I4-D2

Identification of Design Principles

Project number:	IST-2004-506779
Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Document type:	D (deliverable)
Nature of document	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Munich/I4-D2/D/PU/a1
Responsible editor(s):	Tim Furche
Reviewer(s):	Claude Kirchner, Wolfgang May
Contributing participants:	Bucharest, Hannover, Heraklion, Manchester, Nancy, Vienna, Venice
Contributing workpackages:	I4
Contractual date of delivery:	31 August 2004

Abstract
This report identifies those design principles for a (possibly new) query and transformation language for the Web supporting inference that are considered essential. Based upon these design principles an initial strawman is selected. Scenarios for querying the Semantic Web illustrate the design principles and their reflection in the initial strawman, i.e., a first draft of the query language to be designed and implemented by the REWERSE working group I4.

Keyword List
reasoning, query language, design principles

Identification of Design Principles

François Bry[1], **Tim Furche**[2], **Liviu Badea**[3], **Christoph Koch**[4], **Sebastian Schaffert**[5], **Sacha Berger**[6]

[1] Institute for Informatics, University of Munich, Germany
Email: Francois.Bry@ifi.lmu.de

[2] Institute for Informatics, University of Munich, Germany
Email: Tim.Furche@ifi.lmu.de

[3] National Institute for Research and Development in Informatics, Bucharest, Romania
Email: badea@ici.ro

[4] Computer Science Department, Vienna University of Technology, Austria
Email: koch@dbai.tuwien.ac.at

[5] Institute for Informatics, University of Munich, Germany
Email: Sebastian.Schaffert@ifi.lmu.de

[6] Institute for Informatics, University of Munich, Germany
Email: Sacha.Berger@ifi.lmu.de

15 August 2004

Abstract
This report identifies those design principles for a (possibly new) query and transformation language for the Web supporting inference that are considered essential. Based upon these design principles an initial strawman is selected. Scenarios for querying the Semantic Web illustrate the design principles and their reflection in the initial strawman, i.e., a first draft of the query language to be designed and implemented by the REWERSE working group I4.

Keyword List
reasoning, query language, design principles

Contents

1 Introduction

The "Semantic Web" is an endeavor which Tim Berners-Lee, the father of HTML and of HTTP, James Hendler, and Ora Lassila initiated in 2001 with an article in the Scientific American [6]. The "Semantic Web" vision is that of the current Web which consists of (X)HTML and documents in other XML formats being extended with meta-data specifying the meaning of these documents in forms usable by both, human beings and computers:

> The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. [6]

One might see the Semantic Web meta-data added to today's Web as semantic indices similar to encyclopedias. A considerable advantage over conventional encyclopedias printed on paper is that the relationships expressed by Semantic Web meta-data can be followed by computers, very much like hyperlinks can be followed by programs, and be used for drawing conclusion 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. [6]

A number of formalisms have been proposed in recent years for representing Semantic Web meta-data, e.g., RDF [28], Topic Maps [25], and OWL [2]. Whereas RDF and Topic Maps provide merely a syntax for representing assertions on relationships like "a text is authored by some person", schema or ontology languages such as RDFS [9] and DAML+OIL [24] allow 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 schemata (as detailed in the architectural road map for the Semantic Web [5]), rules expressed in, e.g., SWRL [23] or RuleML [8], 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, or any other XML format. Considering the large amount and the distributed storage of data available already 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 a reasonably high-level, declarative query language is needed for such efficient and convenient access, as it allows to separate the actual data storage from the view of the data a query programmer operates on.

In this paper, design principles are presented for guiding the development of a query language that allows access to both standard and Semantic Web data. Three principles are at the core of this proposal:

- As discussed above, **the same query language should provide convenient and efficient access to any kind of data expected to be found in the Semantic Web**, e.g., to documents written in (X)HTML as well as to RDF descriptions of these documents and even to ontologies providing terms for the RDF descriptions. Only by intertwining data from all the different layers of the Semantic Web, as described in [5], that vision can be realized in its full potential.

- Convenience for the user of the query language requires the re-use of knowledge obtained in another context. Therefore, **the query language should be based upon rules and patterns** that together allow (1) for **querying existing and constructing new data by a form-filling approach** (similar to, but arguably more expressive than, the query-by-example paradigm [37]) and (2) for **basic reasoning capabilities** including the provision of different views of the same data, even represented in different Web formalisms.

- Building upon the basic reasoning abilities provided by the query language, **it should be possible to add more advanced reasoning through theory extensibility**. This can be motivated by the desire for a more efficient or a more expressive reasoner. This is not only more convenient than separate languages for querying and reasoning, but also allows for a more precise specification of queries and is necessary when considering, e.g., querying ontologies.

It is worth noting, that the above stated core principles and the more detailed discussion of the design principles in Section 2 are describing general principles of the query language, 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 included in these design principles. Rather, the language is defined independently of such issues, but allows for further extensions or restrictions of the language, if necessary for a particular setting or application.

The rest of this paper is organized as follows: In Section 2 the design principles are proposed and justified in detail. This is followed by a short motivation on the selection of the initial strawman (Section 3) and two use cases that provide illustration for the some of the design principles as well as for Xcerpt [33], the language selected as initial strawman. The report is concluded by a brief outlook on future work by the REWERSE working group I4.

2 Design Principles

2.1 A Query Language for both, the Standard Web and the Semantic Web

A thesis underlying the work of the REWERSE I4 working group is that a *common query language for both conventional Web and Semantic Web applications* is desirable (this requirement for a Web query language has also been expressed in [29]). There are two reasons for this thesis:

First, in many cases, data are 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 (formalizing expressions such as “see”, “see also”, “use instead” commonly used in traditional encyclopedia) are marked up, e.g., using XLink or any other ad hoc or generic formalism as one might expect from an online encyclopedia, then the encyclopedia can also be used as “Semantic Web data”, i.e. as meta-data, 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, most likely, Semantic Web applications will combine and intertwine queries to Web data and to meta-data (or Semantic Web data) in all possible manners. There is no reason to assume that Semantic Web applications will rely only on meta-data or that querying of conventional Web data and Semantic Web data will take place in two (or a few) successive querying phases referring each to data of one single kind, "conventional Web data" or "Semantic Web data". 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), (c) mixed conventional and Semantic Web queries (for adding hypertext links from words defined in the (merged) encyclopedia).

