Unified Querying of Ontology Languages with the SIRUP Ontology Query API

Patrick Ziegler Christoph Sturm Klaus R. Dittrich

Database Technology Research Group, Department of Informatics, University of Zurich
Winterthurerstrasse 190, CH-8057 Zürich, Switzerland
pziegler sturm dittrich @ifi.unizh.ch

Abstract:
Ontology languages to represent ontologies exist in large numbers, and users who want to access or reuse ontologies can often be confronted with a language they do not know. Therefore, due to their great number, ontology languages are nowadays themselves a source of heterogeneity.

In this paper, we present the SIRUP Ontology Query API (SOQA) that was developed for the SIRUP approach to semantic data integration [ZD04b]. SOQA is an ontology language- and platform-independent API for query access to ontological metadata and data that can be represented in a variety of ontology languages. Based on SOQA, we provide SOQA-QL, an SQL-like query language that supports declarative queries against ontological metadata and data, and the SOQA Browser, a tool to graphically inspect all ontology information that can be accessed through SOQA.

1 Introduction

In current information systems, ontologies [UG96] are more and more widely used to explicitly represent the intended real-world semantics [OS99] of data and services. Ontologies provide a means to overcome heterogeneity by providing explicit, formal descriptions of concepts and their relationships that exist in a certain universe of discourse, together with a shared vocabulary to refer to these concepts. Based on agreed ontological domain semantics, the danger of semantic heterogeneity can be reduced. Ontologies can, for instance, be applied in the area of data integration for data content explication to ensure semantic interoperability between data sources. Moreover, semantic descriptions of services can be used to find tailored services for certain requirements.

A large number of ontology languages is available to specify ontologies. Besides traditional ontology languages, such as Ontolingua [FFR97], PowerLoom [MCM03], OCML [Mot99], or F-Logic [KLW95], there is a notable number of ontology languages for the Semantic Web, such as SHOE [LSRH97], OIL [FHH+01], DAML[1], and OWL [BHH+04]. That is, ontology languages are nowadays themselves a source of heterogeneity. As ontologies can be specified in a manifold of ontology languages, users looking for suitable ontologies can often be confronted with ontologies that are defined in a language they do

[1] http://www.daml.org

not know. In order to make use of these ontologies, users either have to learn the particular ontology language or to find and employ suitable tools to access the desired ontology. Heterogeneity caused by the use of different ontology languages can therefore be a major obstacle in ontology access.

Besides this, building ontologies is a demanding and time-consuming task because not only a conceptual schema for a single application, but an application-independent and reusable domain model has to be developed. Thus, key concepts and relationships have to be identified in the domain of interest first. Then, precise and unambiguous definitions for these concepts and relationships must be produced and suitable terms have to be found to refer to these concepts, before, at last, the ontology can be coded in a particular ontology language [UG96]. Especially in cases where large all-embracing ontologies are built, such as CYC [Len95], the development phase can be a substantial investment — for example, more than a person-century has been invested in the development of CYC. Therefore, existing ontologies should be reused so that advantage can be taken of the efforts spent during the ontology development phase [UG96]. However, heterogeneity caused by the use of different ontology languages can be a considerable impediment to realize the benefits from ontology development efforts.

For instance, assume that a developer of an information system is looking for an ontology concerning persons from the university domain. Therefore, he or she might use a search engine and find (1) a PowerLoom ontology file `university.ploom`, (2) the Aktors portal ontology[2] that is represented in OWL, and (3) the DAML university ontology[3] from the University of Maryland. In order to find out whether the individual ontologies (or a combination of them) fit the developer's needs, he or she either has to know PowerLoom, OWL, and DAML. Alternatively, our developer might try to find and install one or more browsers capable of displaying the contents of the three ontologies in graphical form. Besides this, he or she could attempt to find and learn a declarative query language that is able to act as an interface to the three ontology languages at hand. In either way, accessing and reusing ontologies that are represented in different ontology languages can be time-consuming and labor-intensive.

In this paper, we present the SIRUP Ontology Query API (SOQA)[4] which is an ontology language- and platform-independent API for query access to ontological metadata and data that can be represented in a variety of ontology languages. Our main goal is to define a set of access primitives suitable to retrieve ontological content from a broad range of ontology languages. In contrast to existing single ontology-language APIs, such as the WonderWeb OWL API [BVL03] that provides access to OWL ontologies, our focus is on providing a general-purpose ontology language query API capable of encompassing a multitude of ontology languages. In addition to this, we aim at improving ease of use when dealing with ontologies. That is, we aim at supporting declarative queries against ontological data and metadata. In addition to this, our goal is to enable users to graphically browse ontologies that are formulated in different ontology languages.

[2] http://www.aktors.org/ontology/portal

[3] http://www.cs.umd.edu/projects/plus/DAML/onts/univ1.0.daml

[4] The term "soqa" originally refers to a traditional Fijian thatch roof made of leaves. In either way, soqa / SQOA denotes a "cover" that represents a unified view of top of underlying components.

As its main application, SOQA is employed for ontology access used for data content explication in the SIRUP (Semantic Integration Reflecting User-specific semantic Perspectives) approach [ZD04b] to semantic data integration. SIRUP addresses the problem of providing user-specific integrated data that perfectly fits a particular user's information needs, emphasizing his or her individual way to perceive a domain of interest [ZD04a]. However, SOQA is intended and designed to be a general-purpose ontology query API that can be used independently of SIRUP.

This paper is structured as follows: The following Sect. 2 gives a brief overview of the SIRUP approach to semantic data integration in which SOQA is mainly employed. Sect. 3 presents the API of SOQA and illustrates how content from different ontology languages can be accessed. Sect. 4 describes the SOQA query language SOQA-QL and Sect. 5 deals with the graphical browser for SOQA. In Sect. 6, related work is discussed. Sect. 7 concludes the paper.

2 A Quick Overview of SIRUP

In the area of data integration, there is a remarkable history of research projects. The spectrum ranges from early multidatabase systems (e.g., Multibase [LR82]) over mediator systems (e.g., Garlic [CHS+95]) to ontology-based integration approaches (e.g., OBSERVER [MKSI96]). These approaches have in common that autonomy of data sources to be integrated is considered to be of paramount importance. Besides this autonomy of data sources, there is the often neglected autonomy and sovereignty of data receivers, i.e., human users and applications [GMS94]. Data receivers are autonomous in the sense that they typically have different information needs and vary in the ways they perceive their particular domain of interest. Sovereignty of data receivers refers to the fact that using integrated data must be *non-intrusive* [SEGM+90]; i.e., users should not be forced to adapt to any standard concerning structure and semantics of data they desire. Therefore, to take a "one integrated schema fits all" approach is definitely not a satisfactory solution.

