How to "Make a Bridge to the New Town" Using OntoAccess

Matthias Hert, Giacomo Ghezzi, Michael Würsch, and Harald C. Gall

s.e.a.l. – software architecture and evolution lab
Department of Informatics
University of Zurich, Switzerland
{hert,ghezzi,wuersch,gall}@ifi.uzh.ch

Abstract. Business-critical legacy applications often rely on relational databases to sustain daily operations. Introducing Semantic Web technology in newly developed systems is often difficult, as these systems need to run in tandem with their predecessors and cooperatively read and update existing data.

A common pattern is to incrementally migrate data from a legacy system to its successor by running the new system in parallel, with a data bridge in between. Existing approaches that can be deployed as a data bridge in theory, restrict Semantic Web-enabled applications to read legacy data in practice, disallowing update operations completely.

This paper explains how our RDB-to-RDF platform OntoAccess can be used to transition legacy systems into Semantic Web-enabled applications. By means of a case study, we exemplify how we successfully made a bridge between one of our own large-scale legacy systems and its long-term replacement. We elaborate on challenges we faced during the migration process and how we were able to overcome them.

1 Introduction

The field of software engineering is in a constant state of flux. New paradigms, programming languages, frameworks, and tools gain tremendous momentum all of a sudden – and then they sink into oblivion as quickly as they have emerged. Short time-to-marked intervals are therefore critical for the success of new tools, be it in an industrial context or in research.

In contrast to short-lived tools, the body of acquired knowledge of a company usually evolves less rapidly and sometimes even remains relevant for decades, stored as data in different, mostly relational databases (RDB). This inevitably leads to challenges, when different generations of applications have to operate on this data.

There are legacy systems relying on a relational view of the database—these applications can not easily be upgraded or simply taken offline and thrown away when requirements change. Legacy systems are often crucial for daily operations and therefore need to be highly available. They are inherently valuable to many organizations but bear typical problems: Maintenance and especially further development have become difficult and costly.

These circumstances and new business opportunities emerging with the advent of paradigms, such as Service-Oriented Architectures and the Semantic Web, lead to the development of next-generation systems. While new development opens the door for incorporating recent best-practices and state-of-the-art technologies, the newly developed applications usually will run in tandem with legacy systems and still need to access legacy databases.

In such scenarios, it is common to *make a bridge to the new town*, that is, to incrementally migrate data from a legacy system by running the new system in parallel, with a data bridge in between [8]. Tools such as D2R Server [4] and OpenLink Virtuoso [9] serve RDF views on relational databases. However, they restrict Semantic Web-enabled applications to read legacy data, disallowing update operations completely.

In this paper, we describe how our RDB-to-RDF platform OntoAccess [19] can be used to facilitate the transition from legacy systems to Semantic Web-enabled applications in practice. OntoAccess provides a semantic layer on top of existing relational databases. It enables RDF-based read and write access to relational data. Based on mappings that bridge the conceptual gap between RDF and the relational model, a mediator translates Semantic Web requests on-the-fly to SQL. This enables relational and RDF-based applications to cooperate on the same data and to further exploit the advantages of the well established database technology such as query performance, scalability, transaction support, and security.

The contribution of this paper is a case study on how we successfully used OntoAccess to advance our Eclipse-based software evolution analysis framework Evolizer [13] to Sofas [17], a service-oriented, distributed, and collaborative software analysis platform. We describe use cases where existing RDB-to-RDF approaches are insufficient and an approach such as OntoAccess is needed.

The remainder of this paper is structured as follows. Section 2 gives a brief introduction to the two systems between which OntoAccess acts as a data-bridge: Evolizer and Sofas. OntoAccess itself bridges the conceptual gap between the relational model and RDF. It is described in Section 3. In Section 4, we present the case study on how we successfully used OntoAccess to advance Evolizer to Sofas. Related work in the context of RDB-to-RDF mapping is reviewed in Section 5. Section 6 concludes this paper with a summary.

2 Background

In this section, we describe our two platforms for software analysis that run in tandem and are able to share data thanks to OntoAccess. The first platform, Evolizer, is considered to be a legacy system, whereas Sofas represents our latest ambitions in providing a scalable, distributed means to analyze the evolution of a software system.

2.1 Evolizer

In the past, we have developed Evolizer [13] – a plug-in-based software evolution analysis and research platform, tightly woven into the Eclipse IDE.

At its core, Evolizer is based on the idea of a *Release History Database (RHDB)* [11]. It is implemented as a set of Eclipse plug-ins and integrates information originating from different software repositories, such as version control, issue tracking, mailing lists, etc. The combination of this diverse, yet interconnected data allows one to uncover and analyze the many different facets of the evolution of a software system and its parts. Examples are the system's fine-grained change history or bug-proneness over time, as well as a complete source code model.