2.2 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 (cf. [16] and the XQuery data model [20]), it does provide non-hierarchical relations, e.g., by using ID/IDREF links or via XLink [18].

Similar to the proposal for an integrated data model and (model-theoretic) Semantics of XML and RDF presented in [30], 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, another 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 most simple Semantic Web use cases, in particular, when considering inference based not only on relations of a single type. Therefore, a (rooted) graph data model is needed.

Indeed, there is one particularity of XML that makes its uniform treatment with data such as RDF and Topic Maps difficult: siblings are always considered ordered in XML. This must be reflected in the query language by allowing both ordered, where the order among siblings is relevant, and unordered queries, where siblings specified in the query can occur in the data in any order.

2.3 Referential Transparency and Answer-Closedness

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": while declarativity is an intuitive notion which possibly cannot be precisely defined, referential transparency is a clearly defined notion. Referentially transparent programs are easier to understand and therefore easier to develop, to maintain, and to optimize. Referential transparency surely is one of the essential properties a query language for the Web should satisfy.

Answer-closedness. Let us call "answer-closed" a query language such that replacing a subquery 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 is not required to—be answer-closed. Logic programming languages are answer-closed but SQL is not. E.g., the answer `person(a)` to the Prolog 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 [37] 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, i.e., the expected answer, and the query specifying these data.

2.4 Answers as Arbitrary XML Data, Answer Ranking and Top-k Answers

Answers as Arbitrary XML Data. XML is the lingua franca of data interchange on the Web. As a consequence, answers should be expressible as every possible XML application. This includes both text without mark-up and freely chosen mark-up and structure. This requirement is obvious and widely accepted for conventional Web query languages. Semantic Web query languages, too, should be capable of delivering answers in every possible XML application so as to make it possible, e.g., to mediate between RDF and XTM (an XML serialization of Topic Maps, cf. [31]) data or to translate RDF data after one RDF syntax into another RDF format.

Answer Ranking and Top-k Answers. In contrast to queries posed to most databases, queries posed on 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, for efficiency reasons, (b) 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).

2.5 Pattern Queries

Patterns provide an expressive and yet easy-to-use mechanism for specifying the characteristics of data sought for. In contrast to path expressions, 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, answers like form fillings.

2.6 Incomplete Query Specifications

Incomplete queries specifying only part of data to retrieve, e.g. only some of the children of an XML element (referring to the tree representation of XML data called "incompleteness in breath") or an element at unspecified nesting depth (referring to the tree representation of XML data called "incompleteness in depth") are as 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 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 are in general easier to compare in terms of only some salient features. Second, the merging of "Semantic Web data", e.g., of Topic Maps, is often done in terms of components common to distinct data items. Third, most Semantic Web data standards such as RDF have data items with optional components. In addition, query languages for the conventional and Semantic Web should ease retrieving only part of (completely or incompletely specified) data items.

2.7 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 children 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 email address but specifying that the email 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, email addresses, and if available cellular phone numbers.

But not only for XML data, the limitation of an answer to "interesting" parts of the selected data is helpful. One 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, that helps understanding or identifying it. In this case, one might want, e.g., 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.

2.8 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 mean for achieving the so-called "separation of concern" 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 to a common format data from different sources. Data mediation is needed both on today's Web and on the emerging Semantic Web because of their heterogeneity. Data mediation is especially needed on the Semantic Web because of the many (about 20) standards for RDF data.

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. Indeed, (1) backward chaining gives rise to propagating restrictions from the goals (or main queries) to (sub)queries posed to Web sites and (2) forward chaining could lead to the evaluation of rule bodies on all possible Web sites—in most cases an impossible endeavor.

Recursion. On the Web recursion is needed

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

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

2.9 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.
- Part 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 thesis of the REWERSE I4 working group that both approaches to queries or programs are hard to read (and, therefore, to write and to 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, i.e. the rule's head, to a rule's query, i.e. the rule's body, (logic programming) variables can be employed.

As discussed in Section 2.13, 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.

2.10 Specific Reasoning as Theories

Many practical applications require special forms of reasoning. E.g. for efficiently reasons, equality reasoning is often performed using the so-called paramodulation rule instead of the equality axioms (transitivity, substitution, and symmetry), temporal data might require conversions between different time zones and/or calendar systems that are expressed in a simpler manner and more efficiently performed using arithmetic instead of logical axioms, or 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 run time system of the query language, additional language constructs giving rise to use 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, e.g., the XML syntax of the query language (cf. 2.12) such rule bases can then be queried using the query language itself and maintained and updated by the reactive system developed in the REWERSE working group I4.

2.11 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 a 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, meta-data describing such Web data, e.g., resource descriptions in RDF stating author, usage restrictions, relations to other resources, reviews, etc., and the ontology that provides the concepts and their relations for the resource description in RDF.

2.12 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 are provided: an XML syntax which is purely term-oriented and another one which combines term expressions with non-term expressions like most programming languages. This other syntax is more compact than the XML syntax and better readable for human beings.

Both syntaxes are interchangeable (translation is a low cost process).

Third, a visual syntax as a mere Hypertext rendering of a textual query language can greatly increase the accessibility of the language, in particular for non-experts. This approach to developing a visual language is novel and has several advantages. It results in a visual language tightly connected to a 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 an Hypertext application is especially accessible for Web and Semantic Web application developers.

2.13 Polynomial Core

Most database query languages are engineered languages designed to support frequently-used queries while being restrictive enough to allow for certain reasoning problems (such as query minimization) over queries to be tractable or at least computable, and other properties such as guaranteed termination of queries given appropriate evaluation algorithms to be trivially satisfied.

