, scaled to the distance between these

---

49 Document Object Model: http://www.w3.org/DOM/
50 Cascading Style Sheets: http://www.w3.org/Style/CSS/
two components and then rotated to the angle of the imaginary line between the centers of the two components.

The components in the data chain are stored as ‘objects’ in the visualization script, making it easier to save and restore the entire data chain.

5. EVALUATION

We performed several steps of evaluation to determine if the prototype implementation of the LDF meets its requirements. In the following section, we will describe how we adapted the AmsterTour application to work with the framework. In the two sections thereafter we will discuss how we used the framework on a common shared hosting platform, and how we used a different data set to power the AmsterTour application. In section 5.4 we will describe how we reused an existing application with the data selected for the AmsterTour application, without changing a single line of code in the reused application. These evaluation steps show that we were able to develop an application framework that serves as a middle layer between Linked Data sources and the application, and that this middle layer solution makes it easier to develop, deploy and maintain Linked Data based applications.

5.1 Adapting ‘AmsterTour’ to work with the LDF

The first evaluation step was to adapt the ‘AmsterTour’ application in such a way that it would use the LDF to retrieve the data it needs to operate. The original version of the application dealt with Linked Open Data directly. To do so, it had the necessary script embedded in the application, including the composition of SPARQL queries to select the data at a Linked Data endpoint. In this first step, the main goal was to adapt the application to use the LDF without changes in operational behaviour. This point was crucial in order to show that use of the LDF would not force us to make compromises or other changes with respect to the main functionality of the application. The adaption of the application involved the following steps:

the framework to serve the application:

- We described the requests for data made by the AmsterTour app (operations) and the input and output documents (XML Schema) for these operations. This information was combined into a WSDL specification. The WSDL file was uploaded to the LDF using its GUI, after which the LDF created the necessary ‘Application module’ in order to edit settings to handle the requests.
- We added a ‘Query’ component to the framework containing the basic SPARQL query as used in the original application, supplemented with some replacement tags in order to insert request specific input data. In this case, these fields concern location information needed to select objects nearby.
- We edited the ‘Data Chain’ to specify how a request should be handled. We added the newly created query component to the data chain and connected it to the request input and output components. Finally, we added the ‘data mappings’ for both links. Figure 6 shows the visual data chain with the data mapper.

Adapting the AmsterTour application:

- We bypassed several procedures responsible for creating the SPARQL queries that were to be sent to the SPARQL endpoint;
- Adaption of the procedure that was responsible for sending requests to the SPARQL endpoint. The adapted version sends request to the LDF instead. Along with the request it sends the parameters that were used in the original version to compose a SPARQL query, whereas the original version added the composed query.
- Adaption of the procedure that processed the returned data. We could eliminate some parts of the script that were needed to deal with the general purpose structure of the SPARQL XML that was originally returned to the application, e.g. the SPARQL result set first defines a set of headers before providing the data using abstract terms such as ‘result, ‘binding’, ‘literal’, etc. Using the LDF, the application can simply assume to receive named columns according to its request.

The whole operation took about eight working hours. However, this included fixing some issues with the framework discovered during the process. In a normal setting, we estimate that six working hours would have been sufficient in this case; one and a half hour to study the input/output of the application and the needed changes, one hour to write the WSDL description, one hour to prepare the framework, two hours to adapt the original application, and half an hour for testing and optimizing settings. We are aware of the fact that this is a rather simple example application, which is why we chose it as a first test case. Therefore, more advanced applications would require more time to adapt. On the other hand, if applications or application modules are build with using the framework in mind, no adaption is necessary. In fact, we expect development time to decrease because of the easy data exchange between application and framework. For example, a developer can just assume to receive a JSON object formatted as suits him or her best.

This evaluation shows that we were able to build a working prototype of the proposed framework within the limited time scope of this project, and that we can demonstrate the use of this framework with a sample application. By doing so, we showed that the LDF meets the first two requirements as laid out at the end of the introduction section.

5.2 Using the LDF on an external shared hosting environment

We developed and tested the LDF on our CentOS server running as a virtual machine, set up specifically for this project. This approach allowed us to experiment with server settings etc. without access restrictions or worrying about bringing down the server. In the end little server specific settings were needed, and we constantly worked with the ‘common shared hosting environment’ in mind. However,
to really test the easiness of deployment, we set up a working version of the AmsterTour application and the LDF in order to feed it with data, on a commercial shared hosting environment.