Evolizer has become a typical legacy system over time: While the platform is still in active use, it becomes harder to adapt it to new requirements and recent advances in technology. The tight coupling to Eclipse makes it hard to adapt and re-use Evolizer's tools and algorithms in new environments such as in a service-oriented context. Further, the RHDB is based on classical relational database technology. It is therefore difficult to interlink information stored in the RHDB with other external data sources, because the relations that we store are local – not universal – and our entities lack unique resource identifiers that can be dereferenced over the Internet. Synergies with related approaches are therefore difficult to exploit. Evolizer's models also lack explicit semantics, such as cardinality, transitivity, symmetry, and so on. Bringing Evolizer and its RHDB to the Semantic Web would therefore be desirable.

Evolizer is still very valuable and our RHDB contains data about the software life-cycle of hundreds of systems. Re-importing this vast amount of data from version control and bug tracking systems would take months, and some of these repositories might not even be available online anymore.

To overcome these limitations, we are in need of a gradual migration path from Evolizer to the next generation of software evolution analysis platforms, allowing us to run the existing platform together with its replacement for the years to come.

2.2 SOFAS

Evolizer allowed us to combine and analyze different aspects of a software's evolution and its development. However, we realized that a big potential lies in having analyses easily accessible and composable, without platform and language limitations, and not having to install and configure particular tools. Based on these premises, we introduced the concept of "Software Analysis as a Service" [16]: getting easy access to different analyses from various tools and providers using Web services. We implemented that concept into a lightweight and flexible platform called Sofas (SOFtware Analysis Services) [17].

Sofas follows the principles of a RESTful architecture [10] and allows for a simple yet effective provisioning and use of software analyses based upon the principles of Representational State Transfer around resources on the Web. Its architecture is made up by three main constituents: Software Analysis Web Services, a Software Analysis Broker, and Software Analysis Ontologies. The services expose their functionalities and data through standard RESTful Web service interfaces. The Software Analysis Broker acts as the service manager and provides the interface between the services and the users. It contains a catalog of all the registered analysis services with respect to a specific software analysis taxonomy. As such, the domain of analysis services is described in a semantic way, enabling users to browse and search for their analysis service of interest. The ontologies – we call them SEON (*cf.* Section 4.2) – are used to define and represent the RDF data consumed and produced by the different services.

3 OntoAccess as a Bridge to the New Town

OntoAccess is a RDB-to-RDF mediation platform that enables Semantic Web-based applications to operate on relational data. It provides a semantic layer on top of existing relational databases to enable RDF-based read and write access to the relational data. Semantic Web requests, *i.e.,* query and update requests, are translated on-the-fly to SQL for execution in the database. OntoAccess therefore eliminates the need for mirroring and synchronizing the relational data and the RDF representation – both data models always operate on the same state of the data. This results in a cooperative use of the data in RDF-based as well as in relational applications. In addition, mediation allows one to further exploit the advantages of the well-established database technology such as query performance, scalability, transaction support, and security. The existing, read-only RDB-to-RDF mapping approaches are limited to data warehouse-like applications where the data can be queried and analyzed but not modified. In comparison, OntoAccess puts relational databases on par with native RDF triple stores by allowing read and write access to the data. This facilitates the transition from RDB-based legacy systems to Semantic Web-enabled applications in practice.

3.1 Architectural Principles of OntoAccess

OntoAccess is designed and implemented as an extensible platform. It encapsulates the RDB-to-RDF translation logic in the core layer which provides the foundation for an extensible set of data access interfaces in the interface layer. The RDB-to-RDF core is responsible for the translation of RDF-based request to SQL and interacts with the database system. The interface layer exposes the functionality of the OntoAccess core to RDF-based applications via different data access approaches. It translates the interface-specific operations to the basic OntoAccess operations, and results back into the interface-specific format. This facilitates the development of additional data access interfaces because the main RDB-to-RDF translation work is performed in the core layer. Currently, the OntoAccess platform supports data access via SPARQL [23], SPARQL/Update [24], Linked Data [2], and various Semantic Web Frameworks,

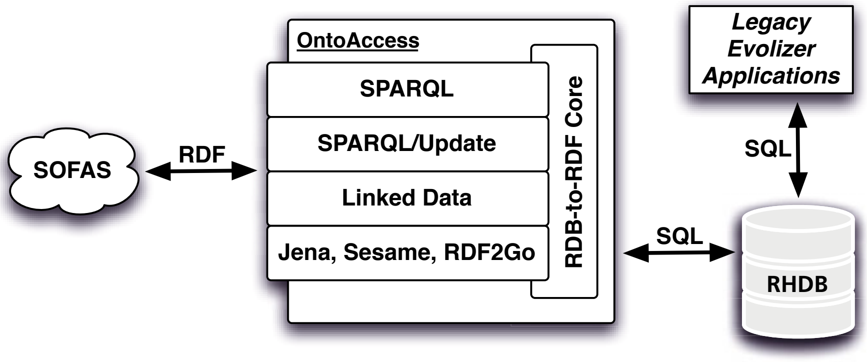

Fig. 1. OntoAccess Architecture Overview

such as Jena, Sesame, and RDF2Go.[1] The data access interfaces are accessible via a HTTP service endpoint if deployed as a stand-alone server. Alternatively, OntoAccess can be integrated into other applications as a library, in which case the data access interfaces are exposed via specific APIs. Figure 1 presents an overview of the overall architecture of OntoAccess and exemplifies how it can be used as a data-bridge in the context of Sofas and the Evolizer RHDB. Next, before we discuss this example in detail in Section 4, we elaborate on the mapping principles of OntoAccess.

