LITEQ: Language Integrated Types, Extensions and Queries for RDF Graphs

Stefan Scheglmann Martin Leinbereger Steffen Staab
University of Koblenz-Landau
{schegi, staab, }@uni-koblenz.de

1. Introduction

RDF data representations are flexible and extensible. Even the schema of a data source can be changed at any time by adding, modifying or removing classes and relationships between classes at any time. While this flexibility facilitates the design and publication of linked data on the Web, it is rather difficult to access and integrate RDF data in programming languages and environments because current programming paradigms expect programmers to know at least structure and content of the data source.

Therefore, a programmer who targets the access of linked data from a host programming language must overcome several challenges. (i) Accessing an external data source requires knowledge about the structure of the data source and its vocabulary. As linked data sources may be extremely large and the data tend to change frequently, it is almost impossible for programmers to know the structures at the time before they develop their programs. Therefore, approaches to simplify access to RDF sources should include a mechanism for exploring and understanding the RDF data source. (ii) There is an impedance mismatch between the way classes (types) are used in programming languages compared to how classes structure linked data. (iii) A query and integration language must be readable and easily usable for an incremental exploration of data sources. (iv) When code in a host language describes how RDF data is to be processed by the resulting program, the RDF data should be typed and type safety should be ensured in order to avoid run time errors and exceptions.

To address these challenges, we present LITEQ, a paradigm for querying RDF data, mapping it for use in a host language, and strongly typing it for taking full advantage of advanced compiler technology. In particular, LITEQ comprises: (1) The node path query language (NPQL), which has an intuitive syntax with operators for the navigation and exploration of RDF graphs. In particular, NPQL offers a variable free notation, which allows for incremental writing of queries and incremental exploration of the RDF data source by the programmer. (2) An extensional semantics for NPQL, which clearly defines the retrieval of RDF resources and allows for their usage at development time and run time. (3) In intensional semantics for NPQL, which clearly defines the retrieval of RDF schema information and allows for its usage in the programming environment and host programming language at development time, compile time and run time. Our integration of NPQL into the host language allows for static typing — using already available schema information from the RDF data source — making it unnecessary for the programmer to manually re-create type structures in the host language.

DDFP'14, January 25, 2015, San Diego, CA, United States.

2. Langauge Integrated Types, Extensions and Queries for RDF Graphs

To illustrate the challenges and the contributions of LITEQ, we consider the following scenario. Bill has to create an application for a municipal administration that allows to manage dogs that live in this city. This application should offer the user two different main functionalities, (i) managing all registered dogs in the community. This includes browsing, adding removing and editing all dogs from within the application. (ii)A tax reminder function, that addresses all dog owners and reminds them to pay their tax. All data about owners, dogs, etc. are published as RDF data assuming the schema in (Example 1). It defines three classes: `Dog` (line 2) and `Person` (lines 3) are subclasses of `ex:Creature` (line 1) (line 12) and two properties / predicates: `ex:has:Owner` (lines 4-6) and `ex:hasName` (lines 7-9)

Listing 1. The RDF Schema

```
1 ex:Creature rdfs:subclassOf rdf:Resource.
2 ex:Dog rdfs:subclassOf ex:Creature.
3 ex:Person rdfs:subclassOf ex:Creature.
4 ex:owns rdf:type rdfs:Property;
5    rdfs:domain ex:Person;
6    rdfs:range ex:Dog.
7 ex:hasName rdf:type rdfs:Property;
8    rdfs:domain ex:Creature;
9    rdfs:range rdfs:Literal.
```

In order to realize the functionalities mentioned above, Bill need three specific types. For functionality (1), Bill needs types to represent "dog" entities and for the tax reminder functionality (2), he needs a "person" representation. To be able to create these types, he has to perform several tasks: (T1) **Data exploration:** At the beginning, the structure and the content of the data source are completely unknown to him. Bill needs to use some tools or query languages in order to explore the data source and find out which content is important for the functionalities in his application. (T2) **Type exploration:** Once Bill has decided to create a type to represent a certain subset of the data source, he has to decide for a signature for that type. (T3) **Type creation:** Once Bill, has decided for several types, he has to implement them.

The objective of LITEQ is to support Bill in all aspects of the integration of the unknown RDF data source into his application. LITEQ is currently realized using the F# Type Providers. It allows to embed NPQL expressions into the F# host language for source exploration, type and entity definition. The current implementation of LITEQ includes operators for (i) refining a class to a subclass (`subType`), (ii) refining a class to a subclass that also has a certain property (`propertySel`), (iii) navigating from a class to one of its instances (`Extension`) and (iv) navigating from a class via a property to another class (`propertyNav`). Starting at the canonical root of the RDF graph, i.e. `rdf:Resource` Bill can navigate the RDF graph using these operators. Depending on the used operator and the current expression, (here: `rdf:Resource`) different views are shown: (i) Using (`subType`) navigation presents Bill a class view showing him all immediate subclasses. (ii) The (`propertyNav`) and (`propertySel`) presents a property view showing him all the properties that have the current expression (here: rdf:Resource) as their domain.(iii) (`Extension`) shows him an instance view showing him all instances of the currently selected class (`rdf:Resource`). Using these three views for NPQL-based autocompletion, Bill may incrementally and interactively explore the data source by writing node paths. He can easily navigate down to dog or person, to define these types (the intension of the node path) for his application. Or work on the set of all individuals (the extension of the node path), e.g all persons who owns dogs for the tax reminder.