Even before starting this phase of the evaluation, we knew that the demands for running the LDF were low:

- It requires no specific server settings other than a single ‘rewrite rule’ to redirect all API calls made to the framework to a standard handling script. A rewrite rule, in this case, is a configuration setting that tells the server to internally redirect to a different location for all calls on a certain map on the server. This rewrite rule can be specified in a ‘.htaccess’ configuration file, a standard allowed by almost all LAMP based hosting providers. If not allowed or not possible, this issue can be solved by placing a ‘wrapper’ script file in the target directory of each defined API call. This wrapper script will do nothing more than ‘include’ the main handling script, which results in it being callable at the specified directory.
- We used no modules besides the SPL, the Standard PHP Library.
- We used a standard MySQL implementation without any additional modules.
- No other additional tools were used.

With this knowledge, we decided to perform this evaluation step with an additional ‘constraint’. Instead of using a LAMP-based shared hosting environment, we decided to see if it was possible to run the LDF on Windows based shared hosting, the other (second) big platform in shared hosting. Since the LDF is PHP-based, this Windows environment would therefore need to run a Windows-built PHP version. Vevida Services\(^{51}\), a Dutch hosting company operating on several national markets, offers shared hosting on Windows servers with support for both ASP\(^{52}\) (the ‘standard’ server side scripting language on Windows servers) as PHP. We used their cheapest package ‘Web hosting Basic’, which sells at € 3,95 per month. The ‘installation’ of the application and the LDF consisted of the following steps:

- Copying of the files for the AmsterTour app to the WWW root directory.
- Copying of the files needed for the Linked Data Framework, all contained in one folder, to the subdirectory ‘/ldf/’ in the WWW root.
- Logging on to the MySQL account (included with the hosting) and importing the SQL-dump of the database tables used by the LDF.
- Changing of the database logon credentials in the ‘settings’ file of the LDF.

These steps took less then one working hour. After performing these steps, the AmsterTour application making use of the framework as well as the GUI of the LDF worked immediately at first attempt. ‘Out of the box.’ In

51 http://www.vevida.com
fact, we anticipated problems with at least the `.htaccess` file needed to configure the rewrite rule to catch calls to all ‘virtual’ endpoint URLs specified on the folder ‘/ldf/api/’, as this is an Apache specific configuration file. As it turned out, version 3 of ISAPI Rewrite\(^{53}\) (which is a similar rewrite module for Windows servers) is compatible with `.htaccess` rewrites.

This evaluation shows that the LDF can easily be used in standard shared hosting environments, which contributes to the easy deployment of Linked Data based applications and makes it possible for the fast majority of developers to use the LDF. Therefore, we have shown that the LDF meets the fifth requirement as laid out at the end of the Introduction section.

5.3 Re-using the AmsterTour app with other data

The next step in evaluating the LDF was to check what efforts were needed to make an application that uses the LDF work with another dataset. Therefore, we wanted to reuse the AmsterTour app with another dataset. As it turned out, the more challenging part in this case was finding another dataset that was suitable for the AmsterTour app. Thought the data needed to the application might be changed, we found that depending on the application or application module, there are restrictions on what data can be used. In case of the AmsterTour application, the date needs to consist of a set of ‘entries’, of which each has an image, a longitude and latitude (for location determination), something that can serve as title and something that can serve as description. Furthermore, since the AmsterTour app is designed to provide a route between the found ‘entries’, it must make sense to provide such a root for found entries. For example, a collection of airports is not very suitable, as one is unlikely to make a ground-based tour along several airports. Especially when reusing the app as a tourist service, the spatial separation of the found ‘entries’ has to be limited.

We found the ‘War monuments’ dataset\(^{54}\) of ‘The National Committee for 4 and 5 May’\(^{55}\) suitable for use with the AmsterTour app. This dataset consists of monuments to memorize victims of war and other inequity. The dataset contains links to images of the monuments as well as descriptions of them. In order to change the app we needed to perform the following steps:

- Make sure the settings in the ‘LOD Source’ component in the LDF matches the specifications of the endpoint for the selected dataset. Alternatively, one could add an extra LOD Source to the LDF if the old source is also still to be used.
- Enter a different SPARQL query that selects the desired data from the dataset. Alternatively, a new query could be added.
- Check the visual data chain to see if any mapping might need to be changed. In this case, this was not needed because we kept the output variable names in the SPARQL query the same. If a different, new query component was added to the LDF, this query should be used in the data chain instead of the original query element.