3.2 Mapping Principles of OntoAccess

Mediation requires a mapping from concepts in a relational database schema to terms defined in an ontology. For OntoAccess, we developed R3M [20] as a bidirectional RDB-to-RDF mapping language that incorporates the requirements of RDF-based write access to relational databases. Existing mapping languages developed for read-only use cases are unsuitable for write access as shown in [15] and by D2R/Update.[2]

R3M extends the mapping approach described in [3]. Tables of the database schema are mapped to classes in an ontology and the attributes of those tables to properties. Special support is provided for *link tables* that are used to represent M:N relationships in the relational model. As such helper constructs are not needed in RDF, link tables are mapped to properties instead of classes. In addition, R3M mappings contain information about datatypes, as well as integrity constraints of the database schema. This results in a mapping language that is not as expressive as the existing, read-only languages (*cf.* [21]) but it is sufficient to cover many application scenarios, including the one presented in this paper. In general, R3M is targeted at mapping highly normalized relational database schemata such as the ones generated by object-relational mappers

[1] `http://openjena.org/`, `http://openrdf.org/`, `http://rdf2go.semweb4j.org/`
[2] `http://d2rqupdate.cs.technion.ac.il/`

Listing 1.1. Example R3M Mappings

```
 1  a)  ex:revision  a  r3m:TableMap;
 2        r3m:hasTableName   "Revision";
 3        r3m:mapsToClass    ver:Version;
 4        r3m:uriPattern     "http://.../revision_%%number%%";
 5        r3m:hasAttribute   ex:revision_number, ....
 6
 7  b)  ex:revision_number  a  r3m:AttributeMap;
 8        r3m:hasAttributeName       "number";
 9        r3m:mapsToObjectProperty   ver:hasID;
10        r3m:dbType                 [ a  r3m:VarChar;
11                                     r3m:length 255 ];
12        r3m:hasConstraint          [ a  r3m:NotNull ].
13
14  c)  ex:release_revision  a  r3m:LinkTableMap;
15        r3m:hasTableName           "Release_Revision";
16        r3m:mapsToObjectProperty   ver:comprises;
17        r3m:hasSubjectAttribute    ex:rr_release;
18        r3m:hasObjectAttribute     ex:rr_revision.
```

(*e.g.,* Hibernate[3]), and at the so-called *direct mapping* where an equivalent RDF representation of the relational data is needed (*e.g.,* for use cases as described in [12]).

Listing 1.1 presents examples of the three main mapping constructs in R3M. The namespace prefixes used in the examples are defined as follows: `r3m` represents our mapping language vocabulary `http://ontoaccess.org/r3m/` while `ex` is used for the namespace `http://example.com/mapping/` of our example mapping. `ver` represents the namespace of the SEON version control ontology `http://evolizer.org/ontologies/seon/2010/03/versions.owl` (*cf.* Section 4.2). Listing 1.1a) depicts a *TableMap* representing the mapping of a database table to a class in the ontology. A *TableMap* contains the name of the table (line 2) and the class it is mapped to (line 3). The URI pattern (line 4, abbreviated) is used to generate the URIs for instances of this table based on values of table attributes that are specified between double percentage signs (e.g. %%*number*%% where *number* is the name of an unique attribute such as the primary key). A *TableMap* further contains a list of *AttributeMap*s (line 5, abbreviated).

Listing 1.1b) presents an example of an *AttributeMap* that maps a database attribute to a property in the ontology. An *AttributeMap* contains the name of the attribute in the database schema (line 8) and the property it is mapped to (line 9). Additionally, an *AttributeMap* includes information about the datatype of the database attribute (lines 10 and 11) as well as information about (database) constraints defined on that attribute (e.g., a not null constraint; line 12). R3M

[3] `http://hibernate.org/`

supports the constraints `r3m:PrimaryKey`, `r3m:ForeignKey`, `r3m:NotNull`, `r3m-:Default`, and `r3m:Check`.

Listing 1.1c) shows a *LinkTableMap* representing the mapping of a link table to an ontology property. A *LinkTableMap* specifies the name of the link table in the database (line 15) and the property it is mapped to (line 16). A link table always contains two foreign key attributes that point to the tables of the N:M relationship. These attributes are represented as *AttributeMap*s (line 17 and 18) that provide the names of the attributes, the foreign key references to the tables, and the direction of the relationship (from subject to object).

4 A Case Study on Bridging Software Analysis Data

To motivate our case study, we present use cases that require interoperability between Evolizer and Sofas. These use cases need a bidirectional data exchange, *i.e.,* from Evolizer to Sofas and vice versa.