In the context of REWERSE, the language requirements discussed in this document have led us to consider a general-purpose database programming language. Our language is probably Turing-complete, and queries do not necessarily terminate. (Moreover, it is undecidable whether in general a query terminates on all input databases.) However, it is essential that for the most frequently used queries, small upper bounds on the resources taken to evaluate queries (such as main memory and query evaluation time) can be guaranteed. For that reason, it is desirable to identify a fragment of our language for which termination can be guaranteed and that 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.

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

One complexity class which is usually identified with efficiently solvable problems (or queries) is that of all problems solvable in polynomial time. PTIME queries can still be rather inefficient on large databases, so another, even more desirable class of queries would 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, already conjunctive relational queries are NP-complete with respect to query complexity. Conjunctive queries can only 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” or “bounded hypertree-width”, 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 (such as 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 (but have non-elementary query complexity). Reasoning on the Semantic Web naturally happens on graph data, and results for trees remain relevant because many graphs are trees (e.g., in Xcerpt, terms without references). However, the linear time results already fail if very simple comparisons of data values in the trees are permitted (which are among our basic requirements for a query language for the Semantic Web).

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 of recursion. Only the latter is relevant in the context of the I4 query language language. A PTIME upper-bound on the data complexity of recursive query languages is achieved by either disallowing recursion or imposing an appropriate monotonicity requirement (such as those which form the basis of PTIME data complexity in standard Datalog or Datalog with inflationary fixpoint Semantics). The main source of difficulty in achieving this is the complex structure of data terms that can be computed in the I4 query language, which can be arbitrarily deep.

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 is usually also best done on restricted query language fragments, in particular if such fragments exhibit alternative algebraic, logical, or game-theoretic characterizations.

Above, two evaluation strategies for dealing with recursion were discussed (forward chaining vs. backward chaining). The efficient evaluation of a recursive language such as Xcerpt could also profit greatly from sideways binding-passing techniques in the spirit of the magic sets technique for efficiently evaluating recursive Datalog. These in turn are only feasible for a language fragment in which all queries are guaranteed to terminate.

2.14 Modeling, Visualizing, and Verbalizing: Integration with I1

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. In cooperation with the REWERSE working group I1 on “Rule Mark-up Languages”, the modeling, visualization, and verbalization of queries should be investigated.

For verbalizing queries, as well as their in- 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 wide-spread success of Web search engines.

As discussed above, a visualization based on styling of queries is highly advantageous in a Semantic Web setting. As demonstrated in [4], it can also serve as a foundation for interactive features such as authoring of queries. On this foundation, more advanced authoring tools, e.g., for verification and validation of queries, can be implemented.

Finally, a strict XML syntax allows an easy implementation of automatic generation and manipulation tools, e.g., for model-driven query design or source-to-source transformations

for optimization or validation.

2.15 Typing and Composing Queries: Integration with I3 and A1

Typing of queries provides not only for earlier and often easier detection of errors, but can also be the basis of advanced optimization techniques for query evaluation. In the Semantic Web, typing is often based on ontologies specified in schema languages such as RDFS or OWL. Leveraging such schema information for typing of queries and querying data by their type allows the expression of interesting queries as demonstrated, e.g., in [26], and the detection of inconsistencies.

Although, typing and type systems can be considered as separate to the specification of the core query language, the usefulness of querying data based on its type is obvious, in particular for the Semantic Web, where ontologies provide type information. Such querying abilities have to be aligned with the general principles of the query language detailed here.

In cooperation with the REWERSE working group I3 type languages and systems for Web and Semantic Web query language [36, 10], together with the REWERSE working group A2 extensions of such type systems for temporal, calendar, and location data [12, 14, 15] are investigated.

2.16 Querying and Evolution: Integration with I5

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 is 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., in 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. Following the REWERSE vision, we believe that the specification of both queries on occurring events (the "event" part of ECA-rules) and on the condition of the data, that should hold for a specific action to be performed, should be closely related or even embedding the general purpose query language whose principles are discussed in this report.

In cooperation with the REWERSE working group I5, the integration of a query language and a reactive language in the context of the (Semantic) Web is investigated [11, 1].

3 Choice of an Initial Strawman

As the comparison of Web and Semantic Web query languages given in the REWERSE deliverable I4-D1 [21] shows, none of the already proposed query languages clearly fulfill all the design principles mentioned above. The survey shows that the vast majority of current proposals for Web query languages only provides limited reasoning abilities and where such abilities are

provided they tend to be hard to use. This is despite the recognition that such abilities are crucial for handling the requirements of Semantic Web querying. The classification scheme derived from the results in [21] hints at a possible explanation: So far, most query languages proposed for the (Semantic) Web have been concentrated on providing a rather low-level, but easy to understand and implement query language.

Arguably, Xcerpt [11, 33] is the query language closest to fulfill these principles:

- Xcerpt is a query language conceived for both, the (standard) Web and the Semantic Web, as discussed in [11],
- Xcerpt is referentially transparent and answer-closed,
- Answer to Xcerpt queries can be XML data of any kind and top-k answers can be expressed in Xcerpt,
- Xcerpt queries are pattern-queries and cover a wide range of incompleteness relevant for many practical applications,
- Xcerpt gives rise to specify partial answers,
- Xcerpt is rule-based, offers both, chaining and recursion.

Other languages (e.g., [34, 7, 22] surveyed in [21]) that provide similar reasoning capabilities are either limited to, e.g., querying XML or RDF, or fail to express incomplete queries as required in Section 2.6.

Admittedly, specific reasoning is not yet supported by Xcerpt in its current stage of development and implementation. However, first results towards extensions of this kind are already available [14, 12]. Also a polynomial core for Xcerpt has not yet been identified clearly.

It is also worth mentioning that expertise with Xcerpt lies within a REWERSE participant highly involved in I4, and that participant also has control over Xcerpt's development. Furthermore, a couple of REWERSE activities, especially on reactivity and change on the Web and Semantic Web (in the working group I5) and on types (in the working group I3) build up upon or are closely related to Xcerpt.