Making the AmsterTour app work with a different dataset, created and maintained by different people, took less than one hour.

This step of the evaluation shows that the settings in the LDF can be changed to make our sample application work with a different data set. By doing so, we showed that the LDF meets the third requirement as laid out at the end of the introduction section.

5.4 Using the LDF to feed an existing API-based application with Linked Data.

In the previous step of the evaluation, we had to search for another dataset to reuse the AmsterTour app. This scenario can occur ‘the other way around’ as well: When you have a nice set of data, you want to find (additional) ways to use this data. We therefore wanted to cover this other perspective in our evaluation: Re-using the data used for the AmsterTour app with another application. In addition, we also wanted to cover another possible step in the evaluation: Empowering an existing application with Linked Open Data. An application that was not originally designed to work directly with Linked Data, and that was developed before the idea of the LDF came to live.

We could combine these two evaluation steps in re-using the AgoraTouch demonstrator\(^{56}\). The AgoraTouch demonstrator is an application developed for the Agora project\(^{57}\), in order to demonstrate their event based browsing of historical multimedia objects, such as pictures and video’s. The application uses an API interface to retrieve its data and was designed for use on tablets and multi-touch tables.

This evaluation step differs from the first evaluation step, adapting the AmsterTour app to use the LDF, in the sense that in contrast to the AmsterTour app the AgoraTouch demonstrator uses an external data source (an API interface) to retrieve its data. Therefore, by ‘mimicking’ the API interface expected by the AgoraTouch demonstrator, we were able to feed data from the ‘Rijksmonumenten’ data set into the AgoraTouch demonstrator without changing a single line of code in the application. We only had to change two settings in the AgoraTouch demonstrator, containing the address of the API interface, in order for the application to address the framework instead of the original API.

We had to take the following steps in order to make the AgoraTouch demonstrator work with the data selected for the AmsterTour app:

- Examine the API interface of the demonstrator, and determine the input and output that took place on data requests. We did this simply by looking at the requests made by the application and the data


\(^{54}\) [http://www.opencultuurdata.nl/2012/06/datablog-database-oorlogsmonumenten/](http://www.opencultuurdata.nl/2012/06/datablog-database-oorlogsmonumenten/)

\(^{55}\) [http://www.4en5mei.nl/english](http://www.4en5mei.nl/english)


\(^{57}\) [http://agora.cs.vu.nl/](http://agora.cs.vu.nl/)
returned by the API, using the standard developer tools provided in browsers such as Google Chrome and/or Firefox.

- Create a WSDL schema that describes the operations and their input/output.
- Load this WSDL file into the Linked Data Framework.
- Add two queries to the framework to select the data needed to fulfill the requests. We created two queries, one to select a list of monumental buildings nearby a specified location (latitude longitude pair), and one to select details of a specific (chosen) monumental building.
- Using these two queries, we set up the data chains for the two separate data requests (‘operations’ in WSDL terminology) needed by the AgoraTouch application. One of them contains the query generating a list of objects, with the location information being the center of Amsterdam. This request is made at application startup. The second request is made when an object is selected. The data chain for this request contains both queries, the ‘details of object’ query to retrieve details of the chosen object, and the ‘list of objects’ query to select monumental buildings near to the selected monument. In order to retrieve a list of nearby monumental buildings, the latitude - longitude information of the first query is used as input (center point) for the second query. The necessary mappings were made using the mappings screen for the connection between the two queries.
- We also mapped some of the metadata available in the ‘Rijksmonumenten’ dataset to fields in the resultset that were used for metadata in the original application.
- The AgoraTouch demonstrator expects to receive JSON in return of its requests, so we selected JSON as output format at the input/output settings page for the requests.

The whole process took us about five hours. The most time consuming parts were examining the requests and data structure expected by the AgoraTouch application and finding/determining what data to map to this structure. Figure 7 shows the AgoraTouch application on a tablet, using data from the Rijksmonumenten data set.

This evaluation shows that the LDF can be used to empower existing applications with Linked data; applications that were not originally designed to work with Linked Data. We did this without changing code in the application to show the power of the data chain and the mapping tool of the LDF. This evaluation step shows the importance of the visual data chain editor and the mappings tool in the LDF, as for without these features we would not have been able to perform this evaluation step. The same holds for the support for multiple input/output formats. In this case, the application expected to receive JSON instead of XML. In this section, we extensively showed the usability of the framework and the flexibility of its settings. We showed that the LDF meets the fourth requirement as laid out at the end of the Introduction section.