First, Evolizer contains data about hundreds of software systems that were imported over the past years (*cf.* Section 2.1). The Sofas platform needs to be able to access this data without the need for re-importing it. This requires RDF-based **read access** to the Evolizer database. Second, Evolizer implements importers to import source code and history data from centralized version control systems, such as CVS and SVN. Lately decentralized version control systems, such as Git or Mercurial, gained popularity. Therefore, respective import services were developed for the Sofas platform. The data produced by these importer services is modeled in RDF, based on the SEON ontologies described in Section 4.2. This data is also valuable to Evolizer because existing tools could be used to leverage it. This, however, requires RDF-based **write access** to the Evolizer database. Lastly, Sofas implements an extensible framework to compute software metrics on the data. Again, this data is modeled in RDF, but matching relations are available in the Evolizer database schema. RDF-based write access to the RHDB is needed to make the metrics data available to Evolizer.

These use cases indicate that, for making a bridge between Evolizer and Sofas, a RDB-to-RDF mapper is needed that provides RDF-based read and write access to relational data. Whereas existing approaches are limited to read-only queries, we developed OntoAccess that provides read and write access. In the remainder of this section, we present the relational data model of Evolizer, the ontology-based data model of Sofas, and how the mapping of OntoAccess is used to bridge the conceptual gap between these two data models. We further discuss challenges we encountered during this case study.

4.1 Data Schema of Software Analysis within Evolizer

The data schema of Evolizer consists of several distinct parts covering many aspects of the Software Engineering domain. For this case study, we focus on those parts that are concerned with historical aspects and with source code

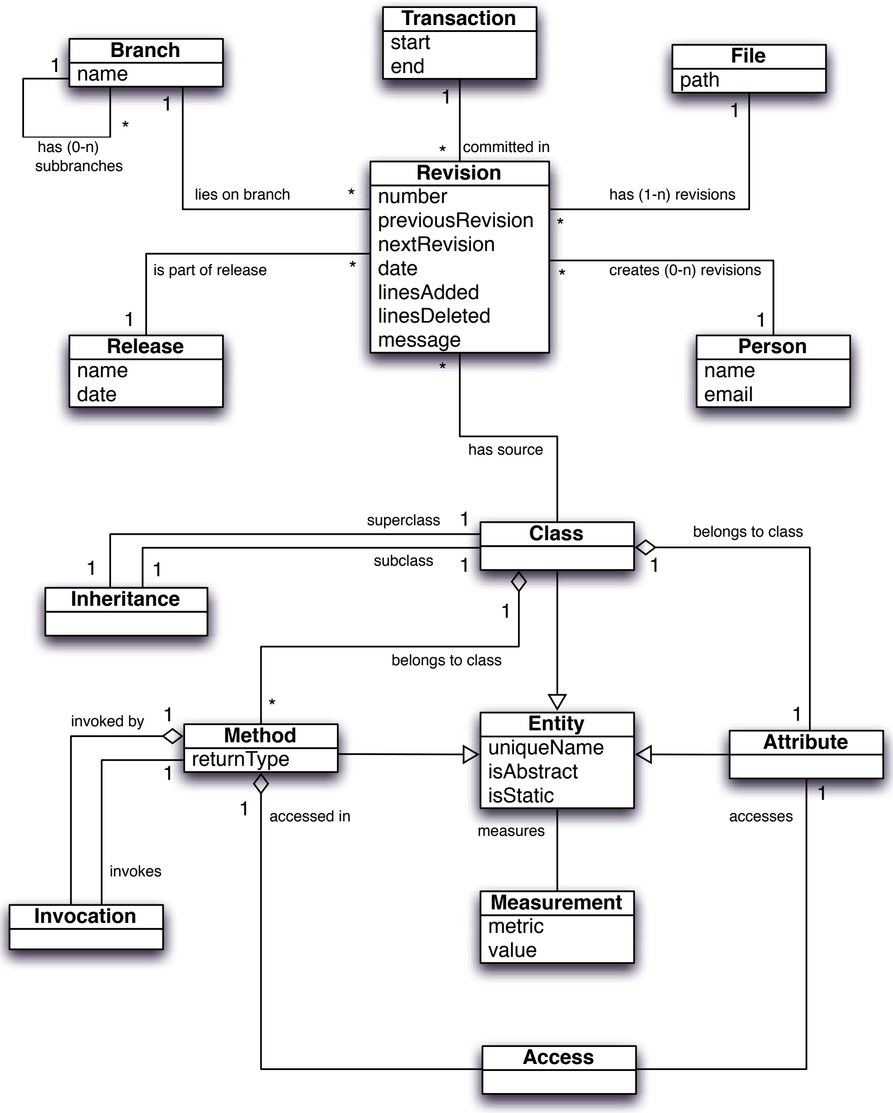

Fig. 2. Evolizer Data Schema for Source Code and Historical Analysis

information. The history model is generic, *i.e.,* it applies to a certain extent to centralized version control systems, as well as to distributed ones. In the following, we describe the most important parts of our schema. An overview of the simplified version of the schema is given in Figure 2; the full schema consists of approximately 40 distinct tables.

