Querying the Web Reconsidered:
Design Principles for Versatile Web Query Languages

François Bry, University of Munich, Germany
Christoph Koch, University of Technology, Ausria
Tim Furche, University of Munich, Germany
Sebastian Schaffert, University of Munich, Germany
Liviu Badea, Nat. Institute for Research and Development in Informatics, Bucharest, Germany
Sacha Berger, University of Munich, Germany

ABSTRACT

A decade of experience with research proposals as well as standardized query languages for the conventional Web and the recent emergence of query languages for the Semantic Web call for a reconsideration of design principles for Web and Semantic Web query languages. This chapter first argues that a new generation of versatile *Web query languages is needed for solving the challenges posed by the changing Web: We call versatile those query languages able to cope with both Web and Semantic Web data expressed in any (Web or Semantic Web) markup language. This chapter further suggests that well-known referential transparency and novel answer-closedness are essential features of versatile query languages. Indeed, they allow queries to be considered like forms and answers like form-fillings in the spirit of the query-by-example paradigm. This chapter finally suggests that the decentralized and heterogeneous nature of the Web requires incomplete data specifications (or incomplete queries) and incomplete data selections (or incomplete answers); the form-like query can be specified without precise knowledge of the queried data, and answers can be restricted to contain only an excerpt of the queried data.*

Keywords: Please provide

INTRODUCTION

After a decade of experience with research proposals as well as standardized query languages for the conventional Web, and after following the recent emergence of query languages for the Semantic Web a reconsideration of design principles for Web and Semantic Web query languages is called for.

The Semantic Web is an endeavor widely publicized in 2001 by an influential but also controversial article from Tim Berners-Lee, James Hendler, and Ora

Lassila (Berners-Lee et al., 2001). The Semantic Web vision is that of the current Web which consists of (X)HTML and documents in other XML formats extended by metadata specifying the meaning of these documents in forms usable by both humans and computers.

One might see the Semantic Web metadata added to today's Web documents as semantic indices similar to encyclopedias. A considerable advantage over paper-printed encyclopedias is that the relationships expressed by Semantic Web metadata can be followed by computers, very much like hyperlinks, and be used for drawing conclusions using automated reasoning methods:

> *For the Semantic Web to function, computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning.* (Berners-Lee et al., 2001, p. 000)

A number of formalisms have been proposed in recent years for representing Semantic Web metadata (e.g., RDF [Klyne et al., 2004], Topic Maps [ISO, 1999], and OWL [Bechhofer et al., 2004]). Whereas RDF and Topic Maps provide merely a syntax for representing assertions on relationships like "a text *T* is authored by person *P*," schema or ontology languages such as RDFS (Brickley et al., 2004) and OWL allow one to state properties of the terms used in such assertions (e.g., that no person can be a text). Building upon descriptions of resources and their schemas, as detailed in the architectural road map for the Semantic Web (Berners-Lee, 1998), rules expressed in SWRL (Horrocks et al., 2004) or RuleML (Boley et al., 2002), for example, allow the specification of actions to be taken, knowledge to be derived, or constraints to be enforced.

Essential for realizing this vision is the integrated access to *all* kinds of data represented in any of these representation formalisms or even in standard Web languages such as (X)HTML, SVG. Considering the large amount and the distributed storage of data already available on the Web, the efficient and convenient access to such data becomes *the* enabling requirement for the Semantic Web vision. It has been recognized that reasonably high-level, declarative query languages are needed for such efficient and convenient access, as they allow separation of the actual data storage from the view of the data that a query programmer operates on. This chapter presents a novel position on design principles for guiding the development of query languages that allow access to both standard and Semantic Web data. The authors believe that it is worthwhile to reconsider principles that have been stated almost a decade ago for query languages such as XML-QL (Deutsch et al., 1998) and XQuery (Boag et al., 2004), then agnostic of the challenges imposed by the emerging Semantic Web.

Three principles are at the core of this chapter:

- As discussed above, the same query language should provide convenient and efficient access to any kind of data expected to be found on the Semantic Web (e.g., to documents written in

(X)HTML, to RDF descriptions of these documents, and even to ontologies). Only by intertwining data from all the different layers of the Semantic Web can vision be realized in its full potential.

- Convenience for the user of the query language requires the reuse of knowledge obtained in another context. Therefore, the query language should be based upon the principles of referential transparency and answer-closedness (see Section 2.4) realized by rules and patterns. Together, these principles allow for (1) querying existing and constructing new data by a form-filling approach (similar to but arguably more expressive than the query-by-example paradigm [Zloof, 1975]); and (2) basic reasoning capabilities including the provision of different views of the same data even represented in different Web formalisms.
- The decentralized and heterogeneous nature of the Web requires query languages that allow queries and answers to be incomplete: In queries, only known parts of the requested information are specified, similar to a form, leaving other parts incomplete. Conversely, the answer to a query may leave out uninteresting parts of the matching data.

It is worth noting that these core principles and the more detailed discussion of the design principles in Section 2 are describing general principles of query languages rather than specific issues of an implementation or storage system. Therefore, implementation issues, such as processing model (in-memory vs. database vs. data stream) or distributed query evaluation, are not discussed in this chapter. Rather, the language requirements are considered independently of such issues but allow for further extensions or restrictions of the language, if necessary, for a particular setting or application.

These design principles result in large part from experience in the design of Web query languages by the authors, in particular from the experience in designing the Web query language Xcerpt (Schaffert & Bry, 2004).