In the SIRUP (Semantic Integration Reflecting User-specific semantic Perspectives) approach [ZD04b], we address the problem of how user-specific ways to perceive a particular application domain can be taken into account in the process of semantically integrating data from heterogeneous data sources. The general goal of SIRUP is to provide means that allow data from heterogeneous sources to be integrated in a way that it perfectly fits a particular user's information needs, emphasizing his or her individual way to perceive an application domain of interest. Additionally, we abstract the user from low-level heterogeneity and technical details of underlying data sources. In contrast to traditional ex-post definition of views on top of already existing integrated global schemas, we advocate a method called *ex-ante view definition* which allows that only data items are integrated which are semantically related according to the user's individual perception of the particular application domain. In the end, we provide well-structured user-specific schemas with extensive metadata and explicit, queryable semantics for integrated data from selected sources.

SIRUP generally concentrates on querying, not on manipulation of integrated data; for integration, alphanumeric data from a broad range of data sources (i.e., database systems, web services, application interfaces, file systems, and the web) is considered. In SIRUP, all data that is provided by data sources for integration is linked to so-called IConcepts (see Fig. 1). An IConcept (short for *Intermediate Concept*[5]) is a basic conceptual building block that acts as a linking element between data providers and data users interested in data for their information needs. Each IConcept has a queryable link to at least one concept of an ontology to explicitly define the semantics of the real-world concept it represents (e.g., "professor" or "student"). Data sources can provide attributes for an ontological concept represented by a particular IConcept (e.g., "first name" or "birthday" for "professor"). Thus, data sources can declare what attribute data they are capable and willing to provide concerning a given IConcept. For each of these attributes, additional structural metadata (data type, measurement unit, precision, constraints, etc.) is provided.

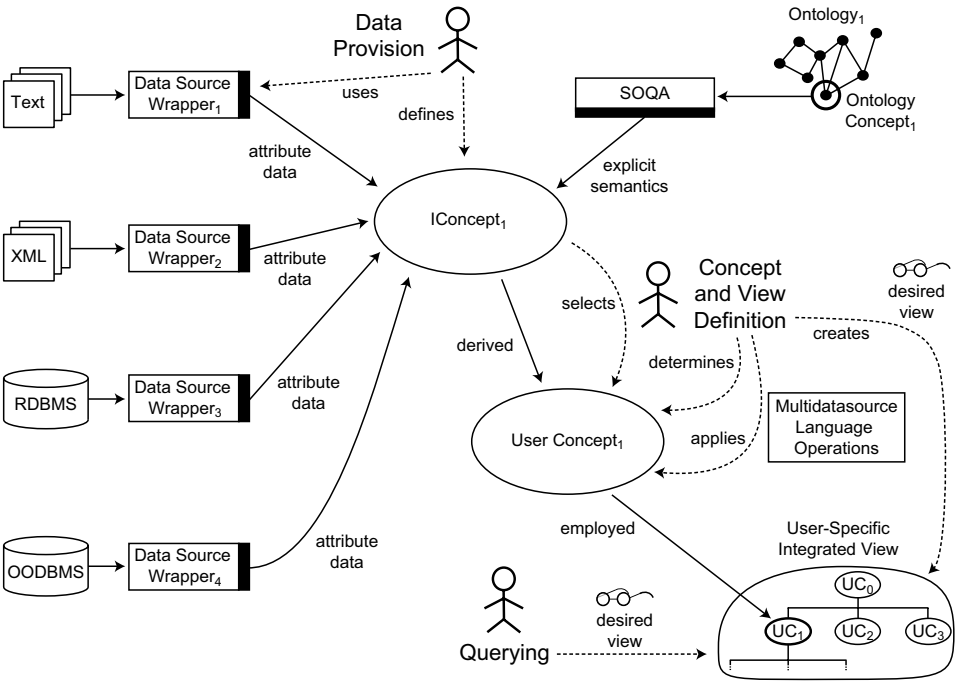

Figure 1: Overview of the SIRUP Approach to Data Integration

Starting from a centralized ontology index, SIRUP users can browse for IConcepts relevant for their information needs and select, combine and restructure IConcepts to build tailored User Concepts and User Concept hierarchies. A User Concept is a user-specific concept

[5]"Intermediate", because an IConcept is (1) not a fully-fledged user-specific concept (see the next paragraph) but just a building block to construct user-specific concepts and (2) it has an intermediate position between data sources/data providers and data users/query issuers.

that is built by selecting and combining user-specific copies of IConcepts (e.g., by defining "student of informatics" based on "student"). Using selection, projection, join, cartesian product, and set operators (union, set difference, intersection), users can declaratively define User Concepts based on the selected IConcept copies and already existing other User Concepts. Hence, IConcept copies constitute the base nodes on top of which users can incrementally build in a bottom-up manner user-specific concepts and concept hierarchies that together form the desired integrated view. In query processing, relevant attribute data can be retrieved from the data sources by the IConcept copies which user-specific concept hierarchies are based on. During concept definition, ontology links inherited from underlying IConcept copies as well as attribute metadata are automatically maintained for User Concepts so that explicitly defined queryable semantics and up-to-date structural metadata are still available. By this metadata, each User Concept possesses a highly structured, explicit schema that describes the attributes that are available.

In the SIRUP approach, a declarative multidatasource language is provided for data provision as well as for specifying User Concepts and integrated views. This language supports querying of explicit semantics and metadata assigned to User Concepts and IConcepts. Additionally, data queries against integrated data from user-specific integrated views are supported. For data providers, the SIRUP multidatasource language offers the means to perform integration of semantically equivalent IConcept attributes that originate from different data sources. This language also supports IConcept definition and linking of attributes from structured, semi-structured, and unstructured data sources. For metadata and data access from these data sources, we employ data source wrappers.

Ontologies are used in the SIRUP approach for making semantics of data explicit to human users. In order to not constrain the set of applicable ontologies, we support different ontology languages. Thus, ontologies that are specified in various ontology languages can be used in SIRUP for data content explication. We employ ontology wrappers to cope with heterogeneity caused by differences in these ontology languages. Concerning their content, the set of available ontologies is generally open and extensible so that general foundational ontologies as well as specialized domain-specific ontologies can be used for accurate data content explication. Using the SIRUP multidatasource language, users can be provided with explicit, queryable semantics regardless of the ontology language used to represent the intended real-world semantics of data. In SIRUP, the basis for access to explicit data semantics that are specified in various ontology languages is provided by the SIRUP Ontology Query API (SOQA) that is described in detail in the next section.

3 The SIRUP Ontology Query API

In general, ontology languages are designed for a particular purpose and, therefore, they differ in their syntax and semantics. That is, differences in the syntactical notations and their meaning as well as in the modeling capabilities can occur. As a starting point to our work, we conducted a survey on widely-used ontology languages to determine which modeling capabilities are commonly shared among them. On this basis, we defined the *SOQA Ontology Meta Model* that represents modeling capabilities that are typically sup-

ported by ontology languages to describe ontologies and their components. In the sense of the SOQA Ontology Meta Model, an ontology consists of the following components:

- Metadata to describe the ontology itself. This includes name, author, date of last modification, (header) documentation, version, copyright, and URI (Uniform Resource Identifier) of the ontology as well as the name of the ontology language the ontology is specified in. In addition, each ontology has extensions of all concepts, attributes, methods, relationships, and instances that appear in it.