One of the core entities is *Revision.* A revision is a particular version of a file; a *Person*, that is to say a software developer, edits a file, and commits the modifications to the version control system. The latter tracks the date of the commit, the reason for the modification (*i.e.,* the commit message provided by the developer), as well as additional information such as the number of lines that were affected. A *Release* constitutes an important milestone in the life-cycle of a software system. It is often identified by a codename and contains a snapshot of the most recent revisions of all the files at the release data. New or experimental features, as well as bug fixes, are often developed on a *Branch.* Once the code is stable, it is merged back into the trunk.

To obtain a meaningful source code model, the system under analysis needs to be in a consistent state. This is generally guaranteed only for a release and therefore, for revisions that are part of a release, we can parse the source code and instantiate a reasonable model accordingly. A revision then corresponds to one top-level *Class* in Java, C#, etc. Classes have a set of members, *i.e.,* they contain *Attributes* and *Methods.* Classes, attributes, and methods are generalized into *Entities.*

Relationships between source code entities, such as *Invocations* between methods, *Accesses* from methods to attributes, and *Inheritance* between classes, are also made explicit by representing them as an association class or link table. While they are hard to distinguish from real entities, this is the only means that the relational model provides when we want to explicitly query for relationships.

Entities can be measured. Such a *Measurement* is specified by a metric, for example 'number of lines of code' for a class or method, or 'number of accesses' for an attribute, and the value that has been measured.

4.2 Ontologies of Software Analysis within SOFAS

To describe the data produced and consumed by SOFAS, we developed a family of Software Evolution ONtologies (SEON). They describe different aspects of software and its evolution, such as version control, issue tracking, static source code structure, change coupling, software design metrics, and so on. SEON is organized as several ontology pyramids. For each of the major subdomains, we have developed higher level ontologies defining their common concepts. For system-specific or language-dependent concepts we developed some concrete low-level ontologies. For instance, there is a high-level version control ontology and several low-level ontologies for concrete version control systems, such as CVS, SVN, and Git, that extend the high-level version control ontology. In this paper, we limit the discussion to the main terms of the source code ontology and the version control ontology. The source code ontology models the static source code structures based on the FAMIX meta model. It is therefore similar to the EVOLIZER data

Table 1. Source Code Ontology Overview

Class: Class	**Class: Method**
→ declaresMethod : Method	→ accessesField : Field
→ declaresField : Field	→ hasParameter : Parameter
→ isReturnTypeOf : Method	→ invokesMethod : Method
→ isSubclassOf : Class	→ hasReturnClass : Class
→ isSuperclassOf : Class	→ isInvokedByMethod : Method
→ hasName : xsd:string	→ isMethodOf : Class
Class: Field	→ hasName : xsd:string
→ isDeclaredFieldOf : Class	**Class: Parameter**
→ isAccessedByMethod : Method	→ isParameterOf : Method
→ hasName : xsd:string	→ hasName : xsd:string

Table 2. Version Control Ontology Overview

Class: Version	**Class: ChangeSet**
→ hasID : xsd:string	→ hasCommitDate : xsd:date
→ follows : Version	**Class: Branch**
→ precedes : Version	→ hasTag : xsd:string
→ hasCreationDate : xsd:date	**Class: Release**
→ linesAdded : xsd:int	→ hasReleaseDate : xsd:string
→ linesDeleted : xsd:int	→ hasTag : xsd:string
→ hasMessage : xsd:string	

schema described in the previous section. Table 1 provides an overview of the main classes and properties of the SEON source code ontology. The full ontology covers many more concepts such as *interfaces, local variables,* and *exceptions.*

The version control ontology models the structure of version control systems and is based on the data model described in [11]. Table 2 provides an overview of the main classes and properties of the SEON version control ontology.

4.3 OntoAccess as a Bridge to the New Town of Software Analysis

OntoAccess bridges the conceptual gap between the RDB-based Evolizer and the Semantic Web-enabled Sofas. It introduces a RDB-to-RDF mapping and provides on-the-fly translation of RDF-based read and write requests to the Evolizer RHDB. Table 3 and Table 4 contain an overview of the mapping in a schematic representation. Again, we focus in the presentation of the mapping on the parts of the Evolizer RHDB that are relevant to this case study. The mapping uses the following namespace declarations. `ver` for the SEON version control ontology `http://evolizer.org/ontologies/seon/2010/03/versions.owl`, `top` for `http://evolizer.org/ontologies/seon/2010/03/top.owl`, `java` for the SEON source code ontology `http://evolizer.org/ontologies/seon/2009/06/java.owl`, and `foaf` for `http://xmlns.com/foaf/0.1/`. Table 3 lists the mapping of tables from Figure 2 that represent a domain concept and their attributes. The table consists of four columns. The first names the table as in Figure 2 and the