DESIGN PRINCIPLES

The rest of this chapter is organized around 13 design principles deemed essential for versatile Web query languages, starting with principles concerning the dual use of a query language for both Web and Semantic Web data (Section 2.1), the specific requirements on how to specify data selection (Section 2.2), the makeup of an answer (Section 2.3), further principles regarding declarativity and structuring of query programs (Section 2.4), reasoning support (Section 2.5), and finally, those regarding the relation of querying and evolution (Section 2.6) are outlined.

Versatility: Data, Syntax, and Interface

A Query Language for the Standard Web and Semantic Web

A hypothesis of this chapter is that a *common query language* for both conventional Web and Semantic Web appli-

cations is desirable (this requirement for a Web query language also has been expressed by other authors (e.g., Olken & McCarthy, 1998). There are two reasons for this hypothesis:

First, in many cases, data is not inherently conventional Web data or Semantic Web data. Instead, it is the usage that gives data a conventional Web or Semantic Web status. Consider, for example, a computer science encyclopedia. It can be queried like any other Web document using a Web query language. If its encyclopedia relationships (i.e., formalizing expressions such as "see," "see also," or "use instead," commonly used in traditional encyclopedia) are marked up using, for example, XLink or any other ad hoc or generic formalism, as one might expect from an online encyclopedia, then the encyclopedia also can be used as Semantic Web data (i.e., metadata) in retrieving computer science texts (e.g., the encyclopedia could relate a query referring to Linux to Web content referring to "operating systems of the 90s") or enhance the rendering of Web contents (e.g., adding hypertext links from some words to their definitions in the encyclopedia).

Second, Semantic Web applications most likely will combine and intertwine queries to Web data and to metadata (or Semantic Web data) in all possible manners. There is no reason to assume that Semantic Web applications will rely only on metadata or that querying of conventional Web data and Semantic Web data will take place in two (or several) successive querying phases, referring each to data of one single kind. Consider again the computer science encyclopedia example. Instead of one single encyclopedia, one might use several encyclopedias that might be listed in a (conventional Web) document. Retrieving the encyclopedias requires a conventional Web query. Merging the encyclopedias is likely to call for specific features of a Semantic Web query language. Enhancing the rendering of a conventional Web document using the resulting (merged) encyclopedia is likely to require (a) conventional Web queries (for retrieving conventional Web documents and the addresses of the relevant encyclopedias), (b) Semantic Web queries (for merging the encyclopedias), or (c) mixed conventional and Semantic Web queries (for adding hypertext links from words defined in the merged encyclopedia).

Integrated View of Standard and Semantic Web Data: Graph Data

Both XML and semi-structured data in general, as predominantly used on the (standard) Web, and RDF, the envisioned standard for representing Semantic Web data, can be represented in a graph data model. Although XML is often seen as a tree model only (see XML Information Set [Cowan & Tobin, 2004] and the XQuery data model [Fernandez et al., 2004]), it does provide nonhierarchical relations (e.g., by using ID/IDREF links or XLink [DeRose et al., 2001]).

Similar to the proposal for an integrated data model and (model-theoretic) Semantics of XML and RDF presented in Patel-Schneider and Simeon (2002), a query language for both standard and Semantic Web must be able to query any such data in a natural way. In particular,

an abstraction of the various linking mechanisms is desirable for easy query formulation; one approach is the automatic dereferencing of ID/IDREF-links in XML data, and another is the unified treatment of typed relations provided both in RDF and XLink.

The restriction to hierarchical (i.e., acyclic) relations is not realistic beyond the simplest Semantic Web use cases. Even if each relation for itself is acyclic, inference based not only on relations of a single type must be able to cope with cycles. Therefore, a (rooted) graph data model is called for.

Three Syntaxes: XML, Compact Human-Readable, and Visual

While it is desirable that a query language for the (conventional and/or Semantic) Web has an XML syntax, because it makes it easier to exchange query programs on the Web and to manipulate them using the query language, a second, more compact syntax easier for humans to read and write is desirable. Therefore, two textual syntaxes should be provided: a purely term-oriented XML syntax and another one that combines term expressions with non-term expressions like most programming languages. This other syntax should be more compact than the XML syntax and better readable for human beings. Both syntaxes should be interchangeable (the translation being a low cost process).

Third, a visual syntax can greatly increase the accessibility of the language, in particular for non-experts. This visual syntax should be a mere rendering of the textual language, a novel approach to developing a visual language with several advantages. It results in a visual language tightly connected to the textual language; namely, it is a rendering of the textual language. This tight connection makes it possible to use both the visual and the textual language in the development of applications. Last, but not least, a visual query language conceived as a hypertext application is especially accessible for Web and Semantic Web application developers.

Modeling, Verbalizing, and Visualizing

Authoring and Modeling. Authoring correct and consistent queries often requires considerable effort from the query programmer. Therefore, semi-automated or fully automated tool support both for authoring and for reading and understanding queries is essential.

Verbalization. For verbalizing queries, as well as their input and output, some form of controlled natural language processing is promising and can provide an interface to the query language for untrained users. The importance of such a seemingly free-form "natural" interface for the Web is demonstrated by the widespread success of Web search engines.

Visualization. As already discussed, a visualization based on styling of queries is highly advantageous in a Semantic Web setting. As demonstrated in Berger et al. (2003], it also can serve as a foundation for interactive features such as authoring of queries. On this foundation, more advanced authoring tools (e.g., for verifica-

tion and validation of queries) can be implemented.

Data Selection: Pattern-Based, Incomplete

Every query language has to define means for accessing or selecting data. This section discusses principles for data selection in a Web context.