The full definition of NPQL syntax and NPQL semantics is skipped here for brevity, please refer to our technical report[1].

3. Related Work

There is related research on mappings between Web data sources and (typed) programming languages. LITEQ benefits from type providers in F# that support the integration of information sources into F# [10] such that external data sources are directly available in programs. Type provider use F# LINQ queries [4] to retrieve schema information from (Web) data sources in order to build the corresponding types at run-time. Several Type Provider demonstrate the integration of large data sources on the Web, like the Freebase Type Provider that allows for the navigation within the graph-structure of Freebase [2].

The problem of accessing and integrating linked data in programming environments has already been recognized as a challenge in various work. Frameworks like OntoMDE [9] or Àgogo [7] focus on ontology driven code generation. An overview can be found at Tripresso[3], a project web site on mapping RDF to the object-oriented world. All of frameworks, metioned there have in common that they translate the concepts of the ontology into an object-oriented representation. However, compared to LITEQ, they do not consider the exploration of data sources.

There exists basic mapping principles of RDF triples to objects in object-oriented programming languages [**?**] and programming language extensions to integrate RDF or OWL constructs [6]. In contrast to LITEQ, there is no means for querying and navigating unknown data sources, instead, the developer must be aware of the structure of the ontology.

Other research work has a dedicated focus on exploration and visualization of Web data sources. tFacet [1] and gFacet [3] are tools for faceted exploration of linked data sources via SPARQL enpoints. The navigation of RDF data for the purpose of visualizing parts of the data source is studied in [2], but these approaches do not consider any kind of integration aspects like code generation and typing.

[1] Technical Report: `http://west.uni-koblenz.de/Research/systems/liteq`

[2] The Freebase Wiki about the Schema: `http://wiki.freebase.com/wiki/Schema`

[3] `http://semanticweb.org/wiki/Tripresso` last visit Nov 10, 2010

4. Conclusion

In this talk, we present several challenges a programmer must overcome to access and integrate external RDF data sources in his application. As a solution to these challenges, we present our integrated approach LITEQ and our language NPQL. A language to integrate, explore and query linked data via SPARQL endpoints from within programming environments. We discuss the syntax of the language, its usage and its semantics at development, compile and run time of programs. This discussion may be helpful in conducting future research on programming models and programming environment integrated query languages for accessing and processing large amounts of semantically rich data.

Acknowledgments

This work has been supported by Microsoft.

References

[1] Sren Brunk and Philipp Heim. tFacet: Hierarchical Faceted Exploration of Semantic Data Using Well-Known Interaction Concepts. In *International Workshop on Data-Centric Interactions on the Web (DCI 2011)*, volume 817 of *CEUR-WS.org*, pages 31–36, 2011.

[2] Jirí Dokulil and Jana Katreniaková. Navigation in RDF Data. In *12th International Conference on Information Visualisation*, pages 26–31. IEEE Computer Society, 2008.

[3] Philipp Heim, Jürgen Ziegler, and Steffen Lohmann. gFacet: A Browser for the Web of Data. In *International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW 2008)*, volume 417 of *CEUR-WS*, pages 49–58, 2008.

[4] Erik Meijer, Brian Beckman, and Gavin Bierman. LINQ: Reconciling Object, Relations and XML in the .NET Framework. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, page 706, Chicago, Illinois, June 2006. ACM Press.

[5] Eyal Oren, Renaud Delbru, Sebastian Gerke, Armin Haller, and Stefan Decker. Activerdf: object-oriented semantic web programming. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 817–824. ACM, 2007.

[6] Alexander Paar and Denny Vrandecic. Zhi# - OWL Aware Compilation. In *The Semanic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29 - June 2, 2011, Proceedings, Part II*, volume 6644 of *LNCS*, pages 315–329. Springer, 2011.

[7] Fernando Silva Parreiras, Carsten Saathoff, Tobias Walter, Thomas Franz, and Steffen Staab. 'a gogo: Automatic Generation of Ontology APIs. In *IEEE Int. Conference on Semantic Computing*. IEEE Press, 2009.

[8] Tirdad Rahmani, Daniel Oberle, and Marco Dahms. An adjustable transformation from owl to ecore. In Dorina C. Petriu, Nicolas Rouquette, and ystein Haugen, editors, *MoDELS (2)*, volume 6395 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2010.

[9] Stefan Scheglmann, Ansgar Scherp, and Steffen Staab. Declarative representation of programming access to ontologies. In Elena Simperl, Philipp Cimiano, Axel Polleres, scar Corcho, and Valentina Presutti, editors, *ESWC*, volume 7295 of *LNCS*, pages 659–673. Springer, 2012.

[10] D. Syme, K. Battocchi, K. Takeda, D. Malayeri, J. Fisher, J. Hu, T. Liu, B. McNamaa, D. Quirk, M. Taveggia, W. Chae, U. Matsveyeu, and T. Petricek. F# 3.0 — Strongly Typed Language Support for Internet-Scale Information Sources. Technical Report MSR-TR-2012-101, Microsoft Research, 2012.