- Concepts which are entity types that occur in the particular ontology's universe of discourse — that is, concepts are descriptions of a group of individuals that share common characteristics. In the SOQA Ontology Meta Model, each concept is characterized by a name, documentation, and a definition that includes constraints;[6] additionally, it can be described by attributes, methods, and relationships. Further, each concept can have direct and indirect super- and subconcepts, equivalent and antonym concepts, and coordinate concepts (that are situated on the same hierarchy level as the concept itself). For example, ontology language constructs like `<owl:Class ...>` from OWL and `(defconcept ...)` from PowerLoom are represented as concepts in the SOQA Ontology Meta Model.

- Attributes that represent properties of concepts. Each attribute has a name, documentation, data type, definition, and the name of the concept it is specified in.

- Methods which are functions that transform zero or more input parameters into an output value. Each method is described by a name, documentation, definition, its parameters, return type, and the name of the concept the method is declared for.

- Relationships that can be established between concepts, for instance, to build taxonomies or compositions. Similar to the other ontology components, a name, documentation, and definition can be accessed for each relationship. In addition, the arity of relationship, i.e., the number of concepts it relates, as well as the names of these related concepts are available.

- Instances of the available concepts that together form the extension of the particular concept. Each instance has a name and provides concrete incarnations for the attribute values and relationships that are specified in its concept definition. Furthermore, the name of the concept the instance belongs to is available.

A UML class diagram of the SOQA Ontology Meta Model is shown in Fig. 2. Note that the SOQA Ontology Meta Model is deliberately designed to not only represent the least common denominator of modeling capabilities of widely-used ontology languages. In deciding whether or not to incorporate additional functionality that is not supported by some ontology languages, we opted for including these additional modeling capabilities (e.g., information on methods, antonym concepts, ontology authors, etc.) provided that they are useful for users of the SOQA API and available in important ontology languages.

[6] In SOQA, axioms/constraints are subsumed by the definitions of the particular meta model elements.

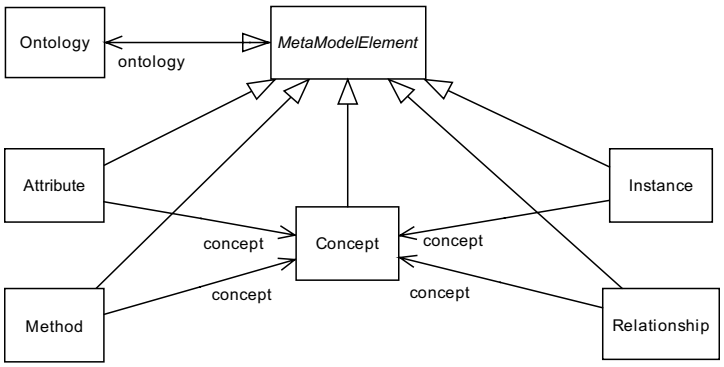

Figure 2: Overview of the SOQA Ontology Meta Model as a UML Class Diagram

Based on the SOQA Ontology Meta Model, the functionality of the SOQA API was designed. The SIRUP Ontology Query API (SOQA) is an ontology language- and platform-independent API for query access to ontological metadata and data that can be represented in a multitude of ontology languages. That is, SOQA provides read access to ontologies through a uniform API that is independent of the underlying ontology language and hardware/software platform. Thus, accessing and reusing general foundational ontologies as well as specialized domain-specific ontologies can be facilitated. With SOQA, users and applications can be provided with unified access to metadata and data of ontologies according the SOQA Ontology Meta Model. Besides, data of concept instances can be retrieved through SOQA. Note that despite the fact that SOQA is mainly employed for ontology access used for data content explication in the SIRUP integration approach [ZD04b], SOQA is intended and designed to be a general-purpose ontology query API that can be used independently of SIRUP. SOQA and all its components are fully implemented in Java 1.5.

From an architectural perspective, the SOQA API reflects the Facade [GHJV95] design pattern: SOQA provides a unified interface to a subsystem which is in charge of retrieving information from ontologies that are specified in different ontology languages. SOQA as a Facade shields external clients from the internal SOQA components and represents a single point for unified ontology access (see Fig. 3). Examples for external clients of the API provided by the SOQA Facade are :

- The query language SOQA-QL (see Sect. 4), which supports declarative queries against data and metadata of ontologies that are accessed through SOQA;
- The SOQA browser (see Sect. 5) that enables users to graphically inspect the contents of ontologies independent of the ontology language they are specified in;
- (Third-party) Java applications that use SOQA as a single point of access to information that is specified in different ontology languages. Possible application areas are the Semantic Web, knowledge management, e-business, and enterprise application integration. For instance, SOQA can be regarded as part of a Semantic Web

infrastructure; that is, SOQA can be used to provide uniform access, on behalf of Semantic Web tools, to existing ontologies that are defined in traditional ontology languages. Another application example is SIRUP, which uses SOQA as a standard ontology interface for data content explication.

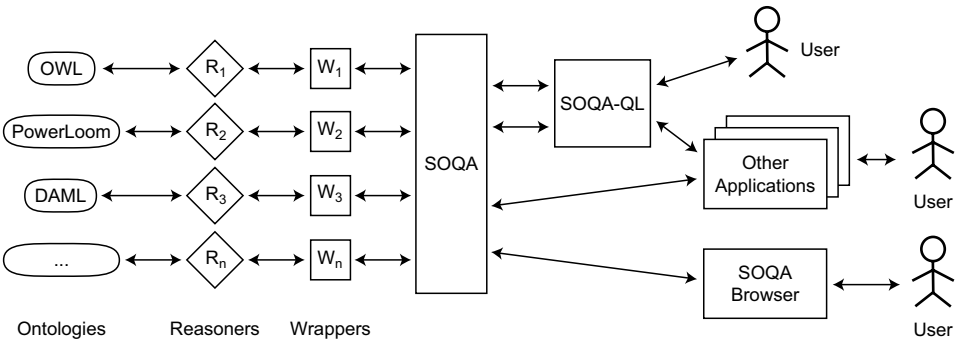

Figure 3: Overview of the SOQA Software Architecture

Behind the SOQA Facade, ontology wrappers are used as an interface to reasoners that are specific to a particular ontology language (see Fig. 3). Each wrapper is a coupling module that is capable of accessing a single reasoning engine tailored for a respective ontology language. Thus, wrappers provide a uniform interface to various ontological reasoners on behalf of SOQA. By (low-level) services provided by the reasoners, desired information from the underlying ontology is retrieved and returned to the wrappers. A wrapper then produces a uniform result to the SOQA method call by transforming the results returned by the reasoner into Java objects that represent the ontology components defined in the SOQA Ontology Meta Model. That is, for each method that is called on SOQA, a collection of Java objects is returned that reflect the SOQA Ontology Meta Model. After this transformation, the Java objects are returned through SOQA to the client that called the SOQA method. Note that these returned Java objects are *fully-functional* logical representations of elements of the particular ontology that was accessed. That is, external Java applications that use SOQA can post-process results returned by SOQA and directly access additional data and metadata through object-specific methods (e.g., `getAttributes(...)` for instances of `Concept`, etc.).