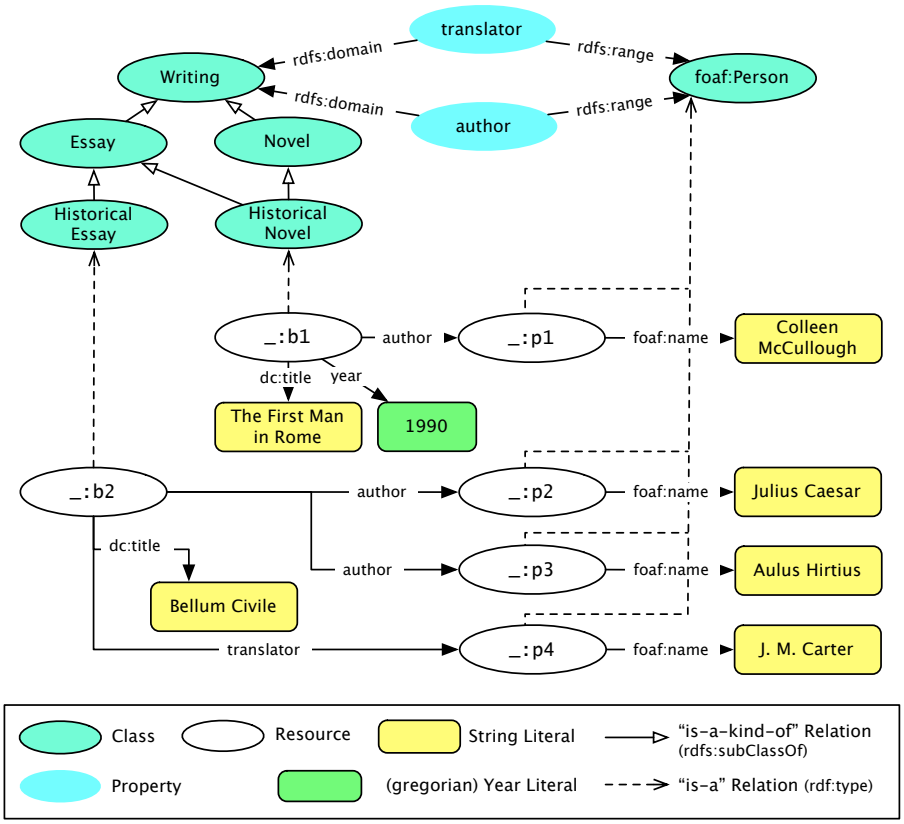
No other candidate initial query language for *both* the Web and Semantic Web strawman seems to be available within REWERSE, especially within the working group I4.

Therefore, Xcerpt seems to be a reasonable choice for as an initial strawman for the working group I4.

4 Scenarios

The following two use cases illustrate some of the design principles discussed in Section 2 and give some intuition on how to use Xcerpt for querying both standard and Semantic Web data. The first scenario illustrates a number of issues involved in querying Semantic Web data even for a rather simple setting. The setting demonstrates how querying existing data (e.g., about books) can be improved if some additional semantic information, here in the form of an ontology, is available and considered during query processing. The second scenario is more complex as it introduces an application where conventional Web (HTML pages) and Semantic Web data is intertwined. It is also demonstrated that semantic information can not only be used to improve the quality of the answers but might also help in the actual query processing (e.g., by allowing a more informed query planning).

Figure 1 RDF Graph for Example 1



4.1 Ontology-driven Literature Retrieval

The following examples illustrate based on a small set of RDF data some of the peculiarities and pitfalls of a Semantic Web query language and how these can be handled in Xcerpt.

Example 1. The data term shows a small excerpt from a book database together with a sample ontology over novels and other literary works. Some of the concepts used are drawn from the "Friend of a Friend" (foaf) project[1]. The rest of this report uses prefixes to abbreviate the URLs for RDF, RDFS and OWL properties.

[1] `http://www.foaf-project.org/`

At site http://bookdealer.com:

```
RDF {
  Historical_Novel {
    author {
      foaf:Person {
        foaf:name{"Colleen McCullough"}
      }
    },
    dc:title{"The First Man in Rome"}
  }
  Historical_Essay {
    author {
      foaf:Person {
        foaf:name { "Julius Caesar" }
      },
      foaf:Person {
        foaf:name { "Aulus Hirtius" }
      }
    },
    dc:title { "Bellum Civile" },
    translator {
      foaf:Person {
        foaf:name { "J. M. Carter" }
    } } }
  &author @ rdf:Property {
    rdfs:domain {
      ^&writing
    }
    rdfs:range {
      ^&foaf:Person
    }
  }
  &translator @ rdf:Property {
    rdfs:domain {
      ^&writing
    }
    rdfs:range {
      ^&foaf:Person
    }
  }
  &historical_novel @ rdfs:Class {
    rdfs:label { "Historical_Novel" },
    rdfs:subClassOf {
      &novel @ rdfs:Class {
        rdfs:label { "Novel" },
        rdfs:subClassOf {
          &writing @ rdfs:Class {
            rdfs:label { "Writing" }
          } } }
    }
    rdfs:subClassOf {
      &historical_essay @ rdfs:Class {
        rdfs:label { "Historical_Essay" }
        rdfs:subClassOf {
          &essay @ rdfs:Class {
            rdfs:label { "Essay" }
            rdfs:subClassOf {
              ^&writing
          } } } }
    }
  }
}
```

This data term represents the RDF graph shown in Figure 1: There are two books in the data, the first one is classified (via `rdf:type`) as a *Historical Novel*, a term defined in the sample ontology. Furthermore, it has an `author` that is a `foaf:Person` with `foaf:name` "Colleen McCullough". The second one also has a `translator` and several `author`s. The sample ontology is basically a conceptual hierarchy for a (small subset of) terms used to classify books and other literary works. The terms are related by `rdfs:subClassOf`, indicating that, e.g., a *Historical Novel* is a kind of *Novel* that, in turn, is a kind of *Writing*. Note the Xcerpt notation `id @ ...` (`^id`, resp.) for representing `ID` (`IDREF`, resp.) attributes.

For reasons of brevity and readability, a representation of the RDF graph as an Xcerpt data term is used that is very close to the syntactic representation of RDF graphs in XML [3]. In this respect, our approach is similar to [32] for querying RDF data with XQuery. However, as is illustrated in the following, there are several peculiarities of Semantic Web data that most query languages for the conventional Web do not support easily.