Pattern Queries

Patterns, as used in Xcerpt (Schaffert & Bry, 2004) and XML-QL (Deutsch et al., 1998), for example, provide an expressive and yet easy-to-use mechanism for specifying the characteristics of data sought. In contrast to path expressions, as used in XPath (Clark & DeRose, 1999) and languages building upon it, for example, they allow an easy realization of answer-closedness in the spirit of "query by example" query languages. Query patterns are especially well suited for a visual language, because they give queries a structure very close to that of possible answers. One might say that query patterns are like forms and answers are like form fillings.

Incomplete Query Specifications

Incomplete queries specify only part of the data to retrieve (i.e., only some of the children of an XML element (referring to the tree representation of XML data called "incompleteness in breadth") or an element at unspecified nesting depth (referring to the tree representation of XML data called "incompleteness in depth"). Such queries are important on the conventional Web because of its heterogeneity; one often knows only part of the structure of the XML documents to retrieve.

Incomplete queries specifying only part of the data to retrieve are also important on the Semantic Web. There are three reasons for this: first, Semantic Web data such as RDF or Topic Map data might be found in different (XML) formats that, in general, are easier to compare in terms of only some salient features. Second, the merging of Semantic Web data is often done in terms of components common to distinct data items. Third, most Semantic Web data standards allow data items with optional components. In addition, query languages for the conventional and Semantic Web should ease retrieving only parts of (completely or incompletely specified) data items.

Incomplete Data Selections

Because Web data is heterogeneous in its structure, one is often interested in "incomplete answers." Two kinds of incomplete answers can be considered. First, one might not be interested in some of the children of an XML (sub-) document retrieved by a query. Second, one might be interested in some child elements if they are available, but would accept answers without such elements.

An example of the first case would be a query against a list of students asking for the name of students having an e-mail address but specifying that the e-mail address should not be delivered with the answer.

An example of the second case would be a query against an address book asking for names, e-mail addresses, and, if available, cellular phone numbers.

But the limitation of an answer to "interesting" parts of the selected data is helpful not only for XML data. A common desire when querying descriptions of Web sites, documents, or other resources stored in RDF is to query a description of a resource (i.e., everything related to the resource helping to understand or identify it). In this case, for example, one might want to retrieve only data related by, at most, *n* relations to the original resource and also avoid following certain relation types not helpful in identifying a resource.

Polynomial Core

The design principles discussed in this chapter point toward a general-purpose, and due to general recursion, most likely Turing-complete, database programming language. However, it is essential that for the most frequently used queries, small upper bounds on the resources taken to evaluate queries (i.e., main memory and query evaluation time) can be guaranteed. As a consequence, it is desirable to identify an interesting and useful fragment of a query language for which termination can be guaranteed and which can be evaluated efficiently.

When studying the complexity of database query languages, one distinguishes between at least three complexity measures: data complexity, where the database is considered to be the input and the query is assumed fixed; query complexity, where the database is assumed fixed and the query is the input; and combined complexity, which takes both the database and the query as input and expresses the complexity of query evaluation for the language in terms of the sizes of both (Vardi, 1982).

For a given language, query and combined complexity are usually much higher than data complexity. In most relational query languages are by one exponential factor harder (e.g., in PSPACE vs. LOGSPACE-complete for first-order queries and EXPTIME-complete vs. PTIME-complete for Datalog, cf. [Abiteboul et al., 1995]). On the other hand, since data sizes are usually much larger than query sizes, the data complexity of a query language is the dominating measure of the hardness of queries.

One complexity class that is usually identified with efficiently solvable problems (or queries) is that of all problems solvable in polynomial time. PTIME queries still can be rather inefficient on large databases. Another even more desirable class of queries would thus be that of those queries solvable in linear time in the size of the data.

Database theory provides us with a number of negative results on the complexity of query languages that suggest that neither polynomial-time query complexity nor linear-time data complexity are feasible for data-transformation languages that construct complex structures as the result. For example, even conjunctive relational queries are NP-complete with respect to query complexity (Chandra & Merlin, 1977). Conjunctive queries only can apply selection, projection, and joins to the input data, all features that are

among the requirements for query languages for the Semantic Web. There are a number of structural classes of tractable (polynomial-time) conjunctive queries, such as those of so-called "bounded tree-width" (Flum et al., 2002) or "bounded hypertree-width" (Gottlob et al., 2002), but these restrictions are not transparent or easy to grasp by users. Moreover, even if such restrictions are made, general data transformation queries only need very basic features (i.e., joins or pairing) to produce query results that are of super-linear size. That is, just writing the results of such queries is not feasible in linear time.

If one considers more restrictive queries that view data as graphs or, more precisely, as trees, and which only select nodes of these trees, there are a number of positive results. The most important is the one that monadic (i.e., node-selecting) queries in monadic second-order logic on trees are in linear time with respect to data complexity (Courcelle, 1990) but have non-elementary query complexity (Grohe & Schweikardt, 2003). Reasoning on the Semantic Web naturally happens on graph data, and results for trees remain relevant because many graphs are trees. However, the linear time results already fail, if very simple comparisons of data values in the trees are permitted.

Thus, the best we can hope for in a data transformation query language fragment for reasoning on the Semantic Web is PTIME data complexity. This is usually rather easy to achieve in query languages by controlling the expressiveness of higher-order quantification and recursion. In particular, the latter is relevant in the context of the design principles laid out here. A PTIME upper bound on the data complexity of recursive query languages is achieved either by disallowing recursion or by imposing an appropriate monotonicity requirement (i.e., those that form the basis of PTIME data complexity in standard Datalog or Datalog with inflationary fix-point semantics (Abiteboul et al., 1995).