5.5 Discussion with respect to the evaluation
Through these evaluations we have showed that the LDF can be used to extract the data gathering process away from the target application, and with it the added complexity of dealing with domain specific data structures, semantics and Linked Data specific technology. We showed that it is easy to adapt the data selection process using the LDF and that the framework is easy to deploy. On top of that we showed that the LDF can be used to reuse existing applications with other Linked Data.

Even though we have performed several steps of evaluation, we are aware of the fact that further evaluation will be necessary to determine if the LDF will be of use to other developers and data administrators. The 'ultimate evaluation' would therefore be to let a group of unrelated developers and data administrators use the framework and record their findings. Though this process would be very interesting, we knew beforehand that such a big operation would not be part of the scope of this limited project. For this project, the focus lays on developing a first prototype of the framework and performing basic evaluation. Even though we have performed several steps of evaluation, we are aware of the fact that further evaluation will be necessary to determine if the LDF will be of use to other developers and data administrators. The ’ultimate evaluation’ would therefore be to let a group of unrelated developers and data administrators use the framework and record their findings. Though this process would be very interesting, we knew beforehand that such a big operation would not be part of the scope of this limited project. For this project, the focus lays on developing a first prototype of the framework and performing basic evaluation. However, this extended evaluation is a good candidate for ‘future work’.

6. DISCUSSION
When we started the development of the LDF, our initial requirements consisted of the ability to perform a SPARQL query in order to gather the data to fulfill a particular data request, and to be able to map the outcome of this query to a possibly different set of named variables as expected by the application. The request, the query and the simple mapping approach together formed one component with a number of settings. During the project, we realised that the LDF would gain power on separation of these elements, incorporation of standards such as XML Schema and WSDL, and support of multiple input/output formats. We also recognised the added value of a plug-in system, to allow for the intermediate adaption of data. Having separate queries and plug-ins that might be shared among users also created the need for a more advanced data mapping solution.

Figure 7. The AgoraTouch demonstrator using data from the Rijksmonumenten dataset, without changing a single line of code.
6.1 Using the Linked Data Framework

In order for people to use the LDF, a manual will need to be made. This manual needs to include information on how to ‘install’ the LDF as well as how to use it. Installation is straightforward and consists of copying a folder with script files to the server and importing the settings database in MySQL.

With the LDF, developing new Linked Data applications can be easier for both developers and data owners. A developer, in this case, needs no specific knowledge of Linked Data or any related technology such as RDF or SPARQL. Furthermore, the developer of the application does not necessarily need to know anything about the data source, its structure and its relations to other data sources. On the other side, the ‘owner’ of the data, who has this Linked Data specific knowledge, does not need to have any knowledge of general application development. With the separation of the application and its data gathering process, the two responsibilities can also be split over different people, who can operate almost entirely independent. The two of them only need to agree on the input and output formats for the API interface. A typical scenario could look as follows:

1. A data owner or administrator would like to realize a new application to utilize the data. He or she can brainstorm about the functionality this application should have, and approach a ‘random’ application developer to develop the application.
2. The developer, having no specific knowledge of Linked Data, can tell the data administrator what data is needed to achieve the desired functionality. This can be done in simple terms such as ‘I need an array containing 10 objects, each containing a building title, description, image and architect. Then, for the second API call I will present you an array containing 10 objects, each with a building title, description, image and architect.’ etc.
3. The data administrator can use the LDF to devise queries and a mapping to deliver the desired data in the desired format.

Seen from the other side, a scenario could look like this:
1. A developer wants to use data from the Linked Data cloud in a new application. The developer has heard that the data he or she needs for the application can be found there, but has no experience with Linked Data. He or she can contact the administrator of the desired data source and ask this person for help.
2. If the data administrator is willing to cooperate, the rest of this scenario can take place as the previous one. If not, the developer can ask another Linked Data expert to help.

6.2 Limitations

The prototype implementation of the LDF fulfills the requirements we set out for it, and therefore it is a solution to the goals mentioned in the problem statement: How to ease the development, deployment and maintenance of Linked Data applications. However, we do realize that the LDF is not the only possible solution. Many other approaches could have been taken to find a solution for the problem in general, as well as for the development of the LDF in specific.