Properties are optional and multi-valued. In RDF, relations such as `author` or `rdf:type` between objects (also referred to as resources) are called properties. In contrast to traditional knowledge representation techniques, such as frames, all properties in RDF are optional and multi-valued: it is not possible to formally restrict the number of properties of the same type between two resources. E.g., a `Writing` may have no translator, one translator, or any number of translators.

In Xcerpt, optional and multi-valued properties can easily be retrieved by using `all` and `optional`, as shown in Example 2.

Example 2. Retrieve all writings from `http://bookdealer.com` together with their title only if they have a title. Also return any authors or translators for each book, if there are any. `subClassOf[ var BookType, "Writing"]` expresses that the type of the resource must be a subclass of `Writing` (cf. Example 4).

```
CONSTRUCT
  RDF {
     var BookType {
       optional dc:title{  var Title },
       all optional author{  var Author },
       all optional translator{  var Translator }
     }
  }
FROM
  and {
     in {
       resource { "http://bookdealer.com" },
       RDF {{
         desc  var BookType {
           optional author{  var Author },
           optional dc:title{  var Title },
           optional translator{  var Translator }
         }
       }}
     },
     subClassOf[ rdfs:Class {{ rdfs:label{ var BookType} }},
                 rdfs:Class {{ rdfs:label{"Writing"} }}     ]
  }
END
```

Inference. One of the most fundamental promises of the Semantic Web is that, given a machine-processable Semantics for data, new data can be inferred from existing one automatically.

Example 3. All persons that have published together can be retrieved by the following program. The reflexivity of the co-author relation is expressed using unordered subterm specification (i.e. curly braces), as in `co-author{ var X,  var Y}`.

```
CONSTRUCT
  co-author{ var X,  var Y}
FROM
  and {
     in {
       resource { "http://bookdealer.com" },
       RDF {{
          var BookType {{
           author {  var X },
           author {  var Y },
         }}
       }}
     },
     subClassOf[ rdfs:Class {{ rdfs:label{ var BookType} }},
                 rdfs:Class {{ rdfs:label{"Writing"} }}     ]
  }
END
```

More interesting is the kind of inference that arises from traversing the structure of the RDF graph recursively. This is required, e.g., to compute the closure of transitive relations. RDF Schema defines two such transitive relations `rdfs:subClassOf` and `rdfs:subPropertyOf`, OWL allows the definition of additional transitive properties by classifying them as subclasses of `owl:TransitiveProperty`.

Support for RDF Schema. RDF Schema extends RDF with a set of predefined properties and specifies a (partial) Semantics for these properties. Most notably, means for defining a subsumption hierarchy for concepts and properties are provided by `rdfs:subClassOf` and `rdfs:subPropertyOf`. These properties are transitive. E.g., if a query asks for all `Writings`, also resources that are classified as `Novels` or `Historical_Essays` should be returned (under the sample ontology specified above).

Example 4. The Semantics of, e.g., `rdf:subClassOf` can be easily implemented in Xcerpt as demonstrated by the following program. The transitive closure of `rdf:subClassOf` is computed using recursion.

```
CONSTRUCT
  subClassOf[  var Subclass,  var Superclass ]
FROM
  or { RDF {{
         desc  var Subclass ⇝ rdfs:Class {{
           rdfs:subClassOf {
              var Superclass ⇝ rdfs:Class {{ }}
           }
         }}
       }},
       and [ RDF {{
               desc  var Subclass ⇝ rdfs:Class {{
                 rdfs:subClassOf {
                    var Z ⇝ rdfs:Class {{ }}
                 }
               }}
             }},
             subClassOf[  var Z,  var Superclass ]    ]
    }
END
```

Other predefined relations from RDF (Schema) such as `rdf:type`, `rdfs:domain`, or `rdfs:range` can be implemented in a similar manner.

4.2 Distributed Hardware Configuration using Source Information

Motivation. Taking advantage of the huge amount of knowledge implicit and distributed on the Web is a significant challenge. The main obstacle is due to the fact that most Web pages were designed for human-centered browsing rather than being machine-processable. In addition to static HTML pages the Web currently offers online access to a large number of information resources, such as databases with a Web interface. But real-life applications frequently require combining the information from several such resources, which may not have been developed with this interoperability requirement in mind. Thus, a large amount of knowledge is implicit, heterogeneously distributed among various resources and thus hard to process automatically.

The recent developments towards a "Semantic Web" should help address these problems. Being able to explicitly represent domain-specific knowledge in the form of ontologies, should allow reasoning about such machine-processable Web pages.

The emergence of standards for data mark-up and interchange such as XML and for representing information about resources and their Semantics (such as RDF and RDF Schema) can be seen as a first step in the transition towards a Semantic Web. However, the vast majority of Web pages still conform to the HTML standard, which only controls their visual aspects rather than their informational content. Extracting the informational content from such pages

which essentially contain free text is a difficult practical problem. The Resource Description Framework (RDF) has been designed to complement such human-oriented text with machine-processable annotations. A large number of prototype systems able to read and reason about such annotations have been developed (TRIPLE [34], Metalog [27], SiLRI [17], Ontobroker [19]). However, currently only a very small minority of Web pages have RDF annotations. Moreover, existing annotations tend to refer to basic features such as document author, creation date, etc., but do *not* duplicate the information content of the page.

On the other hand, a large number of information sources have a Web interface and could be the building blocks of complex applications, were it not for the unavoidable Semantic mismatch between such resources developed by different people. Such Web interfaces produce pages with a partially stable structure, so that their content can be *automatically extracted* using wrappers, thus replacing human annotation (which is a significant bottleneck in practice).

Dealing with information sources rather than a fixed set of Web pages may pose additional problems. For example, systems like TRIPLE read all the relevant (RDF) annotations before reasoning about them. In the case of large data sources however, it is obviously impossible to retrieve the entire content of such sources before starting reasoning. Also, if additional knowledge is available about the sources, some source accesses may be avoided altogether. Therefore, dealing with information sources requires a certain form of **query planning**, i.e. the ability of constructing and reasoning about alternative sequences of source accesses (plans) before actually querying these sources. Also, *streaming* the query responses may allow starting processing before the entire content of the information source is retrieved.

