Künstliche Intelligenz manuscript No.
(will be inserted by the editor)

SEPAL—Schema Enhanced Programming for Linked Data

Stefan Scheglmann · Martin Leinberger · Thomas Gottron · Steffen Staab · Ralf Lämmel

Received: date / Accepted: date

Abstract The Linked Data cloud provides a large collection of interlinked data and is supposed to be seen as one, big data source. However, when developing applications against this data source, it becomes apparent that different challenges arise in the various steps of programming. Among these are the selection and conceptualization of data as well as the process of actually accessing the data. The SEPAL (Schema Enhanced Programming for Linked Data) Project provides a new approach for integrating Linked Data sources when developing Semantic Web applications. It does this by crawling the LoD cloud, analyzing the extracted data and providing this information to a developer during development through a framework that extends the programing language. In this paper, we will motivate the necessity for a project like SEPAL and explain the core components of the project.

1 Introduction

The Linked Data initiative provides billions of triples spread across thousands of interlinked data sources on the Web. Tapping this vast resource of information and incorporating it into Semantic Web applications remains a challenge, though. Linked Data is provided in a decentralized, distributed and ad-hoc manner, it is semi-structured and loosely typed in its nature. To support consumption of linked data, it would be desirable that a programmer could ignore, where relevant data is located, how it has to be retrieved, how it has to be mapped from RDF into object-oriented programming and how robustness against Quality of Service (QoS) issues with the data sources is to be realized. The objective of the SEPAL project[1] is to build an unified infrastructure and parts thereof, which allow a developer to treat the LOD cloud in his application and the application development process as what it really should be: one big source of interlinked data.

The typical data driven application development process can be broadly broken down to three steps: (1) Datasource and data discovery, (2) refinement and (3) programming. Let us illustrate this by the means of an example. Consider an application developer who builds an application depicting a network of researchers, their publications and co-authorships. The application should work on LOD sources, e.g. foaf-profiles (single URIs) and publication databases (providing a SPARQL endpoint). During every step of the development process, the developer has to cope with different problems arising from the LOD characteristics.

(1) First, we assume that the exact contents of the LOD cloud is largely unknown to the developer, e.g. he does not know the URIs of every relevant foaf-profile or other interesting data-sources. The developer has a clear vision on what kind of entities his application should work on – the developers perspective, but he is not familiar with the conceptualizations used in the sources – the perspective of the available data. Therefore the developer has to explore the LOD cloud. Technologies like semantic search [1,2] can help to identify relevant data-sources and pieces of data, gain an understanding of the classes and concrete data entities. As an example, a developer might be interested in information about researchers and scientific publications but his data-source provides him with data about persons and documents.

(2) Once all relevant data-sources and classes have been identified, the developer has to implement the concrete queries for his data. Usually applications do not

Stefan Scheglmann
Institute for Web Science and Technologies
University of Koblenz-Landau
Universittsstrasse 1
56070 Koblenz Tel.: +49-261-2872718
Fax: +49-261-2871002704
E-mail: schegi@uni-koblenz.de

[1] http://west.uni-koblenz.de/en/forschung/forschungsprojekte/sepal

query data by classes, as they are defined in the data-sources, but according to their needs, viz. the developer has to refine the classes identified in the previous step. For example, he is interested in scientific publications and researchers. Therefore, he queries for documents in specific channels to ensure that he only gets scientific publications. He also restricts persons to the set of persons that have already published a scientific document.

(3) Finally, the developer programs the object access and manipulation. Here he has to realize the data access and the data representation. The different technological infrastructures behind LOD sources, result in heterogenous access methods with different expressiveness. In order to obtain data from the different sources, e.g. the foaf-profiles from our example can be accessed resource-wise via dereferencing their URIs, but the publication database provides a SPARQL endpoint which allows for further complexity in the queries, the developer has to realize source-centric access methods or he decides to use some kind of query federation [3,4] approach as unified access layer. Additionally, he cannot make any guarantees about the Quality of Service of the data-sources, c.f. C. Buil-Aranda et al. [11,16]. For example, a high workload or an outage of a server could lead to availability problems of single foaf profiles or a whole publication database. The application has to have some means of robustness against these kind of Quality of Service problems [10]. In this case a QoS aware source selection [9,10] might be of use. Once data access is realized, the programmer has to realize the data representations in his code. Triple-object mapping approaches [5–8] try to automate this step of integration.

Though many of the issues arising in these steps have appeared in the past in conventional information system settings, the novel characteristics of the LOD cloud - embarrassing distributedness, heterogeneity, variance of QoS - makes it necessary to rethink previous solutions in order to deliver a valid platform to LOD programmers. For most of the problems arising in those different steps, solutions exist. But most of these solutions only deal with single aspects of the whole workflow. Ideally, a developer would expect an application development infrastructure, that abstracts from all of these challenges and allows him to integrate LOD as one data source used in his application. The goal of this project is to identify unsolved challenges in LOD application development and come up with an integrated infrastructure for a unified approach to LOD application development. In the remainder of this article, we introduce the SEPAL framework and describe how particular parts of it are realized.