Table 3. Mapping Overview Part I

table	**→ class**	**attribute**	**→ property**
Revision	→ `ver:Version`	number	→ `ver:hasID`
		previousRevision	→ `ver:follows`
		nextRevision	→ `ver:precedes`
		date	→ `ver:hasCreationDate`
		linesAdded	→ `ver:linesAdded`
		linesDeleted	→ `ver:linesDeleted`
		message	→ `ver:hasMessage`
Transaction	→ `ver:ChangeSet`	start	→ -
		end	→ `ver:hasCommitDate`
Branch	→ `ver:Branch`	name	→ `ver:hasTag`
File	→ `top:File`	path	→ `top:filePath`
Release	→ `ver:Release`	name	→ `ver:hasTag`
		date	→ `ver:hasReleaseDate`
Person	→ `foaf:Person`	name	→ `foaf:name`
		email	→ `foaf:mbox`
Entity	→ –	isAbstract	→ `java:isAbstract`
		isStatic	→ `java:isStatic`
Class	→ `java:Class`		
Method	→ `java:Method`	returnType	→ `java:hasReturnType`
Attribute	→ `java:Field`		
Measurement	→ `met:SoftwareMetric`	metric	→ `met:hasName`
		value	→ `met:hasValue`

second the class in the ontology it is mapped to. Column 3 contains the attributes of the respective table and their mapping to properties depicted in Column 4. A dash in the Columns 2 or 4 means that there is no mapping. The table *Entity* is not mapped to a class in the ontology but its attributes are mapped to properties. *Entity* is just a super type of several of the other concepts and only those (sub-)concepts are represented in the ontology (*cf.* Section 4.4).

Table 4 lists the mapping of link tables that represent M:N relationships in RDBs. As RDF provides different means to represent M:N relationships, such helper constructs are not needed and those tables are mapped to ontology properties instead. The table consists of three columns. The first names the link tables that are represented in Figure 2 as connecting lines between two concepts or as explicit concepts themselves. In the first case, the name is composed of the two participating table names separated by an underscore. Column 2 lists the property that each link table is mapped to. Column 3 lists the corresponding inverse property. For instance, the relationship from *Release* to *Revision* is mapped to the property `ver:comprises` and the inverse relationship from *Revision* to *Release* is mapped to the property `ver:appearsIn`.

4.4 Discussion

In our case study, we showed how OntoAccess has been successfully deployed to *make a bridge to the new town.* It provides a gradual migration path from

Table 4. Mapping Overview Part II

link table	**→ property**	**: inverse property**
Release_ Revision	→ `ver:comprises`	: `ver:appearsIn`
Branch_ Revision	→ `ver:comprises`	: `ver:isOn`
Transaction_ Revision	→ `ver:comprises`	: `ver:commitedIn`
File_ Revision	→ `ver:hasVersion`	: `ver:belongsTo`
Person_ Revision	→ -	: `ver:committedBy`
Class_ Revision	→ `ver:hasSource`	: -
Method_ Class	→ `java:isDeclaredMethodOf`	: `java:declaresMethod`
Attribute_ Class	→ `java:isDeclaredFieldOf`	: `java:declaresField`
Measurement_ Entity	→ `met:isMetricOf`	: `met:hasMetric`
Inheritance	→ `java:hasSubClass`	: `java:hasSuperClass`
Invocation	→ `java:invokesMethod`	: `java:isInvokedByMethod`
Access	→ `java:accessField`	: `java:isAccessedByMethod`

a legacy system such as EVOLIZER, to a new platform, in our example SOFAS. We demonstrated how ONTOACCESS bridges the conceptual gap between the relational data model of EVOLIZER and the RDF-based SOFAS. We further motivated that existing, read-only RDB-to-RDF mapping approaches are unsuitable for this application scenario as they limit RDF-based data access to read-only queries. During this case study, we faced several challenges w.r.t. to the mapping in ONTOACCESS. In the following, we report on two major ones and the solutions we developed to overcome them.

The first challenge is related to the representation of concept inheritance in relational database systems. Inheritance is a central concept in the object-oriented methodology and is therefore commonly used in object-oriented systems, including EVOLIZER. Relational, unlike object-relational or object-oriented databases, do not directly support inheritance. However, there exist three principal strategies to implement inheritance in relational database schemata (*cf.* [14]): *table-per-hierarchy* represents all classes of the inheritance hierarchy in a single table. This table contains columns for the attributes of all classes and a special column, called discriminator, that stores the type (*i.e.,* class) for each instance. *Table-per-concrete-class* represents each class in its own table. Each of those tables contains columns for the attributes of the class and all super-classes up to the root of the inheritance hierarchy. As a result, attributes of a common super-class are duplicated in all of its sub-classes. The third strategy, called *table-per-subclass*, also represents each class in its own table. In contrast to the *table-per-concrete-class* strategy, the attributes of the super-class(es) are not duplicated as columns in the sub-classes. Instead, a shared primary key is used to connect the tables representing classes in the inheritance hierarchy.