The most significant problem faced when trying to combine several resources is related to their heterogeneity. This heterogeneity can be either structural (different schemas), or Semantic (the same entity can be represented using different terms from different vocabularies). Integrating such resources can be achieved by mapping them (both their schemas and their content) to a common knowledge level, at which their interoperation is straight-forward. This common level involves not just a common (domain-specific) vocabulary, but also formal (machine processable) descriptions of the terms of this vocabulary, as well as the relationships between them, which form a so-called ontology. A mediator architecture [35] can be used for query answering.

The present use case deals with such more complex Semantic Web scenarios, involving the integration of heterogeneous resources using rules and ontologies.

Description of the Scenario. We consider a hardware configuration problem as a typical user scenario involving a mediated view over a set of information sources on the Web. A component integrator selling customized computer configurations may use components from several component providers, while trying to satisfy a set of compatibility and user constraints.

This problem has many typical features of the Semantic Web:

- it involves distributed and dynamically changing information sources (the component catalogs of the different providers available via Web interfaces)
- the information is Semantically heterogeneous, requiring a domain ontology for a uniform access
- the domain involves complex compatibility constraints between components, requiring constraint propagation during query planning

- there are several alternative sources for some given component, which makes query planning necessary.

Here we consider just fragments of this hardware configuration domain, in order to emphasize the main design principles of the query language.

Assume we have two main component providers, or vendors, flamingo[2] and oktal[3]. A user query may ask for a system with an upper bound on the total price. Here we just concentrate on a "simple" system containing just a motherboard and a processor. Of course, the components must satisfy compatibility constraints (e.g. one should not attempt to use an AMD processor on a motherboard solely designed for an Intel Pentium).

Information extraction. Since most of the existing component catalogs are accessible via the Web in HTML format, some of the sources providing also XML output, the query language should be able to deal with both Standard and Semantic Web. For example, the following rule extracts the prices of motherboards from an HTML page provided by flamingo. Note that unlike in the case of XML data, HTML tags are used mainly for formatting purposes and regular expressions are normally needed for extracting the relevant information (in this case the price).

```
CONSTRUCT
  model[var Model, var Price]
FROM
  in { resource  { "fl.html", "html" },
    html{{ head [[ title [ var Model ] ]],
      body [[ desc table  [[desc div{{/.*PRET: var Price -> [0-9.]+/}} ]]
    ]] }}
  }
END
```

This simple query (in the FROM part) above also emphasizes the need for incomplete queries, as in most practical cases we can have both incompleteness in breadth (the double brackets and braces in the patterns above only constrain a subset of tags/sub-terms) as well as incompleteness in depth (indicated by the desc construct).

Information integration using "mapping rules". Since the sources are heterogeneous, so-called **"mapping rules"** may be useful for describing the content of the sources in terms of a given ontology of hardware components. In the following example, motherboards sold by flamingo are mapped onto the ontology concept "motherboard" (note that here we have a trivial mapping of all slots, with an additional vendor slot recording the name of the vendor):

```
CONSTRUCT
  motherboard [
    brand [var MBr], model [var Mmo], price [var MBr],
    socket [var Mso], supported_CPU [var MCPU],
    FSB [optional var MFSB], max_CPU_speed  [var MSp],
    vendor [ "flamingo" ]
  ]
FROM
  in { resource{ "flamingo_motherboards.xml", "xml" },
    fl_motherboard {{ brand {var MBr}, model {var Mmo}, price {var MBr},
    socket {var Mso}, supported_CPU {var MCPU},
    optional FSB {var MFSB}, max_CPU_speed {var MSp} }}
```

[2] http://www.flamingo.ro/

[3] http://www.oktal.ro/

```
  }
END
```

Note again that the ability of writing incomplete queries (in the FROM part of the rule above) allows us to describe only the tags that are related to the information we are interested in. Additionally, the language should be able to deal with semi-structured data for which some of the tags may be missing occasionally (like the FSB, or front side bus, in the above example).

We have similar mapping rules for oktal motherboards, as well as for processors (available also from both flamingo and oktal).

The mapping rules have to reflect the variability of the structure of objects contained in the information sources. For example, a processor description might have different attributes (tags) for different brands (speed for Intel Pentium and eq_speed in the case of AMD processors where the speed of an equivalent Intel Pentium processor is used).

```
CONSTRUCT
  processor[ var Proc, var Speed ]
FROM
  in { resource { "file:proc_fl.xml", "xml"},
    desc processor {{
      attributes {{id {{var Proc}} }} ,
      brand [ "AMD" ],
      eq_speed[ var Speed ]
    }} }
END
CONSTRUCT
  processor[ var Proc, var Speed ]
FROM
  in { resource { "file:proc_fl.xml", "xml"},
    desc processor {{
      attributes {{id {{var Proc}} }} ,
      brand[ "INTEL" ],
      speed[ var Speed ]
  }} }
END
```

Query planning and source descriptions. Any additional knowledge about the content of the sources may be very useful during query planning for discarding inconsistent plans even before accessing the sources. For example, we may know a lower bound for the price of a flamingo motherboard and we may also know that flamingo distributes only Intel and MSI boards.

Such source descriptions express knowledge or even integrity constraints about the content of the sources. (These descriptions could either be provided by the knowledge engineer or could be automatically retrieved from previous source accesses.) Reasoning with such source description rules frequently requires reasoning with various types of constraints, in specific theories. Our example mainly deals with simple numerical equality and inequality constraints. Using such constraints for query planning could either be done by the query processor itself (e.g., using some form of meta-programming) or based upon a reactive system as investigated in the REWERSE working group I5 on “Evolution and Reactivity”.