SOQA does not supply its own reasoning engine(s) for the supported ontology languages. Instead, SOQA is capable of employing existing language-specific reasoners, such as the OWL reasoners Pellet[7] or RACER[8]. At startup, SOQA is initialized by specifying the URI of an ontology to be accessed together with a statement which ontology language the ontology is specified in. SOQA then instantiates a suitable wrapper and reasoner that subsequently handle information requests concerning the ontology. In addition to this, SOQA can be parameterized at startup to do caching of ontology information that is retrieved

[7] http://www.mindswap.org/2003/pellet/
[8] http://www.sts.tu-harburg.de/~r.f.moeller/racer/

through ontology wrappers. Hence, performance improvements in ontology information access can be gained.

In SOQA, ontology information requests are generally processed based on the principle of lazy (late) evaluation: Information from ontologies is not retrieved and converted into the SOQA Ontology Meta Model in advance, but only when it is actually needed. Thus, the Java objects reflecting the SOQA Ontology Meta Model, which are returned in response to a SOQA method call, do not contain references to, for example, all subconcepts of a particular concept, but have access methods that can be called to retrieve, if necessary, the subconcepts of the particular concept. Again, advantage can be taken from ontology information caching in SOQA whenever such an access method is repeatedly called.

Based on its Facade architecture, SOQA provides a set of methods that act as access primitives to retrieve ontological content from a broad range of ontology languages. Basically, SOQA provides access methods for all ontological components including all their information and subcomponents as defined in the SOQA Ontology Meta Model. The following four method signatures (S1) to (S4) illustrate how ontology information can be accessed through SOQA:

```
public Concept getConcept(String conceptName)                    (S1)

public Collection<Concept> getDirectSuperConcepts(               (S2)
       String conceptName)

public String getAttributeDocumentation(String                   (S3)
       conceptName, String attributeName)

public Collection<Instance> getInstances()                       (S4)
```

In the examples given before, method signature (S1) provides access to an ontological concept based on its name and (S2) enables SOQA clients to retrieve all direct superconcepts of a particular concept. (S3) is the signature of the method that returns the documentation of an attribute that belongs to a concept. Last, but not least, (S4) provides access to the extension of all instances that are defined in an ontology.

Beyond access to ontology information according to the SOQA Ontology Meta Model, SOQA provides a variety of helper methods for improved convenience — for example, `isMethod(String methodName)` to determine whether a certain method exists, `getRootConcept()` to access the root concept of the particular ontology, or the `getInstancesWithValue(String conceptName, String attributeName, Object value)` method to retrieve instances of a concept that have a certain value for a given attribute. All in all, SOQA comprises more than 70 Java methods for unified ontology access. In Table 1, a more detailed overview of the SOQA API methods is given.

Recall that in our university example from Sect. 1, a developer of an information system is looking for an ontology on persons from the university domain. In this scenario, SOQA can be used as follows: After having found, through a search engine, the three ontologies

API Method Category	Method Examples
Ontology Information	getAuthor(), getVersion(), getCopyright(), getOntologyURI(), getDocumentation(), getOntologyLanguageName()
Naming Information	getLocalName(String name), getNamespaces()
Ontology Element Access	getConcept(String conceptName), getRootConcept(), getRelationship(String conceptName, String relationshipName), getAttributesWithName(String attributeName)
Extension Access	getConcepts(), getAttributes(), getRelationships()
Concept Information	getAttributes(String conceptName), getInstances(String conceptName), getInheritedRelationships(String conceptName), getConceptDocumentation(String conceptName), getConceptDefinition(String conceptName)
Attribute Information	getAttributeDataType(String conceptName, String attributeName), getAttributeRange(String conceptName, String attributeName)
Method Information	getMethodParameters(String conceptName, String methodName), getMethodReturnType(String conceptName, String methodName)
Relationship Information	getRelationshipArity(String conceptName, String relationshipName)
Instance Information	getInstances(String conceptName), getInstancesWithValue(String conceptName, String attributeName, Object value)
Taxonomic Information	getDirectSuperConcepts(String conceptName), getSuperConcepts(String conceptName), getCoordinateConcepts(String conceptName)
Boolean Functions	isAttribute(String attributeName), isConcept(String conceptName), isSuperConcept(String conceptName, String superConceptName)

Table 1: Overview and Classification of Representative SOQA API Methods

represented in PowerLoom, OWL, and DAML respectively, the developer can employ a SOQA client application, such as the SOQA query language SOQA-QL, the SOQA browser, or a third-party application (e.g., a specialized ontology visualization tool) that are based on the SOQA ontology API. In order to retrieve ontology information, for example from the found DAML university ontology, the particular SOQA client application starts SOQA by specifying the URI of the ontology together with a statement that the ontology is represented in DAML. Behind the scenes, SQOA then initializes the necessary wrapper and reasoner instances. Subsequently, information from the ontology can be uniformly retrieved by the respective SOQA client application on behalf of the developer; he or she can then comfortably inspect the ontology and determine whether it fits the particular needs without having to cope with different ontology representations. Moreover, the developer can afterwards access the ontology from an application through SOQA if it is considered to be useful; thus, no additional DAML ontology access functionality has to be implemented in applications.

SOQA is designed to provide access to ontological content from a broad range of ontology languages. That is, in contrast to existing single ontology-language APIs, our focus is on providing a general-purpose ontology language query API capable of encompassing a multitude of ontology languages. The spectrum of supported ontology languages ranges from traditional languages, such as Ontolingua, PowerLoom, OCML, or F-Logic, to re-

cent XML and RDF/RDF-Schema based ontology languages for the Semantic Web, such as SHOE, OIL, DAML, and OWL. Up to now, we have implemented SOQA ontology wrappers for OWL, PowerLoom, DAML, and the lexical ontology WordNet [Mil95]. In the future, it is planned to implement additional wrappers for other ontology languages, such as OIL, SHOE, or F-Logic. Besides this, we intend to add support for Cyc/OpenCyc [9], which is a general ontology for commonsense knowledge, as well as for the OKBC protocol [CFF+98] to access information from knowledge representation systems.