Finding a large fragment of a database programming language and determining its precise complexity is an important first step. However, even more important than worst-case complexity bounds is the efficiency of query evaluation in practice. This leads to the problem of query optimization. Optimization also is usually best done on restricted query language fragments, in particular if such fragments exhibit alternative algebraic, logical, or game-theoretic characterizations.

Answers: Arbitrary XML, Ranked

Answers as Arbitrary XML Data

XML is the *lingua franca* of data interchange on the Web. As a consequence, answers should be expressible in every possible XML application. This includes both text without markup and freely chosen markup and structure. This requirement is obvious and widely accepted for conventional Web query languages. Semantic Web query languages also should be capable of delivering answers in every possible XML application in order to make it possible, for instance, to mediate between RDF and XTM (an XML serialization of Topic Maps [Pepper & Moore, 2001]) data or to translate RDF data from

one RDF syntax into another RDF syntax.

Answer Ranking and Top-k Answers

In contrast to queries posed to most databases, queries posed to the conventional and Semantic Web might have a rather unpredictable number of answers. As a consequence, it is often desirable to rank answers according to some application-dependent criteria. It is desirable that Web and Semantic Web query languages offer (a) basic means for specifying ranking criteria and, (b) for efficiency reasons, evaluation methods computing only the top-*k* answers (i.e., a given number *k* of best-ranked answers according to a user-specified ranking criterion).

Query Programs: Declarative, Rule based

The following design principles concern the design of query programs beyond the data selection facilities discussed in Section 2.2.

Referential Transparency

This property means that within a definition scope, all occurrences of an expression have the same value (i.e., denote the same data). Referential transparency is an essential, precisely-defined trait of the rather vague notion of "declarativity."

Referential transparency is a typical feature of modern functional programming languages. For example, evaluating the expression f 5 in the language Haskell will always yield the same value (assuming the same definition of f is used). Contrasting with languages like C or Java: the expression f(5) might yield different results every time it is called because its definition depends on constantly changing state information.

Referentially transparent programs are easier to understand and, therefore, easier to develop, maintain, and optimize as referential transparency allows query optimizers to dynamically rearrange the evaluation order of (sub-) expressions (e.g., for evaluating in a "lazy manner" or computing an optimal query evaluation plan). Therefore, referential transparency surely is one of the essential properties a query language for the Web should satisfy.

Answer-Closedness

We call a query language "answer-closed" if replacing a sub-query in a compound query by a possible (not necessarily actual) single answer always yields a syntactically valid query. Answer-closed query languages ensure in particular that every data item (i.e., every possible answer to some query) is a syntactically valid query. Functional programs can but are not required to be answer-closed. Logic programming languages are answer-closed, but SQL is not (e.g., the answer person(a) to the Datalog query person(X) is itself a possible query, while the answer "name = 'a' " to the SQL query SELECT name FROM person cannot (without significant syntactical changes) be used as a query. Answer-closedness is the distinguishing property of the "query by example" paradigm (Zloof, 1975), even

though it is called differently there, separating it from previous approaches for query languages. Answer-closedness eases the specification of queries because it keeps limited the unavoidable shift in syntax from the data sought for and the query specifying these data.

To illustrate the importance of answer-closedness in the Web context, assume an XML document containing a list of books with titles, authors, and prices (e.g., the XML Query Use Case XMP [Chamberlain et al., 2003]). The XPath (Clark and DeRose, 1999) query

```
/bib/book/title/text()
```

selects the (text of) titles of books, while a similar query in the (answer-closed) language Xcerpt ([Schaffert and Bry, 2004) is

```
bib {{ book {{ title { var T } }} }}.
```

XPath does not allow to substitution (e.g., the string "Data on the Web" for the query is thus not answer-closed). In Xcerpt, on the other hand, the following is both an answer to the query and a perfectly valid query in itself:

```
bib {{ book {{ title { "Data on the
    Web" } }} }}
```

Answer-closedness is useful (e.g., when joining several documents). For instance, a query first could select book titles in a person's favorite book list and then substitute these titles in the previous query:

```
and {
 my-favorite-books {{ title { var T } }},
 bib {{ book {{ title { var T } }} }}
}
```

Rule-Based, Chaining, and Recursion

Rule-Based. Rules are understood here as means to specify novel, maybe virtual, data in terms of queries (i.e., what is called "views" in (relational) databases, regardless of whether this data is materialized or not). Views (i.e., rule-defined data) are desirable for both conventional and Semantic Web applications. There are three reasons for this:

First, view definitions or rules are a means for achieving the so-called "separation of concerns" in query programs (i.e., the stepwise specifications of data to retrieve and/or to construct). In other words, rules and view definitions are a means for "procedural abstraction" (i.e., rules [view definitions, resp.] are the Prolog and Datalog (SQL, resp.) counterpart of functions and/or procedures.

Second, rules and view definitions give rise to easily specifying inference methods needed (e.g., by Semantic Web applications).

Third, rules and view definitions are means for "data mediation." Data mediation means translating data to a common format from different sources. Data mediation is needed both on today's Web and on the emerging Semantic Web because of their heterogeneity.

Backward and Forward Chaining. On the Web, backward chaining (i.e., computing answers starting from rule

heads) is, in general, preferable to forward chaining (i.e., computing answers from rule's bodies). While forward chaining is, in general, considered to be more efficient then backward chaining, there are many situations where backward chaining is necessary, in particular when dealing with Web data. For example, a query might dynamically query Web pages depending on the results of previous queries and, thus, unknown in advance. Thus, a forward chaining evaluation would require considering the *whole* Web, which is clearly unfeasible.