2 The SEPAL Framework

The SEPAL project aims to support the developer in all three stages of the development process. Using an index of datasets and instances built by crawling the LoD cloud, it allows the developer to work on the whole LoD cloud and therefore enables the developer to work without having to search and pick specific sources manually. Assuming that the developer relies on modern programming languages and their tools, such as an IDE, the remaining problems are best solved in combination with these. SEPAL uses LITEQ (Language Integrated types, queries and extensions), a library and IDE extension to provide the supporting functionality. LITEQ allows data to be conceptualized as needed directly in the IDE, meaning that the developer can formulate restrictions, such as the foaf-persons who have published a scientific document, directly in the IDE as a datatype for his programming language. In order to properly support the developer with autocompletion and error checking, LITEQ needs schematic information. The data in the LoD cloud is analyzed using the schema inferencing provided by SchemEx on the crawled data. This analysis is requested by LITEQ while the developer creates his conceptualizations.These can then be used to do non-trivial programming tasks while data access and QoS problems are handled by LITEQ using the data stored in the index. The architecture of the SEPAL framework is visualized in Figure 1.

2.1 Data-source and data discovery

As described in the introduction, the first step of programming against the LOD cloud is the selection of a data source. The developer has to search for a dataset that caters to his information need. SEPAL solves this step by crawling the LOD cloud and creating an index of datasets and instances. The index is stored together with meta-information about the datasets and instances, such as whether the data is resolvable via URI dereferencing or SPARQL endpoints. The SEPAL framework virtualizes the access to the data and therefore allows for a unified access mechanism.

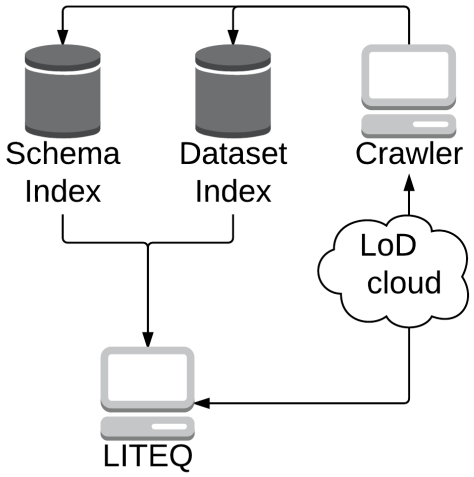

Fig. 1 Architecture of the SEPAL system consisting of LITEQ, a dataset and instance index and SchemEx.

In the example, the developer needs a dataset that contains information about persons that published some scientific publication. The developer can search for datasets and immediately access it as all necessary metadata is stored in the index.

2.2 Data conceptualization and refinement

First, the developer will use queries and conceptualizations to select the parts of the data that he actually needs. In the SEPAL framework, this is done in the LITEQ [15] component using the graph-traversal language NPQL. NPQL queries have two basic semantics associated with them. The first is an intensional semantics which is used for conceptualization of the data. The intensional semantics specifies the declaration of the datatype in the programming language and aims at capturing the intensional definition of the corresponding ontology construct (class or property). The second semantics is a extensional one. This is used for querying the data parts that the developer needs based on the concepts he defined and returns sets of objects the types of which correspond to the datatype defined by the syntactically identical intensional query.

In terms of the example, the developer can define a concept "researcher" based on the *foaf:Person* class and an *authorOf* property. An example of this in NPQL and the F# implementation is depicted in Figure 2.

```
type Researcher =
    LoD.NPQL.foafPerson.``<-exAuthorOf-``.exPublication
```

Fig. 2 Capturing the concept *foaf:Person* $\cap$ $\exists$*ex:authorOf.ex:Publication* in NPQL (F# implementation).

Such a user-driven conceptualization can be overly selective. The developer can easily describe concepts that don't conform to any instance, e.g. by using properties that are not actually used in the data. LITEQ aims to reject such conceptualizations and to provide useful feedback, e.g. suggestions for properties that can be combined into concepts without producing empty extensions. To enable this, schematic information inferred directly from the data as found in the datasets is necessary. This is provided by the Schema index component, which is implemented by SchemEx [13]. SchemEx uses the data gathered by crawling the LoD cloud to observe the property and type combinations that can occur in the different datasets. This knowledge is then used by LITEQ to give feedback to the developer during the concept and query writing phase of the development process. This is visualized in Figure 3.

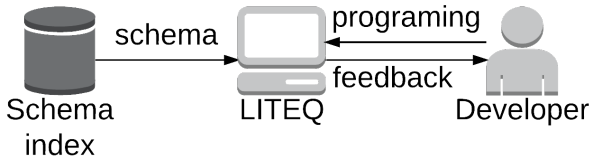

Fig. 3 Data refinement through LITEQ and schema index.