4 The SOQA Query Language

The SOQA Query Language (SOQA-QL) is an SQL-like query language that supports declarative queries against ontology data and metadata. SOQA-QL is based on the ontology API provided by SOQA (see Fig. 3); that is, SOQA-QL enables users to query ontological content independently from the ontology languages particular ontologies are represented in. From a database perspective, SOQA-QL can be regarded as a language for queries against ontologies on a data-dictionary level; i.e., SOQA-QL retrieves descriptive ("schema") information of concepts and their relationships that exist in a given ontology. Besides, SOQA-QL can also be used for data level access to concrete values of concept instances. For the definition of SOQA-QL, we started with SQL and adapted and extended it so that all features provided by the SOQA API can directly be made available in declarative queries against ontologies. In general, a SOQA-QL query has the following form:

SELECT $proj_1, \ldots, proj_n$
FROM $metaModelElementSet_1, \ldots, metaModelElementSet_m$
WHERE $cond_1$ OP_1 $cond_2$ $\ldots$ OP_k $cond_{k+1}$;

where $proj_1, \ldots, proj_n$ is the projection expression and $metaModelElementSet_1, \ldots, metaModelElementSet_m$ is the specification of the element(s) of the SOQA Ontology Meta Model to be considered for the query; $cond_1$ OP_1 $cond_2$ $\ldots$ OP_k $cond_{k+1}$ is the condition expression and $\mathsf{OP}_1 \ldots \mathsf{OP}_k$ are the boolean operators AND and OR. The meaning of these expressions is as follows:

- Each projection expression is either an aggregate function (MIN, MAX, COUNT, AVG, SUM) or a list of attributes that are requested to be represented in the projection result. Note that "attribute" in the context of projections only refers to *metaclass attributes* that are generally defined for elements of the SOQA Ontology Meta Model (and not, for example, to *concrete attributes* of a particular ontological concept): Each element of the SOQA Ontology Meta Model (see Fig. 2) is a metaclass that has a predefined set of metaclass attributes — e.g., for the `Concept` metaclass, the attributes `name`, `documentation`, and `definition` are defined.[10] These

[9] http://www.opencyc.org/

[10] The available metaclass attributes for the other metaclasses of the SOQA Ontology Meta Model are: `name`, `documentation`, `definition`, `datatype`, and `conceptname` for the `Attribute` metaclass; `name`

metaclass attributes are independent of the attributes that instances of the particular metaclass have in a particular ontology (e.g., a concept `Student` can exist in a certain ontology and may have an (additional) attribute `emailAddress`). In order to retrieve information about concrete attributes, specialized data access functions as provided by SOQA may be used; in general, SOQA-QL provides a function for each method that is defined in the API of SOQA.[11] Thus, users of SOQA-QL have the means at their disposal to incorporate SOQA method calls into their declarative ontology queries.

- The specification of the element(s) of the SOQA Ontology Meta Model to be considered for the query may consist of (1) one or more concepts, (2) one or more of the extensions that are available for all concepts, attributes, methods, relationships, and instances, (3) a SOQA-QL subquery, or (4) the keyword `ontology` in case ontology metadata is requested.
- The condition expression consists of a set of attribute comparisons that can be connected with the boolean operators AND and OR; attribute comparisons may also include the logical NOT operator. In addition, subqueries ("IN") are supported.

For more detailed information on SOQA-QL, the grammar of SOQA-QL in Extended Backus-Naur Form (EBNF) is shown in Appendix A.

In the following, we illustrate the capabilities of SOQA-QL with some example queries. For instance, assume the developer in our university example from Sect. 1 wants to get more information about the concept `Student` in the DAML university ontology. He or she might then ask the following query to see whether some concept documentation is available for `Student`:

SELECT documentation FROM base1_0_daml:Student;[12] (Q1)

Then, our developer might be interested in knowing about the attributes of `Student`. This information can be queried by referring to the `getAttributes(conceptName)` method from SOQA with the attributes(conceptName) function in SOQA-QL:

SELECT name, documentation FROM attributes(base1_0_daml:Student); (Q2)

Furthermore, if information about the names of all other concepts except `Student` is desired, a query against the extension of all concepts in the DAML university ontology can be asked:

---

and `conceptname` for `Instance` and `name`, `documentation`, `definition`, and `conceptname` for the `Method` metaclass; for `Relationship`, `name`, `documentation`, `definition`, and `arity` are defined; last, but not least, `name`, `author`, `date`, `version`, `copyright`, (ontology) `languagename`, and `documentation` are defined for the `Ontology` metaclass of the SOQA Ontology Meta Model.

[11] The naming convention is as follows: The SOQA-QL function has the same name as the SOQA method except the leading `get`: For example, the `getConcept(...)` method from SOQA (see (S1) in Sect. 3) can be referenced with `concept(...)` in SOQA-QL.

[12] base1_0_daml is the namespace shortcut for http://www.cs.umd.edu/projects/plus/DAML/onts/base1.0.daml.

```
SELECT name FROM allconcepts WHERE name != 'base1_0_daml:Student';    (Q3)
```

To find out more about the ontology itself, queries against ontology metadata can be formulated in SOQA-QL. For example, the following query retrieves information about the author and documentation of the ontology together with the name of the ontology language the ontology is represented in:

```
SELECT author, documentation, languagename FROM ontology;    (Q4)
```

After having queried the DAML university ontology, the developer in our example may connect to the Aktors portal ontology that is specified in OWL and to the PowerLoom ontology file `university.ploom`, ask the same queries (Q1) to (Q3) against the `Student` concept of these ontologies, and compare the query results. Additionally, the developer could be interested in knowing about the direct superconcepts of `Student` in the portal ontology by asking this query:

```
SELECT * FROM directsuperconcepts(portal:Student) [13]    (Q5)
ORDER BY name ASC;
```

In addition to queries against ontology metadata, SOQA-QL also supports data level access to concrete values of concept instances. For instance, the developer in our example might be curious about all instances of subconcepts of `Student` whose name starts with "A" or "B". This can be returned by the following query:

```
SELECT name, value(portal:emailAddress)    (Q6)
FROM   instances(subconcepts(portal:Student)) WHERE name < 'C';
```

Note the value(portal:emailAddress) expression in (Q6): The value(...) function of SOQA-QL enables data level access to values of *concrete attributes* that are defined for a particular concept in its ontology (in addition to metaclass attributes that are generally defined for all elements of the SOQA Ontology Meta Model — see page 11). Metaclass attributes, on the other hand, can always be directly accessed in SOQA-QL queries since they are generally available regardless of the specifications contained in a particular ontology.

Last, but not least, assume that our developer wants to get more information about the attributes of the Aktors portal ontology — e.g., by querying the names of all concepts from which `Student` inherits one or more attributes. For information needs like this, SOQA-QL supports subqueries:

```
SELECT c.name FROM allconcepts AS c, allattributes AS a    (Q7)
WHERE c.name = a.conceptname AND
      a.name IN ( SELECT name
                  FROM   attributes(superconcepts(portal:Student)) );
```

[13] portal is the namespace shortcut for http://www.aktors.org/ontology/portal.