EVOLIZER uses different strategies for different inheritance hierarchies, for example the *table-per-hierarchy* strategy to implement inheritance for the *Entity* concept and its subconcepts. For the sake of this case study, we had to add explicit support for mapping inheritance hierarchies to ONTOACCESS. The *table-per-concrete-class* strategy was mappable out-of-the-box since it defines

Listing 1.2. Extended R3M Mapping Examples

```
a)  ex:method  a  r3m:TableMap;
      r3m:hasTableName        "Entity";
      r3m:mapsToClass         java:Method;
      r3m:hasDiscriminator    ex:method_type;
      r3m:uriPattern          "http://.../method_%%id%%";
      r3m:hasAttribute        ex:method_type, ....
    ex:method_type  a  r3m:AttributeMap;
      r3m:hasAttributeName    "ctype";
      r3m:hasValue            "Method".

b)  ex:Method  a  r3m:TableMap;
      r3m:hasTableName      "Method";
      r3m:mapsToClass       java:Method;
      r3m:hasParentTable    ex:entity;
      r3m:uriPattern        "http://.../method_%%id%%";
      r3m:hasAttribute      ex:method_returnType.
    ex:entity  a  r3m:TableMap;
      r3m:hasTableName      "Entity";
      r3m:hasAttribute      ex:entity_uniqueName, ....
```

one table per class and the tables are independent from each other. Mapping the other two strategies required support for features such as discriminator columns and relating tables in a parent-child relationship. We addressed this limitation by adding explicit mapping constructs to the OntoAccess mapping language. First, discriminator columns were added to provide support for the *table-per-hierarchy* strategy. Since support for mapping a subset of the columns in a table already exists, it is possible to provide multiple mappings for tables that represent all classes within an inheritance hierarchy (one mapping for each class). Each mapping only contains the respective subset of the columns and a description of the discriminator column with its name and value. Listing 1.2a) depicts a concrete mapping example that is using a discriminator column. We also added a mapping construct for relating two tables to each other in a parent-child relationship to provide support for the *table-per-subclass* strategy. The mapping of a table can reference another table as its parent table. This enables OntoAccess to detect that a concept from the application domain is split among multiple tables in the database schema. As a result, the involved tables can automatically be joined (on the primary key). Listing 1.2b) depicts a concrete mapping example that is using a parent table reference. These two extensions to R3M enable support for mapping the relational representations of concept inheritance with all three strategies.

The second challenge is related to defining the RDB-to-RDF mappings. Mappings in OntoAccess are encoded in RDF which makes them well-suited for automatic processing by machines but hinders the accessibility for human users. Manually defining such mappings is a time-consuming and error-prone task, consisting of mostly repetitive steps. Therefore, tool support for defining mapping is indispensable in more complex application scenarios where the number of database tables and columns is of significance. We built a tool [6] to ease the definition of OntoAccess mappings. It semi-automatically generates a mapping from a RDB schema in two steps. First, it automatically generates a basic mapping, based on information extracted from the schema catalog of the database system. Terms of the target ontology are also generated in this step, based on table and column names in the database schema. Next, the tool displays a graphical editor for refining the mapping. This step is mainly concerned with replacing the generated terms with actual terms from the target ontology. The tool further provides validation of existing mappings to catch errors from manual editing. The tool is implemented as a plug-in for the ontology editor Protégé[4] to enable quick access to the definition of the target ontology.

5 Related Work

D2R Server [4] is an approach for publishing existing relational databases on the Semantic Web. Based on mappings expressed in the D2RQ [5] mapping language, it enables browsing the relational data as RDF via dereferenceable URIs (*i.e.,* as Linked Data). Further, support for the SPARQL query language is provided. D2R is limited to read-only data access, updating RDF data is not supported.

The Virtuoso Universal Server features RDF Views [9] to expose relational data on the Semantic Web. A declarative Meta Schema Language is used for defining the mapping of SQL data to RDF vocabularies. This enables the use of SPARQL as an alternative query language for the relational data. Likewise, Virtuoso implements a Linked Data interface to these views. RDF Views are limited to read-only queries.

R_2O [1] is an extensible and fully declarative language to describe mappings between relational database schemata and ontologies. R_2O is aimed at situations where the similarity between the ontology and the database model is low. It has been conceived to be expressive enough to cope with complex mapping cases where one model is richer, more generic/specific, or better structured than the other. This high expressiveness renders R_2O mappings read-only.

The W3C has recognized the importance of mapping relational data to the Semantic Web by starting the RDB2RDF Incubator Group[5] (XG) to investigate the need for standardization. The XG recommended [22] that the W3C initiates a working group (WG) to define a vendor-independent RDB-to-RDF mapping language. The RDB2RDF WG[6] started its work on R2RML [7] in late 2009. According to their charter [18], the requirements for updating relational data are out of scope and are therefore not addressed by the WG. It was shown in [15] that adding write support to the R2RML approach is impractical.

[4] `http://protege.stanford.edu/`

[5] `http://www.w3.org/2005/Incubator/rdb2rdf/`

[6] `http://www.w3.org/2001/sw/rdb2rdf/`

6 Conclusions

In theory the Semantic Web provides a common framework that greatly facilitates data sharing and reuse across application, enterprise, and community boundaries. In practice its wide adoption is still hampered by the fact that many organizations have locked away their data in relational databases. Business-critical legacy applications rely on these databases to sustain daily operations and newly developed systems often need to run in tandem with their predecessors until the latter can be gradually phased out. Both, the legacy systems, as well as their successors, usually need to operate cooperatively on existing data. This includes reads and updates. A complete paradigm shift in data representation is therefore often extremely difficult and costly to achieve.