Though we evaluated the LDF in several different steps, we acknowledge the fact that further evaluation would be most useful to gather feedback from the developing community. Feedback that can be used for the development of a next version of the LDF.

The focus during this project was on the creation of a first prototype version of the LDF. Therefore, the parsers and serializers we created for XML Schema are not yet capable of dealing with every single aspect of the complete XML Schema specification. Furthermore, even though good thought was given to the design of the framework GUI, no user interface study was conducted for it. Defining the most optimal interface was not the main goal of this project.

We realize that given the direction in which the project developed, we might have entered the field of workflow design and manipulation software. We developed the ‘visual data chain’ in the LDF not because we planned to build a workflow editor, but because it turned out to be the most practical and workable solution. In addition, the visual data chain is only a part of of the LDF and not the main research goal. Therefore, we did do a thorough investigation into the field of workflow management software.

7. CONCLUSION

In this paper, we discussed the development and evaluation of an application framework for Linked Data applications. Having good tools to deal with Linked Data in applications is essential to simplify the development, deployment and maintenance of Linked Data based applications. Especially tools that operate on the boundaries between different technologies are useful, because they can provide a bridge between these technologies.

We have developed such a tool: The Linked Data Framework. The LDF can be used to extract the data gathering process away from the target application, and with it the added complexity of dealing with domain specific data schemas, semantics and Linked Data specific technology. The evaluation shows that the Linked Data Framework meets the requirements we set out for it. Therefore, it can be a valuable tool for the creation of Linked Data based applications. It can help speed-up the development of new applications by the reuse of application modules, queries and plug-ins. It makes adapting the data selection process for an application much simpler. The LDF can also be used to empower existing applications with Linked Data. By using standards such as XML Schema and WSDL, applications that were written to rely on a web service for data retrieval can be reused with Linked Data. The modular structure of the LDF, its support for standards and its ability to map data of different schemas are essential properties for it to meet its requirements. The helpfulness of the framework will grow as more developers will use it for their applications and write and share application modules and plug-ins for it.
However, even without a huge collection of sharable modules and plug-ins, the framework still adds value because of the easy data management for applications.

In addition, our efforts in developing and evaluating the LDF have provided us the insight that the integration of tools is an important factor in the success of Linked Data. The integration of tools can provide developers with a solution that they can use to discover, store and use Linked data. An example would be the integration of the LDF with a triple store and a Linked Data browser. This would create a powerful combination that allows a developer to browse the LOD cloud in search for useful data, store frequently used data locally if needed and define how this data will be used.

Finding other data sources to evaluate the LDF was more challenging than expected. In theory, many data sets are connected in one way or another. However, in practise this leaves much to be desired. Executing a query over multiple (in principle connected) data sets can be difficult, e.g. because they reside in different data stores. There are solutions to overcome these problems, e.g. the ‘Federated Query’ extension 58 to SPARQL version 1.1 (see also Hartig, Bizer & Freytag, 2009), however we want to underline that it is important that such new technology is actually implemented.

8. FUTURE WORK
The first prototype of the LDF meets the requirements we set out for it. However there is enough that remains for future work. We already addressed the added value of further evaluation. An evaluation in which we ask a number of developers to use the LDF and share with us their findings. It would be most interesting to see what they come up with.

Apart from further evaluation, there are also many implementation wishes. As pointed out in the conclusion, it would be very interesting to integrate a triple store and a Linked Data browser in the LDF, to create a tool for the complete process of data discovery, data storage and data usage. Another feature that is high on our wishlist is the option to ‘switch’ between several routes in the data chain, based on specific input data. This input data could originate from either the request input or any other previous component in the data chain. This feature will allow the LDF to travel an alternative route along queries and plug-ins, based on e.g. the outcome of a previous query. A caching solution would also be an interesting feature. Either caching of the complete end result for a request (if possible) or caching of RDF triples used to fulfill a request. Apart from new features, there are several optimizations that can be performed, such as the improvement of the XML Schema parser.

An interesting research perspective would be to investigate how the LDF can contribute to dealing with uncertainty in relation to Linked Data. For example, a system in which rules can be defined to deal with situations of missing or incomplete data.

ACKNOWLEDGEMENTS
We like to thank Victor de Boer and Lora Aroyo for their support, good advice and critical view during this project.

REFERENCES