Frequently, answering queries requires knowledge about the sources that cannot be inferred from their content alone. For example, oktal offers discounts for purchases over a given threshold, while flamingo does not:

```
vendor[ name [ oktal ], discount [ 0.1 ], discount_threshold [ 200 ] ]
vendor[ name [ flamingo ], discount [ 0 ], discount_threshold [ 0 ]  ]
```

This also illustrates the need to view data and meta-data in a uniform way. The domain ontology encodes our knowledge about the particular application domain, such as the concept hierarchy, descriptions of slots/attributes (type, cardinality constraints, etc.), and any additional description rules involving these concepts. For example, motherboards and processors are sub-concepts of component. These descriptions should be compatible with the query language, since the latter should allow the use of ontology terms in queries.

As an example of an ontology rule, we may describe a simple system as a set of matching components, such as a motherboard and a processor. (Note the compatibility constraint between motherboard and processor: the `id` of a processor must be equal to the value of the `supported_CPU` property of the motherboard for them to be compatible.) This rule is very similar to a database view and allows a modular description of the domain.

```
CONSTRUCT
  system { var Mb, var Pr }
FROM
  and {
    var Mb ⇝ motherboard {{ supported_CPU {var CPU} }},
    var Pr ⇝ processor {{ id {var CPU} }}
  }
END
```

While view-like rules are used during query execution, source description rules will be used only for query planning, as they encode knowledge about the contents of the sources available to the planner before actually accessing the sources. This type of knowledge can be used to optimize source accesses, as shown below.

Example 5. As an example, we consider the following user query which asks for a system with an upper bound of 210 on the (possibly discounted) price:

```
GOAL
  system_price{ var S, var S_price }
FROM
  var S ⇝ system {{ motherboard {{ brand { gigabyte } }},
     var Pr -> processor {{ }} }}
  where and{ compute_discounted_priceS, S_price, S_price <= 210 }
END
```

For simplicity, the computation of the discounted price is performed by `compute_discounted_price(S, S_price)`.

Such a query can be answered efficiently by first invoking a query planner, and subsequently executing the resulting plans. As discussed previously, query planning unfolds the query in terms of source calls while propagating constraints with the purpose of discarding the inconsistent plans before the actual source accesses. Note that constraint propagation is interleaved with query unfolding, so that inconsistencies are detected as early as possible.

In our example, motherboards can be acquired either from flamingo or from oktal. However flamingo does not satisfy the gigabyte vendor requirement from the query due to the integrity constraint, so oktal is chosen as the motherboard provider.

Similarly, processors can be acquired also from flamingo or oktal. But flamingo as a processor provider is inconsistent with the price constraints, since no discounts are applicable in this case (flamingo does not offer discounts, while oktal has a higher discount threshold).

The only consistent alternative remains using oktal as a provider of both motherboards and processors, such that now the discount threshold is exceeded and we can take advantage of the 10% discount offered by oktal.

Query planning has therefore retained only the following combination of sources: `okt_processor` and `okt_motherboard`, while discarding the other three possible combinations at planning time, so that only `okt_processor` and `okt_motherboard` are actually queried.

Explicitly describing the capabilities of the sources is essential for their correct invocation. (As opposed to traditional database query languages, such Web sources provide only limited query capabilities.) In our example, flamingo provides a complete list of components or sub-lists associated to specific component types (such as motherboards or processors) depending on parametrized URLs. For instance, `http://www.flamingo.ro/produse.asp?sid=202&cid=2&smt=cgo&lg=ro` retrieves a list of motherboards (the specific parameter value being indicated in bold font). Such source capability descriptions are useful since instead of retrieving all components and then filtering the motherboards, we retrieve only the motherboards.

5 Conclusion and Outlook

In this paper, design principles guiding the work of the REWERSE working group I4 on reasoning-aware querying have been established based on the input of the concurrently performed survey of current standard and Semantic Web query and transformation languages [21].

It has been shown, that Xcerpt, the language selected as initial strawman, already fulfills a number of the design principles, but it is also evident that more research is required to fully understand and develop the vision of an integrated query language for querying standard and Semantic Web data.

To understand further the requirements for such a language, the presented use cases as well as a considerable number of additional ones are currently being investigated. Also, first steps [13] for defining a language Semantics are already underway and will be extended to include a consideration of mixing standard and Semantic Web data.

Acknowledgements. We would like to thank Claude Kirchner and Wolfgang May for reviewing a draft of this deliverable and providing numerous invaluable comments on how to improve both its presentation and content.

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 (cf. `http://rewerse.net`).

References

[1] J. J. Alferes, W. May, and F. Bry. Towards generic query, update, and event languages for the Semantic Web. In *Proc. Workshop on Principles and Practice of Semantic Web Reasoning*, LNCS 3208. Springer-Verlag, 2004. Available from: `http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-15.pdf`.

[2] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. *OWL Web Ontology Language—Reference*, 2004, W3C, Recommendation. Available from: `http://www.w3.org/TR/owl-ref/`.

[3] D. Beckett and B. McBride, editors. *RDF/XML Syntax Specification (Revised)*, 2004, W3C, Recommendation. Available from: http://www.w3.org/TR/rdf-syntax-grammar/.

[4] S. Berger, F. Bry, and S. Schaffert. A Visual Language for Web Querying and Reasoning. In *Proc. Workshop on Principles and Practice of Semantic Web Reasoning*, LNCS 2901. Springer-Verlag, December 2003. Available from: http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2003-6.pdf.

[5] T. Berners-Lee. Semantic Web Road Map (online, retrieved at 2004-08-09). Available from: http://www.w3.org/DesignIssues/Semantic.html.

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

[7] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Simeon, editors. *XQuery 1.0: An XML Query Language*, 2003, W3C, Working Draft. Available from: http://www.w3.org/TR/xquery/.

[8] H. Boley, B. Grosof, M. Sintek, S. Tabet, and G. Wagner. RuleML Design (online). 2002 (retrieved at 2004-08-09). Available from: http://www.ruleml.org/indesign.html.

[9] D. Brickley, R. Guha, and B. McBride, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*, 2004, W3C, Recommendation. Available from: http://www.w3.org/TR/rdf-schema/.

[10] F. Bry, W. Drabent, and J. Maluszynski. On Subtyping of Tree-structured Data—A Polynomial Approach. In *Proc. Workshop on Principles and Practice of Semantic Web Reasoning*, LNCS 3208. Springer-Verlag, 2004. Available from: http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-16.pdf.