2.3 Programming

The actual data access and manipulation is provided by the API of LITEQ. Most importantly, it provides the ability to access all resources that conform to the concepts as defined in the previous step. All data access - no matter whether the resources can be accessed through HTTP dereferencing or SPARQL queries - is encapsulated by LITEQ, allowing the developer to focus on the actual programming logic. All meta-information necessary for this feature has been gathered during the crawling of the LoD cloud and stored in the Dataset index. Furthermore the API alleviates the mentioned QoS problems by working with asynchronous and par-

allel lists. When accessing data, requests are sent in parallel. Responding instances are appended to a list while non-responding ones are ignored. In order to support the responsiveness of applications, the developer can either work asynchronously with the list or wait for all requested resources to respond. Figure 4 depicts a trivial example in which the developer uses the previously created concept and iterates through it.

```
let setOfResearchers = Researcher.Extension
setOfResearchers.Task.Wait()
for researcher in setOfResearchers.Results do
    printfn "%A" researcher
```

Fig. 4 Programming with a conceptualization created in LITEQ.

3 Summary

While the web of data is continually growing, tapping into this vast resource remains a challenge. Especially developing against the LoD cloud is non-trivial. The SEPAL project attempts to change that. It provides a framework in which developers can work on the complete LoD cloud, formulate and refine queries as well as conceptualizations and focus on programming the application logic using these. This is realized by continuously crawling the LoD cloud and storing the necessary meta- and schematic information. A developer then accesses and works with this information through his programming environment. Using this approach, the SEPAL projects aims to ease the life of developers and to ensure that data from the LoD cloud is as easy to use as it should be. The frontend (LITEQ, NPQL) is an outcome of a previous research project sponsored by Microsoft Research. The backend infrastructure (schema and data index, crawler) is currently developed in-house.

References

1. L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari D. Vishal and J. Sachs Swoogle: a search and metadata engine for the semantic web. In *Proceedings of the thirteenth International Conference on Information and Knowledge Management – 2004*, pages 652-659, ACM
2. T. Gottron and A. Scherp, B. Krayer and A. Peters Lodatio: using a schema-level index to support users infinding relevant sources of linked data In *Proceedings of the seventh international conference on Knowledge capture – 2013*, pages 105-108, ACM
3. A. Schwarte, P. Haase, K. Hose, R. Schenkel and M. Schmidt FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *Proceedings of the 10th International Semantic Web Conference – 2011*, volume 7031, pages 601-616, Springer-Verlag
4. O. Görlitz, Olaf and S. Staab SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions In Proceedings of the Second *International Workshop on Consuming Linked Data (COLD2011)*, Bonn, Germany, October 23, 2011
5. E. Oren, R. Delbru, S. Gerke, A. Haller, and S. Decker. Activerdf: object-oriented semantic web programming. In *WWW2007*, pages 817–824. ACM, 2007.
6. F. S. Parreiras, C. Saathoff, T. Walter, T. Franz, and S. Staab. 'a gogo: Automatic Generation of Ontology APIs. In *ICSC2009*. IEEE Press, 2009.
7. A. Kalyanpur, D. J. Pastor, S. Battle, and J. A. Padget. Automatic Mapping of OWL Ontologies into Java. In *SEKE2004*, 2004.
8. S. Scheglmann, A. Scherp, and S. Staab. Declarative representation of programming access to ontologies. In *ESWC2012*, volume 7295 of *LNCS*, pages 659–673. Springer, 2012.
9. X. Wang, T. Vitvar, M. Kerrigan and I. Toma A QoS-Aware Selection Model for Semantic Web Services In Proceedings of the *4th International Conference on Service Oriented Computing (ICSOC) – 2006*, pages 390-401, Springer
10. S. Scheglmann, G. Gröner, S. Staab, and R. Lämmel Incompleteness-aware programming with RDF data In *2013 Workshop on Data Driven Functional Programming, DDFP*, pages 11–14, Rome, Italy, January 22, 2013
11. C. Buil-Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbusshe. Sparql web-querying infrastructure: Ready for action? In *Proceedings of the 12th International Semantic Web Conference, Sydney, Australia*, 2013.
12. S. Homoceanu and p. Wille and W.-T. Balke ProSWIP: Property-Based Data Access for Semantic Web Interactive Programming. In Proceedings of the 12th *International Semantic Web Conference – 2013*, volume 8218 of *Lecture Notes in Computer Science*, pages 184-199, Sydney, Australia, Oct. 2013. Springer-Verlag.
13. M. Konrath,T. Gottron, S. Staab and A. Scherp SchemEX—Efficient Construction of a Data Catalogue by stream-based Indexing of Linked Data In Journal of Web Semantics, volume 16, pages 52–58, 2012, Elsevier
14. N. Beck, S. Scheglmann and T. Gottron LinDA: A Service Infrastructure for Linked Data Analysis and Provision of Data Statistics In *The Semantic Web: Extended Semantic Web Conference ESWC – 2013 Satellite Events*, Montpellier, France, May 26-30, 2013
15. M. Leinberger, S. Scheglmann, R. Lämmel, S. Staab, M. Thimm and E. Viegas Semantic Web Application Development with LITEQ In Proceedings of the *13th International Semantic Web Conference – 2014*, Riva del Garda, Italy, October 19-23, 2014.
16. A. M. Intizar and A. Mileo How Good Is Your SPARQL Endpoint, In *On the Move to Meaningful Internet Systems, OTM – 2014*, pages 491-508, Springer