After SOQA-QL queries are formulated as shown in the examples above, they are sent for evaluation to the SOQA-QL query processor, which was developed with the Java parser generator JavaCC[14]. Using JavaCC, our SOQA-QL grammar specification (see Appendix A) can be converted into a set of Java classes that together form a top-down (recursive descent) parser. This parser is capable of generating parse trees of submitted queries that are formulated according to the SOQA-QL grammar. Parse trees are then taken as an input by the SOQA query processor that transforms each parse tree into an operator tree, resolves all variables and identifiers, and optionally optimizes the operator tree, before the nodes of the tree are finally evaluated to retrieve the results of the query. Then, the query results can be returned to the SOQA-QL user — that is, either in the form of a collection of instances of Java classes reflecting the SOQA Ontology Meta Model or, for instance in case of an aggregate function, as a primitive data type. As a graphical front-end for SOQA-QL query formulation and result display, we provide the SOQA Query Shell (see Fig. 4).

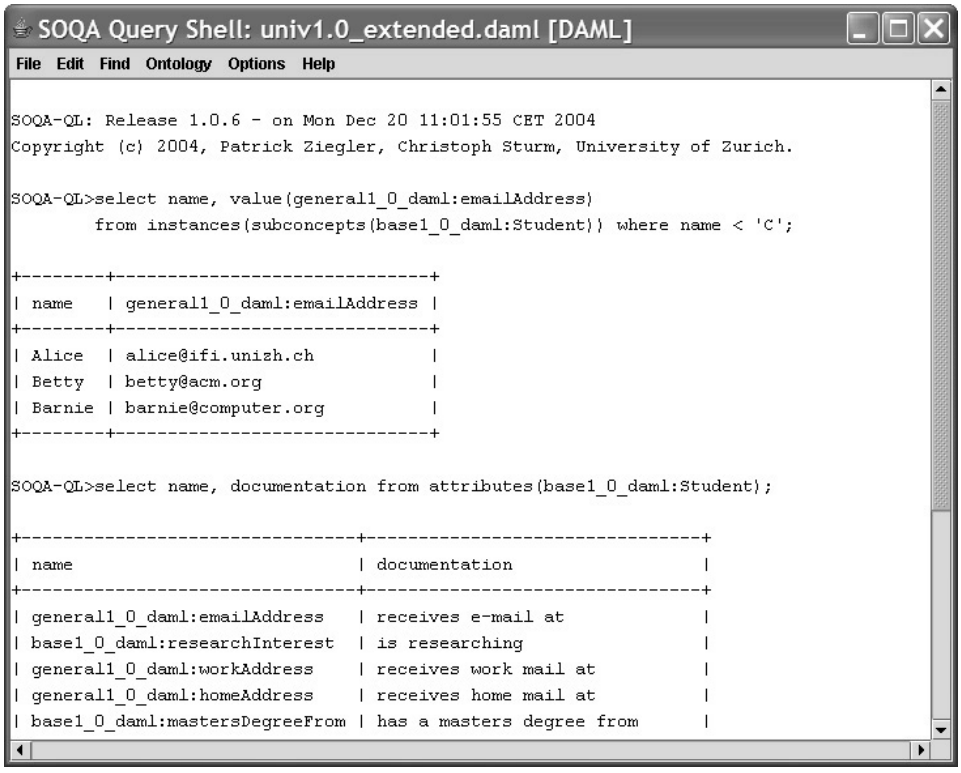

Figure 4: The Example SOQA-QL Queries (Q6) and (Q2) Asked Against the (Extended) DAML University Ontology

[14]https://javacc.dev.java.net/

5 The SOQA Browser

Besides SOQA-QL, the SOQA Browser represents a second application that provides unified ontology access based on the API of SOQA (see Fig. 3). The SOQA Browser is a tool to graphically inspect all ontology information that can be accessed through SOQA — that is, all ontological information as defined in the SOQA Ontology Meta Model. The SOQA Browser can thus be used to quickly overview the concepts and their attributes, methods, relationships, and instances that are defined in a particular ontology as well as the metadata (author, version, ontology language name, etc.) concerning the ontology itself.

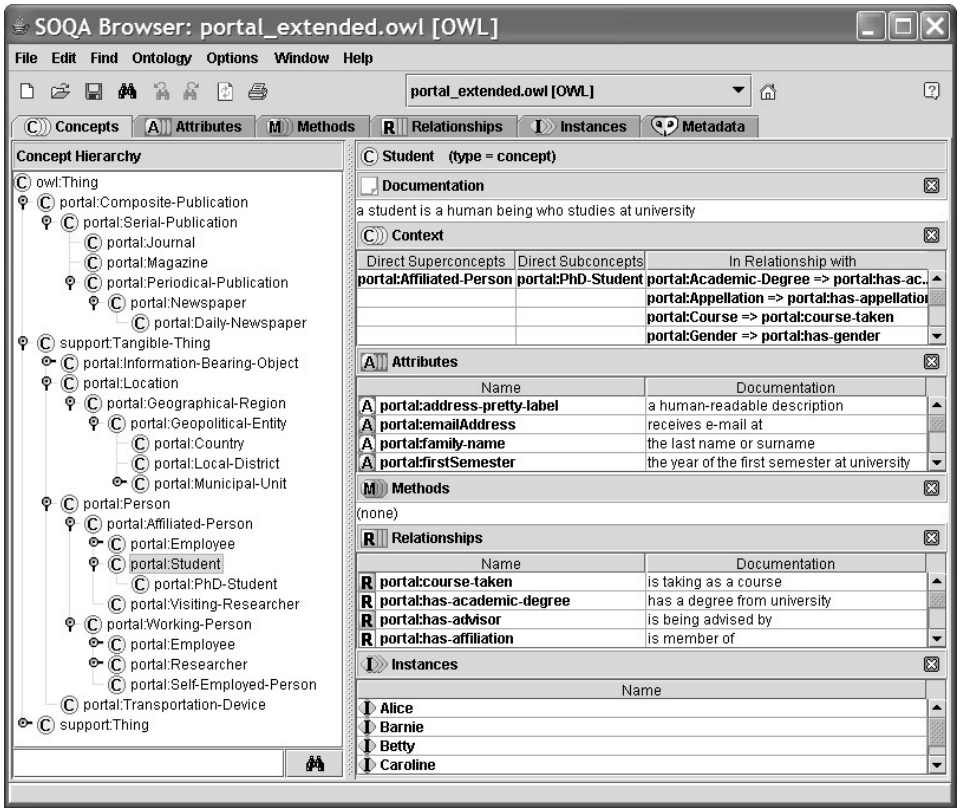

Figure 5: The (Extended) Aktors Portal Ontology in the SOQA Browser

For ease of use, the user interface of the SOQA Browser is similar to the widely-used ontology editor Protégé[15]; hence, the SOQA Browser generally provides a series of tabs that represent different views on the particular ontology. For instance, the concept View Tab as shown in Fig. 5 presents a taxonomy of all concepts of the respective ontology together with additional information on the concept that is currently selected in the concept

[15]http://protege.stanford.edu/

hierarchy. This information includes documentation on the concept itself as well as more detailed information on the attributes, methods, relationships, and instances of the selected concept. Besides the concept View Tab, the SOQA Browser provides tailored view tabs for attributes, methods, relationships, instances, and ontology metadata. In essence, the SOQA Browser provides a unified view on ontologies independently of the ontology language that is used for the particular underlying ontology.