[11] F. Bry, T. Furche, P.-L. Patranjan, and S. Schaffert. Data Retrieval and Evolution on the (Semantic) Web: A Deductive Approach. In *Proc. Workshop on Principles and Practice of Semantic Web Reasoning*, LNCS 3208. Springer-Verlag, 2004. Available from: http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-13.pdf.

[12] F. Bry, B. Lorenz, H. J. Ohlbach, and S. Spranger. On Reasoning on Time and Location on the Web. In *Proc. Workshop on Principles and Practice of Semantic Web Reasoning*, LNCS 2901. Springer-Verlag, 2003.

[13] F. Bry, S. Schaffert, and A. Schröder. A contribution to the Semantics of Xcerpt, a Web Query and Transformation Language. In *Proc. Workshop Logische Programmierung*, March 2004. Available from: http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-5.pdf.

[14] F. Bry and S. Spranger. Temporal Constructs for a Web Language. In *Proc. Workshop on Interval Temporal Logics and Duration Calculi*, 2003.

[15] F. Bry and S. Spranger. Towards a Multi-Calendar Temporal Type System for (Semantic) Web Query Languages. In *Proc. Workshop on Principles and Practice of Semantic Web Reasoning*, LNCS 3208. Springer-Verlag, 2004. Available from: http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-16.pdf.

[16] J. Cowan and R. Tobin, editors. *XML Information Set (Second Edition)*, 2004, W3C, Recommendation. Available from: http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

[17] S. Decker, D. Brickley, J. Saarela, and J. Angele. A Query and Inference Service for RDF. In *Proc. W3C QL'98 – Query Languages 1998*, December 1998. Available from: http://www.w3.org/TandS/QL/QL98/pp/queryservice.html.

[18] S. DeRose, E. Maier, and D. Orchard, editors. *XML Linking Language (XLink) Version 1.0*, 2001, W3C, Recommendation. Available from: http://www.w3.org/TR/xlink/.

[19] D. Fensel, J. Angele, S. Decker, M. Erdmann, H.-P. Schnurr, S. Staab, and R. Studer. On2broker: Semantic-Based Access to Information Sources at the WWW. In *Proc. Workshop of Intelligent Information Integration*, 1999.

[20] M. Fernandez, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh, editors. *XQuery 1.0 and XPath 2.0 Data Model*, 2004, W3C, Working Draft. Available from: http://www.w3.org/TR/xpath-datamodel/.

[21] T. Furche, F. Bry, S. Schaffert, R. Orsini, I. Horrocks, and O. Bolzer. Survey over Existing Query and Transformation Languages. Deliverable I4-D1, REWERSE, 2004. Available from: http://rewerse.net/I4/.

[22] L. Garshol. Extending tolog—Proposal for tolog 1.0. In *Proc. Extreme Markup Languages*, 2003. Available from: http://www.ontopia.net/topicmaps/materials/extending-tolog.html.

[23] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language—Combining OWL and RuleML*, 2004, W3C, Member submission. Available from: http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[24] I. Horrocks, F. van Harmelen, and P. Patel-Schneider, editors. *DAML+OIL*, 2001, Joint US/EU ad hoc Agent Markup Language Committee, Revised Language Specification. Available from: http://www.daml.org/2001/03/daml+oil-index.html.

[25] *ISO/IEC 13250 Topic Maps*, International Organization for Standardization, International Standard. Available from: http://www.y12.doe.gov/sgml/sc34/document/0322_files/iso13250-2nd-ed-v2.pdf.

[26] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. RQL: A Functional Query Language for RDF. In P. Gray, P. King, and A. Poulovassilis, editors, *The Functional Approach to Data Management*, chapter 18, pages 435–465. Springer-Verlag, 2004. Available from: http://athena.ics.forth.gr:9090/RDF/publications/FuncBook.pdf.

[27] M. Marchiori and J. Saarela. Query + Metadata + Logic = Metalog. In *Proc. W3C QL'98 – Query Languages 1998*, December 1998. Available from: http://www.w3.org/TandS/QL/QL98/pp/metalog.html.

[28] M. Nilsson, W. Siberski, and J. Tane. Edutella Retrieval Service: Concepts and RDF Syntax (online). June 2004 (retrieved at 2004-04-26). Available from: http://edutella.jxta.org/spec/retrieval.html.

[29] F. Olken and J. McCarthy. Requirements and Desiderata for an XML Query Language. In *Proc. W3C QL'98 – Query Languages 1998*, December 1998.

[30] P. Patel-Schneider and J. Simeon. The Yin/Yang Web: XML Syntax and RDF Semantics. In *Proc. International World Wide Web Conference*, May 2002.

[31] S. Pepper and G. Moore, editors. *XML Topic Maps (XTM) 1.0*, 2001, TopicMaps.org, Specification. Available from: `http://www.topicmaps.org/xtm/index.html`.

[32] J. Robie, L. M. Garshol, S. Newcomb, M. Fuchs, L. Miller, D. Brickley, V. Christophides, and G. Karvounarakis. The Syntactic Web: Syntax and Semantics on the Web. *Markup Languages: Theory and Practice*, 3(4):411–440, 2001. Available from: `http://www.w3.org/XML/2002/08/robie.syntacticweb.html`.

[33] S. Schaffert and F. Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proc. Extreme Markup Languages*, August 2004. Available from: `http://www.pms.informatik.uni-muenchen.de/publikationen/PMS-FB/PMS-FB-2004-7.pdf`.

[34] M. Sintek and S. Decker. TRIPLE—A Query, Inference, and Transformation Language for the Semantic Web. In *Proc. International Semantic Web Conference*, June 2002. Available from: `http://triple.semanticweb.org/`.

[35] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, 1992.

[36] A. Wilk and W. Drabent. On Types for XML Query Language Xcerpt. In *Proc. Workshop on Principles and Practice of Semantic Web Reasoning*, LNCS 2901. Springer-Verlag, 2003.

[37] M. Zoof. Query By Example. In *Proc. AFIPS National Computer Conference*, 1975.