Recursion. On the Web, recursion is needed at least for:

- traversing arbitrary-length paths in the data structure;
- querying on the standard Web when complex transformations are needed;
- querying on the Semantic Web when inference rules are involved.

Note that a free recursion is often desirable and that recursive traversals of XML documents as offered by the recursive computation model of XSLT 1.0 are not sufficient.

Separation of Queries and Constructions

Two standard and symmetrical approaches are widespread, as far as query and programming languages for the Web are concerned:

- Queries or programs are embedded in a Web page or Web page skeleton giving the structure of answers or data returned by calls to the programs.
- Parts of a Web page specifying the structure of the data returned to a query or program evaluation are embedded in the queries or programs.

It is a hypothesis of this chapter that both approaches to queries or programs are hard to read and, therefore, hard to write and maintain.

Instead of either approach, a strict separation of queries and "constructions" (i.e., expressions specifying the structure of answers) is desirable. With a rule-based language, constructions are rule heads, and queries are rule bodies. In order to relate a rule's construction to a rule's query (logic programming), variables can be employed.

As discussed in Section 2.13, the construction of complex results often requires considerable computation. The separation of querying and construction presented here allows for the separate optimization of both aspects, allowing easier adoption of efficient evaluation techniques.

Reasoning Capabilities

Versatility (Section 2.1) allows access to data in different representation formats, thereby addressing format heterogeneity. However, in a Web context, data often will be heterogeneous not only in the chosen representation format, but also in terms, structure, and so forth. Reasoning capabilities offer a means for the query author to deal with heterogeneous data and to infer new data.

Specific Reasoning as Theories

Many practical applications require special forms of reasoning; for instance, efficient equality reasoning is often performed using the so-called paramodulation rule instead of the equality axioms (transitivity, substitution, and symmetry). Also, temporal data might require conversions between different time zones and/or calendar systems that are expressed in a simpler format and more efficiently performed using arithmetic instead of logical axioms. Finally, reasoning with intervals of possible values instead of exact values (e.g., for appointment scheduling) is conveniently expressed and efficiently performed with constraint programming.

For this reason, it is desirable that a query language for the (conventional and Semantic) Web can be extended with so-called "theories" implementing specific forms of reasoning.

Such "theory extensions" can be realized in two manners:

First, a theory can be implemented as an extension of the runtime system of the query language with additional language constructs for using the extension.

Second, a theory can be implemented using the query language itself and made available to users of this query language through program libraries. In this case, theories are implemented by rules and queries. Based upon the XML syntax of the query language (Section 2.12), for example, such rule bases can then be queried using the query language itself and maintained and updated by a reactive language such as XChange (Bry & Pătrânjan, 2005).

Querying Ontologies and Ontology-Aware Querying

In a Semantic Web context, ontologies can be used in several alternative ways. First, they can be dealt with by a specialized ontology reasoner (the main disadvantage being the impossibility of adding new domain-specific constructs). Second, they can be regarded as descriptions to be used by a set of rules implementing the Semantics of the constructs employed by the ontology. (This is similar to a meta-interpreter and may be slow.) Alternatively, the ontology may be "compiled" to a set of rules.

As discussed in the previous point, the query language should allow for both approaches: extending the query language by specific theory reasoners for a certain ontology language (e.g., OWL-DL) as well as the ability to use rules written in the query language as means for implementing (at least certain aspects) of an ontology language. Examples for such aspects are the transitivity of the subsumption hierarchy represented in many ontologies or the type inference based on domain and range restrictions of properties.

The latter approach is based upon the ability to query the ontology together with the data classified by the ontology. This is possible due to the first design principle. Stated in terms of ontologies, we believe that a query language should be designed in such a way that it can query standard Web data (e.g., an article published on a Web site) in some XML document format metadata describing such Web data (e.g., resource descriptions in RDF stating author, usage restrictions, re-

lations to other resources, reviews, etc.), and the ontology that provides the concepts and their relations for the resource description in RDF.

Querying and Evolution

When considering the vision of the Semantic Web, the ability to cope with both quickly evolving and rather static data is crucial. The design principles for a Web query language discussed in the remainder of this section are mostly agnostic of changes in the data; only a snapshot of the current data is considered for querying; synchronization and distribution issues are transparent to the query programmer.

In many cases, such an approach is very appropriate and allows the query programmer to concentrate on the correct specification of the query intent. However, there are also a large number of cases where information about changes in the data and the propagation of such and similar events is called for (e.g., event notification, change detection, and publish-subscribe systems).

For programming the reactive behavior of such systems, one often employs "event-condition-action" (or ECA) rules. We believe that the specification of both queries on occurring events (the "event" part of ECA-rules) and on the condition of the data, which should hold for a specific action to be performed, should be closely related to or even embed the general purpose query language whose principles are discussed in this chapter (e.g., the reactive language XChange [Bry & P´trânjan, 2005] integrating the query language Xcerpt [Schaffert & Bry, 2004]).

RELATED WORK

Although there have been numerous approaches for accessing Web data, few approaches consider the kind of versatility asked for by the design principles presented in this chapter. This section briefly discusses how the design principles previously introduced relate to selected query languages for XML and RDF data, but does not aim at a full survey over current Web query languages as presented (Furche et al., 2004).