In our running example, the SOQA Ontology Browser can be used by the developer to quickly get an overview of the three ontologies represented in PowerLoom, OWL, and DAML respectively. For instance, by inspecting the concept hierarchy and considering the available information on concepts as shown by the concept View Tab, our developer can more easily and more rapidly determine whether the three individual ontologies (or a combination of them) fit the respective needs. In contrast to a conventional ontology access scenario where several ontology-language specific tools are employed, the developer who takes advantage of SOQA does not have to cope with the different ontology languages that are used for ontology representation. Ontology access and reuse can hence be facilitated based on the services provided by SOQA and SOQA-based tools for the benefit of human users and applications.

6 Related Work

In general, APIs that offer access to a single ontology language on behalf of programming languages, such as Java, can be regarded as work related to SOQA. For instance, the WonderWeb OWL API [BVL03] provides access to data structures representing OWL ontologies. Another example is the DIG Description Logic Interface [BMC03] that is an API for description logic systems [BCM+03]. In contrast to SOQA, the focus of these APIs is generally on accessing content described in a *single* ontology language.

Frameworks that support access to a family of ontology languages are another area of related work. For instance, the Stanford RDF API [Mel01] provides interfaces and methods for parsing and accessing RDF models (that are the foundation of RDF-based ontology languages). Another example is Jena 2 [CDD+04] which is a Java framework for writing Semantic Web applications. It provides an ontology API for accessing ontology data that is based on RDF; that is, more precisely, OWL, DAML+OIL [USEU01], and RDF Schema ontology data. SOQA differs from both, the Stanford RDF API and the Jena 2 ontology API, in that it not only provides access to RDF-based ontology data but offers the means to homogeneously retrieve ontological data and metadata from a manifold of ontology languages. This includes not only ontological data described with recent Semantic Web languages (such as OWL, DAML, OIL, SHOE, etc.), but also data represented in traditional ontology languages (e.g., Ontolingua, PowerLoom, OCML, or F-Logic). Furthermore, SOQA supports ontologies supplied by knowledge bases, such as Cyc/OpenCyc, by lexical ontology systems, such as WordNet, and by protocols, such as OKBC, that provide access to knowledge representation systems.

Closest to SOQA is the ontology-neutral API being under development in the Ontology

Management Working Group (OMWG)[16]. The OMWG's goal is to build an ontology management suite that will provide access to different ontology repositories and reasoning engines through an ontology-neutral API. In contrast to SOQA, the future API will not be the same for all ontology languages but a commonly shared core API will be extended for each particular ontology language to reflect certain particularities. The OMWG ontology meta model is similar to our meta model and consists of concepts, relations, functions, instances, and axioms. Another related ontology metamodel is the Ontology Definition Metamodel (ODM)[17] being designed and discussed in the Object Management Group (OMG). ODM aims at providing a basis for the development of OWL ontologies using UML modeling tools. Hence, the focus and intended use of ODM differ from the respective one of the SOQA Ontology Metamodel.

Concerning SOQA-QL and the SOQA Browser, related work can be found in the areas of query languages for the Semantic Web and ontology browsers, respectively. However, space limitations prevent us from discussing them in detail in this paper.

7 Conclusions and Future Work

In this paper, we presented the SIRUP Ontology Query API (SOQA), an ontology language- and platform-independent API for query access to metadata and data of ontologies. In contrast to existing ontology APIs, SOQA provides read access to ontologies that can be represented in a manifold of ontology languages. This includes not only ontologies specified in recent Semantic Web languages (such as OWL, DAML, OIL, SHOE, etc.), but also in traditional ontology languages (e.g., Ontolingua, PowerLoom, OCML, or F-Logic). Furthermore, SOQA supports ontologies supplied by knowledge bases, such as Cyc/OpenCyc, by lexical ontology systems, such as WordNet, and by protocols, such as OKBC, that provide access to knowledge representation systems. Despite the fact that SOQA is currently employed for ontology access used for data content explication in the SIRUP approach [ZD04b] to semantic data integration, SOQA is intended and designed to be a general-purpose ontology query API that can be used independently of SIRUP. All in all, SOQA comprises more than 70 Java methods for unified ontology access.

Based on SOQA's general-purpose infrastructure for uniform ontology access, ease of use when dealing with ontologies can be improved by suitable tools that take advantage of services provided by SOQA:

- The SOQA Query Language (SOQA-QL) is an SQL-like query language that supports declarative queries against ontology data and metadata that are independent of the ontology language particular ontologies are represented in.
- The SOQA Browser is a tool to graphically inspect all ontology information that can be accessed through SOQA — that is, all ontological information as defined in the SOQA Ontology Meta Model.

[16] http://www.omwg.org
[17] http://www.omg.org/docs/ontology/03-03-01.pdf

- Last, but not least, (third-party) Java applications can use SOQA as a single point of access to information that is specified in different ontology languages. Possible application areas are knowledge management, e-business, enterprise application integration, and the Semantic Web.

Besides the SOQA API itself, the components to formulate and execute SOQA-QL queries, and the SOQA Browser, we have implemented SOQA ontology wrappers for OWL, PowerLoom, DAML, and the lexical ontology WordNet up to now. All these components are fully implemented in Java 1.5. In the future, we plan to implement additional wrappers for other ontology languages, such as OIL, SHOE, or F-Logic. Furthermore, we intend to support Cyc/OpenCyc, which is a general ontology for commonsense knowledge, as well as the OKBC protocol to access information from knowledge representation systems. Other work considered for the future includes the development of a SOQA plugin for Protégé, the extension of SOQA with functionality to edit ontologies, and the implementation of SOQA-based ontology language converters.

References

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[BHH+04] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. W3C, 10 February 2004. http://www.w3.org/TR/owl-ref/.

[BMC03] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG Description Logic Interface. In Diego Calvanese, Giuseppe De Giacomo, and Enrico Franconi, editors, *2003 International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, Rome, Italy September 5-7, 2003.