In this paper, we presented OntoAccess, a RDB-to-RDF mediation platform that enables RDF-based read and write access to relational databases. It greatly facilitates the transition from legacy systems to Semantic Web-enabled applications in practice by providing a semantic layer on top of existing relational databases. Semantic Web query and update requests are translated on-the-fly to SQL for execution in the database system. OntoAccess therefore eliminates the need for mirroring and synchronizing relational data with its RDF representation and, in addition, allows one to further exploit the advantages of the well-established database technology, such as query performance, scalability, transaction support, and security.

In our case study, we have described how we successfully deployed OntoAccess to provide a gradual migration path between two of our own large-scale software systems, namely the legacy application Evolizer and its successor, the Sofas platform. We identified challenges when it comes to mapping inheritance hierarchies with OntoAccess and we have extended the latter accordingly to support different inheritance mapping strategies. Further, we established tooling to semi-automate the process of extracting mappings from a relational database schema to an ontology.

In summary, judging from the experiences made in our case study, we are confident that OntoAccess is a valuable tool that will foster the acceptance of Semantic Web technology in practice.

References

1. Barrasa, J., Corcho, O., Gómez-Pérez, A.: R2O, an Extensible and Semantically Based Database-to-Ontology Mapping Language. In: Proc. Workshop on Sem. Web and Databases (August 2004)
2. Berners-Lee, T.: Linked Data (2009), `http://www.w3.org/DesignIssues/LinkedData.html` (last visited June 2011)
3. Berners-Lee, T.: Relational Databases on the Semantic Web (2009), `http://www.w3.org/DesignIssues/RDB-RDF.html` (last visited June 2011)
4. Bizer, C., Cyganiak, R.: D2R Server – Publishing Releational Databases on the Semantic Web. In: Proc. Int'l Sem. Web Conf. (November 2006)
5. Bizer, C., Seaborne, A.: D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs. In: Proc. Int'l Sem. Web Conf. (November 2004)

6. Brügger, N.: RDB-RDF Mapping Generation from Relational Database Schemata. Master's thesis, University of Zurich (December 2009)
7. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. W3C Working Draft (October 2010), `http://www.w3.org/TR/2010/WD-r2rml-20101028/`
8. Demeyer, S., Ducasse, S., Nierstrasz, O.: Object Oriented Reengineering Patterns. Morgan Kaufmann Publishers Inc., San Francisco (2002)
9. Erling, O., Mikhailov, I.: RDF Support in the Virtuoso DBMS. In: Proc. of the SABRE Conf. on Social Sem. Web (September 2007)
10. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
11. Fischer, M., Pinzger, M., Gall, H.: Populating a Release History Database from Version Control and Bug Tracking Systems. In: Proc. Int'l Conf. Softw. Maintenance (September 2003)
12. Fürber, C., Hepp, M.: Using SPARQL and SPIN for Data Quality Management on the Semantic Web. In: Abramowicz, W., Tolksdorf, R. (eds.) BIS 2010. LNBIP, vol. 47, pp. 35–46. Springer, Heidelberg (2010)
13. Gall, H.C., Fluri, B., Pinzger, M.: Change Analysis with Evolizer and ChangeDistiller. IEEE Softw. (January/February 2009)
14. Garcia-Molina, H., Ullman, J.D., Widom, J.: Database Systems: The Complete Book. Prentice Hall Press (2008)
15. Garrote, A., Garcia, M.N.M.: RESTful Writable APIs for the Web of Linked Data Using Relational Storage Solutions. In: Proc. WWW 2011 Workshop on Linked Data on the Web (April 2011)
16. Ghezzi, G., Gall, H.C.: Towards Software Analysis as a Service. In: Proc. Int'l ERCIM Workshop on Softw. Evolution and Evolvability (September 2008)
17. Ghezzi, G., Gall, H.C.: SOFAS : A Lightweight Architecture for Software Analysis as a Service. In: Working IEEE/IFIP Conf. on Softw. Architecture (June 2011)
18. Halpin, H., Herman, I.: RDB2RDF Working Group Charter (2009), `http://www.w3.org/2009/08/rdb2rdf-charter` (last visited June 2011)
19. Hert, M.: Relational Databases as Semantic Web Endpoints. In: Proc. European Sem. Web Conf. (June 2009)
20. Hert, M., Reif, G., Gall, H.C.: Updating Relational Data via SPARQL/Update. In: EDBT Workshop Proc. (March 2010)
21. Hert, M., Reif, G., Gall, H.C.: A Comparison of RDB-to-RDF Mapping Languages. In: Proc. Int'l Conf. on Semantic Systems (2011)
22. Malhotra, A.: W3C RDB2RDF Incubator Group Report (January 2009), `http://www.w3.org/2005/Incubator/rdb2rdf/XGR-rdb2rdf-20090126/`
23. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (January 2008), `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`
24. Seaborne, A., Manjunath, G., Bizer, C., Breslin, J., Das, S., Davis, I., Harris, S., Idehen, K., Corby, O., Kjernsmo, K., Nowack, B.: SPARQL Update – A Language for Updating RDF Graphs. W3C Member Submission (July 2008), `http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/`