Versatility

Most previous approaches to Web query languages beyond format-agnostic information retrieval systems such as search engines have focused on access to one particular kind of data only (e.g., XML or RDF data). Therefore, such languages fall short of realizing the design principles on versatility described in Section 2.1. Connected to the realization that the vision of a "Semantic Web" requires joint access to XML and RDF data, versatility (at least when restricted to these two W3C representation standards) has been increasingly recognized as a desirable if not necessary characteristic of a Web query language (Patel-Schneider & Simeon, 2002). The charter of the W3C working group on RDF Data Access even asks "for RDF data to be accessible within an XML Query context [and] a way to take a piece of RDF Query abstract syntax and map it into a piece of XML Query" (Prud'hommeaux, 2004).

This recognition, however, has led mostly to approaches where access to

RDF data is added to already established XML query languages. Robie et al. (2001) proposes a library of XQuery accessor functions for normalizing RDF/XML and querying the resulting RDF triples. Notably, the functions for normalizing and querying actually are implemented in XQuery. In contrast, TreeHugger (Steer, 2003) provides a set of (external) extension functions for XSLT (1.0) (Clark, 1999). Both approaches suffer from the lack of expressiveness of the XQuery and XSLT data model when considering RDF data; XQuery and XSLT consider XML data as tree data where references (expressed using ID/IDREF or XLink) have to be resolved explicitly (e.g., by an explicit join or a specialized function). Therefore, Robie et al., (2001) maps RDF graphs to a flat, triple-like XML structure requiring explicit, value-based joins for graph traversal. TreeHugger maps the RDF graph to an XML tree, thus using the more efficient structural access, where possible, requiring special treatment, however, of RDF graphs that are not tree shaped. None of these approaches fulfills entirely the design principles proposed in Section 2.1, but they represent important steps in the direction of a versatile Web query language.

Data Selection

For the remainder of the design principles, Web query languages specialized for a certain representation format such as XML or RDF are worth considering. One of the most enlightening views on the state-of-the-art in both XML and RDF query languages is a view considering how data selection is specified in these languages. Both data formats allow structured information, and data selection facilities emphasize the selection of data based on its own structure and its position in some context (e.g., an XML document or an RDF graph). For specifying such structural relations, three approaches can be observed:

1. purely relational, where the structural relations are represented simply as relations, e.g., child(CONTEXT, X)Ùdescendant(X, Y) for selecting the descendants of a child of some node CONTEXT. This style is used in several RDF query languages (e.g., the widely used RDQL [Seaborne, 2004]) and current drafts of the upcoming W3C RDF query language SPARQL (Prud'hommeaux & Seaborne, 2004). For XML querying, this style has proven convenient for formal considerations of expressiveness and complexity of query languages, for example. In actual Web query languages, it can be observed only sparsely (e.g., in the Web extraction language Elog [Baumgartner et al., 2001]).
2. path-based, where the query language allows several structural relations along a path in the tree of graph structure to be expressed without explicit joins, e.g., child::*/descendant::* for selecting the descendants of childs of the context. This style, originating in object-oriented query languages, is used in the most popular XML query languages such as XPath (Clark & DeRose, 1999), XSLT ([Clark, 1999), and XQuery (Boag et al., 2004), but also

in a number of other XML query languages (XpathLog, 2004) that shows that this style of data selection also can be used for data updates. Several ideas to extend this style to RDF query languages have been discussed (Palmer, 2003), but only RQL (Karvounarakis et al., 2004) proposes a full RDF query language using path expressions for data selection.

3. pattern-based, as discussed in Section 2.2.1. This style is used (e.g., in XML-QL [Deutsch et al., 1998] and Xcerpt [Schaffert & Bry, 2004]) but is also well established for relational databases in the form "query-by-example" and Datalog.

Most Web query languages consider to some extent incomplete query specifications, as Web data is inherently inconsistent, and few assumptions about the schema of the data can be guaranteed. However, only few query languages take into account the two flavors of incomplete data selection discussed in Section 2.2.3 (e.g., Xcerpt [Schaffert & Bry, 2004] and SPARQL [Prud'hommeaux & Seaborne, 2004]).

Polynomial cores have been investigated most notably for XPath (and, therefore, by extension XQuery [Boag et al., 2004] and XSLT [Clark, 1999]); the results are presented, for example, in Gottlob et al. (2003) and Segoufin (2003).

Answers

Naturally, most XML query languages can construct answers in arbitrary XML. This, however, is not true of RDF query languages, many of which, such as RDQL (Seaborne, 2004) do not even allow the construction of arbitrary RDF, but rather outputs only (*n*-ary) tuples of variable bindings.

Answer ranking and top-*k* answers historically have rarely been provided by the core of Web query languages, but rather have been added as an extension (Amer-Yahia et al., 2004), a W3C initiative on adding full-text search and answer ranking to XPath and XQuery (Boag et al., 2004). In relational databases, on the other hand, top-*k* answers are a very common language feature.

Query Programs

Declarativity and referential transparency have long been acknowledged as important design principles for any query language, as a declaratively specified query is more amenable to optimization while also easing query authoring in many cases.

Most of the Web query languages claim to be declarative languages and, oftentimes claim to offer a referentially transparent syntax. In the case of XQuery (Boag et al., 2004), the referential transparency of the language is doubtful due to side effects during element construction. For instance, the XQuery `let $x = <a/> return $x is $x`, where is the XQuery node comparator (i.e., tests whether two nodes are identical, evaluates to true, whereas the query `<a/> is <a/>` evaluates to false, although it is obtained from the first query by replacing all occurrences of `$x` with its value.[1] The reason for this behavior lies in the way elements are constructed in XQuery: In the first query, a single (empty)

a is created, which is, of course, identical to itself. However, in the second case, two a elements are constructed, which are not identical, and, therefore, the node identity comparison using is fails. Interestingly, this behavior is related to XQuery's violation of design principle 2.4.4, that stipulates that querying and construction should be separated in a query language.