[BVL03] Sean Bechhofer, Raphael Volz, and Phillip Lord. Cooking the Semantic Web with the OWL API. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *Second International Semantic Web Conference (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*, pages 659–675, Sanibel Island, FL, USA, October 20-23, 2003. Springer.

[CDD+04] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: Implementing the Semantic Web Recommendations. In *13th World Wide Web Conference (WWW 2004)*, pages 74–83, New York, NY, USA, May 17-22, 2004. ACM.

[CFF+98] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James Rice. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In *Fifteenth National Conference on Artificial Intelligence (AAAI 1998)*, pages 600–607, Madison, Wisconsin, USA, July 26-30, 1998. AAAI Press / The MIT Press.

[CHS+95] M.J. Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.F. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams, and E.L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic

Approach. In *5th International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM 1995)*, pages 124–131, Taipei, Taiwan, March 6-7, 1995.

[FFR97] Adam Farquhar, Richard Fikes, and James Rice. The Ontolingua Server: A Tool for Collaborative Ontology Construction. *International Journal of Human-Computer Studies*, 46(6):707–727, 1997.

[FHH+01] Dieter Fensel, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, 1995.

[GMS94] Cheng Hian Goh, Stuart E. Madnick, and Michael Siegel. Context Interchange: Overcoming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment. In *Third International Conference on Information and Knowledge Management (CIKM 1994)*, pages 337–346, Gaithersburg, USA, November 29 - December 2, 1994. ACM.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.

[Len95] Douglas B. Lenat. CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38(11):32–38, 1995.

[LR82] Terry Landers and Ronni L. Rosenberg. An Overview of MULTIBASE. In Hans-Jochen Schneider, editor, *Second International Symposium on Distributed Data Bases (DDB 1982)*, pages 153–184, Berlin, Germany, September 1-3, 1982. North-Holland.

[LSRH97] Sean Luke, Lee Spector, David Rager, and James Hendler. Ontology-based Web Agents. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *First International Conference on Autonomous Agents (Agents 1997)*, pages 59–68, Marina Del Rey, CA, USA, February 5-8, 1997. ACM.

[MCM03] Robert M. MacGregor, Hans Chalupsky, and Eric R. Melz. PowerLoom Manual, 29 October 2003. http://www.isi.edu/isd/LOOM/PowerLoom/documentation/.

[Mel01] Sergey Melnik. The Stanford RDF API, 2001. http://www-db.stanford.edu/∼melnik/rdf/api.html.

[Mil95] George A. Miller. WordNet:A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[MKSI96] Eduardo Mena, Vipul Kashyap, Amit P. Sheth, and Arantza Illarramendi. OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation Across Pre-existing Ontologies. In *First IFCIS International Conference on Cooperative Information Systems (CoopIS 1996)*, pages 14–25, Brussels, Belgium, June 19-21, 1996. IEEE Computer Society.

[Mot99] Enrico Motta. *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*. IOS Press, Amsterdam, 1999.

[OS99] Aris M. Ouksel and Amit P. Sheth. Semantic Interoperability in Global Information Systems: A Brief Introduction to the Research Area and the Special Section. *SIGMOD Record*, 28(1):5–12, 1999.

[SEGM+90] Peter Scheuermann, Ahmed K. Elmagarmid, Hector Garcia-Molina, Frank Manola, Dennis McLeod, Arnon Rosenthal, and Marjorie Templeton. Report on the Workshop on Heterogenous Database Systems held at Northwestern University, Evanston, Illinois, December 11-13, 1989. *SIGMOD Record*, 19(4):23–31, 1990.

[UG96] Mike Uschold and Michael Grüninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[USEU01] Joint US/EU ad hoc Agent Markup Language Committee. The DAML+OIL Language Specification (March 2001). http://www.daml.org/2001/03/daml+oil-index.html.

[ZD04a] Patrick Ziegler and Klaus R. Dittrich. Three Decades of Data Integration - All Problems Solved? In René Jacquart, editor, *18th IFIP World Computer Congress (WCC 2004)*, volume 156 of *IFIP Conference Proceedings*, pages 3–12, Toulouse, France, August 22-27, 2004. Kluwer.

[ZD04b] Patrick Ziegler and Klaus R. Dittrich. User-Specific Semantic Integration of Heterogeneous Data: The SIRUP Approach. In Mokrane Bouzeghoub, Carole Goble, Vipul Kashyap, and Stefano Spaccapietra, editors, *First International IFIP Conference on Semantics of a Networked World (ICSNW 2004)*, volume 3226 of *Lecture Notes in Computer Science*, pages 44–64, Paris, France, June 17-19, 2004. Springer.

A Simplified Grammar of SOQA-QL in Extended Backus-Naur Form[15]

```
SOQAQuery            ::= SOQAQueryExpression ( ( <INTERSECT> | <UNION> |
                           <MINUS> ) SOQAQueryExpression )* ";"
SOQAQueryExpression  ::= <SELECT> ProjectionAttributes
                         <FROM> FromStatement
                         ( <WHERE> Condition )? ( <ORDER> <BY> Sort )?
FromStatement        ::= MetaModelElementSet ( AsIdentifier )?
                           ( "," MetaModelElementSet ( AsIdentifier )? )*
AsIdentifier         ::= <AS> <IDENTIFIER>
Sort                 ::= Attribute ( "," Attribute )* ( <ASC> | <DESC> )?
Condition            ::= AttributeComparison ( (<AND>|<OR>) Condition )?
AttributeComparison  ::= ( ( (<NOT>)? Attribute <IN>
                           "(" SOQAQueryExpression ")" ) |
                         ( (<NOT>)? Attribute <COMPARISON_OPERATOR>
                           ( Attribute | Value ) ) )
ProjectionAttributes ::= AggregateFunction | <ALL> |
                         ( Attribute ( "," Attribute )* )
AggregateFunction    ::= ( ( ( <MIN> | <MAX> | <AVG> | <SUM> )
                           "(" Attribute ")" ) |
                         ( <COUNT> "(" ( <ALL> | Attribute ) ")" ) )
Attribute            ::= <IDENTIFIER> | ValueFunction
ValueFunction        ::= <VALUE> "(" ( <IDENTIFIER> ) ")"
MetaModelElementSet  ::= Attribute | Function |
                           "(" SOQAQueryExpression ")"
Value                ::= ( <INTEGER_LITERAL> | <FLOAT_LITERAL> |
                           "'" <IDENTIFIER> "'" )
Function             ::= ( <SUPERCONCEPTS> | <SUBCONCEPTS> | <...> )
                           ( "()" | ( "(" ( MetaModelElementSet|Value )
                           ( "," ( MetaModelElementSet|Value ) )* ")" ) )
```

[15]For the sake of brevity, "`<...>`" in the `Function` production rule is a replacement for all names of all functions besides `getSuperConcepts` and `getSubConcepts` that represent API methods from SOQA.