In contrast to referential transparency, answer-closedness cannot be observed in many Web query languages. With the exception of Xcerpt (Schaffert & Bry, 2004), Web query languages provide, if at all, only a limited form of answer-closedness, where only certain answers also can be used as queries.

Related to answer-closedness is the desire to be able to easily recognize the result of a query. This can be achieved by a strict separation of querying and construction, where the construction specifies a kind of form filled with data selected by the query. Such a strict separation is not used in most XML query languages but can be observed in many RDF query languages (e.g., RDF and SPARQL) due to the restricted form of construction considered in these languages (following a similar syntax as SQL, but restricting the SELECT clause to lists of variables, for example).

Section 2.4.3 proposes the use of (possibly recursive) rules for separation of concern, view specification. This has been a popular choice for Web query languages (e.g., XSLT [Clark, 1999], Algae [Prud'hommeaux, 2004]), in particular when combined with reasoning capabilities (e.g., in TRIPLE [Sintek, 2002], XPathLog [May, 2004]).

Reasoning Capabilities

Reasoning capabilities, as discussed in Section 2.5, are very convenient means to handle and enrich heterogeneous Web data. Nevertheless, the number of XML query languages featuring built-in reasoning capabilities is rather limited, examples being XPathLog (May, 2004) and Xcerpt (Schaffert & Bry, 2004). In contrast, several RDF query languages provide at least limited forms of reasoning for computing the transitive closure of arbitrary relations (e.g., TRIPLE [Sintek, 2002], Algae [Prud'hommeaux, 2004]. Some RDF query languages also consider ontology-aware querying with RDFS (Brickley et al., 2004) as ontology language. For XML query languages, this has not been considered at length.

CONCLUSION AND OUTLOOK

In this chapter, design principles for (Semantic) Web query languages have been derived from the experience with previous conventional Web query language proposals from academia and industry as well as recent Semantic Web query activities.

In contrast to most previous proposals, these design principles are focused on *versatile* query languages (i.e., query languages) able to query data in any of the heterogeneous representation formats used in both the standard and the Semantic Web.

As argued in Section 3, most previous approaches to Web query languages

fail to address the design principles discussed in this chapter; most notably, very few consider access to heterogeneous representation formats.

Currently, the Web query language Xcerpt (Schaffert & Bry, 2004), which already reflects many of these design principles, is being further refined to a true versatile query language along the principles outlined in this chapter.

We believe that versatile query languages will be essential for providing efficient and effective access to data on the Web of the future, effective as the use of data from different representation formats allows to serve better answers (e.g., by enriching, filtering, or ranking data with metadata available in other representation formats). Efficient as previous approaches suffer from the separation of data access by representation formats requiring either multiple query languages or hard to comprehend and expensive data transformations.

ACKNOWLEDGMENTS

We would like to thank Claude Kirchner (LORIA, Nancy, France) and Wolfgang May (Göttingen University, Göttingen, Germany) for providing numerous valuable comments on how to improve both presentation and content of an early draft of this chapter. We would also like to thank the anonymous reviewers for insightful comments on how to strengthen the arguments presented in this chapter.

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (see http://rewerse.net).

REFERENCES

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of databases*. Addison-Wesley.

Amer-Yahia, S., et al. (2004). *XQuery and XPath full-text* [working draft]. W3C.

Alferes, J.J., May, W., & Bry, F. (2004). Towards generic query, update, and event languages for the Semantic Web. *Proceedings of the Workshop on Principles and Practice of Semantic Web Reasoning*.

Baumgartner, R., Flesca, S., & Gottlob, G. (2001). The elog Web extraction language. *Proceedings of the International Conference on Logic Programming, Artifical Intelligence, and Reasoning (LPAR)*.

Bechhofer, S., et al. (2004). *OWL Web ontology language—Reference*. W3C.

Berger, S., Bry, F., & Schaffert, S. (2003). A visual language for Web querying and reasoning. *Proceedings of the Workshop on Principles and Practice of Semantic Web Reasoning*.

Berners-Lee, T. (1998). Semantic Web road map. W3C.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web—A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*.

Boag, S., et al. (Eds.) (2004). *XQuery 1.0: An XML query language* [work-

ing draft]. *W3C*. Retrieved from http://*www.w3.org/TR/xquery/*

Boley, H., et al. (2002). RuleML design. RuleML Initiative. Retrieved from *http://www.ruleml.org/indesign.html*

Brickley, D., Guha, R., & McBride, B. (Eds.) (2004). *RDF vocabulary description language 1.0: RDF Schema*. W3C, Recommendation.

Bry, F., Drabent, W., & Maluszynski, J. (2004). On subtyping of treestructured data—A polynomial approach. *Proceedings of the Workshop on Principles and Practice of Semantic Web Reasoning* .

Bry, F., Furche, T., Pătrânjan, P.-L., & Schaffert, S. (2004). Data retrieval and evolution on the (Semantic) Web: A deductive approach. *Proceedings of the Workshop on Principles and Practice of Semantic Web Reasoning*.

Bry, F. & Pătrânjan, P.-L. (2005). Reactivity on the Web: Paradigms and applications of the language XChange. *Proceedings of the ACM Symposium on Applied Computing (SAC)*.

Bry, F., Schaffert, S., & Schröder, A. (2004). A contribution to the semantics of Xcerpt, a Web query and transformation language. *Proc. of the Workshop Logische Programmierung*.

Chamberlin, D., Fankhauser, P., Florescu, D., Marchiori, M., & Robie, J. (Eds.) (2003). *XML query use cases*. W3C, Recommendation.

Chandra, A.K., & Merlin, P.M. (1977). Optimal implementation of conjunctive queries in relational data bases. *Proceedings of the ACM Symposium on Theory of Computing (STOC)*.

Clark, J. (1999). XSL transformation (XSLT) version 1.0. W3C, Recommendation.

Clark, J., & DeRose, S. (1999). *XML path language (XPath) version 1.0*, W3C, Recommendation.

Courcelle, B. (1990). Graph rewriting: An algebraic and logic approach. In J. Leeuwen, *Handbook of theoretical computer science* (pp. 193-242). Elsevier Science Publishers.

Cowan, J., & Tobin, R. (2004). *XML information set (second edition)*. W3C, Recommendation.

DeRose, S., Maier, E., & Orchard, D. (Eds.) (2001). *XML linking language (XLink) version 1.0*. W3C, Recommendation.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A., & Suciu, D. (1998). *XML-QL: A query language for XML*. W3C.

Fernandez, M., Malhotra, A., Marsh, J., Nagy, M., & Walsh, N. (2004). *XQuery 1.0 and XPath 2.0 data model* (working draft). W3C.

Flum, J., Frick, M., & Grohe, M. (2002). Query evaluation via tree-decompositions. *Journal of the ACM, 49*(6).

Furche, T., et al. (2004). Survey over existing query and transformation languages. Deliverable I4-D1, REWERSE. *http://rewerse.net/I4/deliverables#D1*

Grohe, M., & Schweikardt, N. (2003). Comparing the succinctness of monadic query languages over finite trees. *Proceedings of the Workshop on Computer Science Logic (CSL)*.

Gottlob, G., Leone, N., & Scarcello, F. (2002). Hypertree decompositions and

tractable queries. *Journal of Computer and System Sciences, 64*(3).

Gottlob, G., Koch, C., & Pichler, R. (2003). The complexity of XPath query evaluation. *Proceedings of the ACM Symposium on Principles of Database Systems*.

Horrocks, I., et al. (2004). *SWRL: A semantic Web rule language—Combining OWL and ruleML*. W3C, Member submission.

ISO/IEC 13250 Topic Maps (1999). International organization for standardization, international standard.

Karvounarakis, G., et al. (2004). RQL: A functional query language for RDF. In P. Gray, P. King, & A. Poulovassilis (Eds.), *The functional approach to data management* (pp. 435-465). SpringerVerlag.

Klyne, G., Carroll, J.J., & McBride, B. (2004). *Resource description framework (RDF): Concepts and abstract syntax*. W3C, Recommendation.

May, W. (2004). XPath-logic and XPathLog: A logic-programming style XML data manipulation language. *Theory and Practice of Logic Programming, 3*(4),499-526.

Olken, F., & McCarthy, J. (1998). Requirements and desiderata for an XML query language. *W3C QL'98 – Query Languages 1998*.

Palmer, S.B. (2003). Pondering RDF path. *http://infomesh.net/2003/rdfpath/*

Patel-Schneider, P., & Simeon, J. (2002). The Yin/Yang Web: XML syntax and RDF semantics. *Proceedings of the International World Wide Web Conference*.

Pepper, S., & Moore, G. (Eds.) (2001). *XML topic maps (XTM) 1.0*. TopicMaps.org, Specification.

Prud'hommeaux, E. (Ed.) (2004a). RDF data access working group charter. *W3C*. Retrieved from *http://www.w3.org/2003/12/swa/dawg-charter*

Prud'hommeaux, E. (2004b). Algae RDF query language. *W3C*. Retrieved from *http://www.w3.org/2004/05/06Algae/*

Prud'hommeaux, E., & Seaborne, A. (2004). *SPARQL query language for RDF*. W3C, Working Draft. Retrieved from *http://www.w3.org/TR/rdf-sparql-query/*

Robie, J., et al. (2001). The syntactic Web: Syntax and semantics on the Web. *Markup Languages: Theory and Practice*, *3*(4), 411-440.

Schaffert, S., & Bry, F. (2004). Querying the Web reconsidered: A practical introduction to Xcerpt. *Proceedings of the Extreme Markup Languages*.

Seaborne, A. (2004). *RDQL—A query language for RDF*. W3C, Member Submission. *http://www.w3.org/Submission/2004/SUBM-RDQ-20040109/*

Segoufin, L. (2003). Typing and querying XML documents: Some complexity bounds. *Proceedings of the ACM Symposium on Principles of Database Systems*.

Sintek, M. (2002). TRIPLE—A query, inference, and transformation language for the semantic Web. *Proceedings of the International Semantic Web Conference*.

Steer, D. (2003). TreeHugger 1.0 intro-

duction. Retrieved from *http://www.semanticplanet.com/2003/08/rdft/spec*

Vardi, M.Y. (1982). The complexity of relational query languages. *Proceedings of the ACM Symposium on Theory of Computing (STOC)*.

Wilk, A., & Drabent, W. (2003). On types for XML query language Xcerpt. *Proceedings of the Workshop on Principles and Practice of Semantic Web Reasoning*.

Zloof, M. (1975). Query by example. *Proceedings of the AFIPS National Computer Conference*.

ENDNOTES

[1] This has been pointed out, in a slight variation, by Dana Florescu on the XML-DEV mailing list. *http://lists.xml.org/archives/xml-dev/200412/msg00228.html*

Please provide bios