Towards generic query, update, and event languages for the Semantic Web

Wolfgang May José Júlio Alferes François Bry

Institut für Informatik, Universität Göttingen, Germany
may@informatik.uni-goettingen.de
CENTRIA, Universidade Nova de Lisboa, Portugal
jja@di.fct.unl.pt
Institut für Informatik, Ludwig-Maximilians-Universität München, Germany
francois.bry@ifi.lmu.de

Abstract. We outline aspects of querying and updating resources on the Web and on the Semantic Web, including the development of query and update languages in course of the REWERSE project. When considering updates and communication of updates between autonomous sources, reactive behavior plays an important role such that an event language is required. This article provides a systematic outline of the intended research steps towards handling reactivity and evolution on the Web.

1 Introduction

Use of the *Web* today –commonly known as the "World Wide Web"– mostly focuses on the page-oriented perspective: most of the Web consists of browsable HTML pages only. From this point of view, the Web can be seen as a graph that consists of the resources as nodes, and the *hyperlinks* form the edges. Here, queries are stated against individual nodes, or against several nodes, e.g., with formalisms like F-Logic [16] or Lixto [3]; or in case that the sources are provided in XHTML, they can be queried by XQuery, XPathLog [17], or Xcerpt [5]. As such, the Web is mainly seen from its static perspective of autonomous *sources*, whereas the *behavior* of the sources, including active interaction of resources does not play any important role here.

But there is more on the Web of today than HTML pages. Leaving the superficial point of view of HTML pages, the Web can be seen as a set of *data sources*, some of which are still browsing-oriented, but there are also database-like resources that can actually be queried. In fact, besides HTML documents that are accessed as a whole, and in which (query) processing has then to be done separately, there are XML (including XHTML) data sources, described by DTDs or XML Schemas, that in general can be queried (e.g., by XPath/XQuery, XPathLog or Xcerpt). Moreover, specialized information sources (that we call, abstractly, *Web services*) exist that provide answers only to a restricted set of queries that are given abstractly as a set of name-value pairs.

With these representations, the perspective may shift more to the idea of a Web consisting of (a graph of) *information systems*. In these information systems, data extraction may be thought not only in terms of local queries, but also in terms of global queries that are stated against the Web, or against a group (or community) of nodes on the Web. Given the highly heterogeneous and autonomous characteristics of the Web, this requires appropriate query languages, and a way to deal with the integration of data from the various sources. The aspects of query languages for this Web are discussed below, in Section 2.

But such an infrastructure of autonomous sources should allow for more than querying. Consider a set of sources of travel agencies and airline companies. It is important to be capable of querying such a set for, e.g. timetables of flights, availability of flight tickets, etc. But a Web consisting of information systems should allow for more. For example: it should allow for drawing conclusions based on knowledge (e.g. in the form of derivation rules) available on each node; it should allow for making reservations via a travel agency, and automatically make the corresponding airline company (and also other travel agencies) aware of that; it should allow airline companies to change their selling policies, and have travel agencies automatically aware of those changes; etc. The Web, as we see it, with such capabilities can be seen as forming an active, "living" infrastructure of autonomous systems, where reactivity, evolution and its propagation plays a central role. Given these requirements, the knowledge of one such system could be classified into three conceptual levels: facts (e.g. in XML or wrapped in XML); a knowledge base, given by derivation rules (e.g. marked up in RuleML); and behavior/reaction rules, specifying which actions are to be taken upon which event and given what conditions. In this context events may either be internal events (e.g., updates), or external events (e.g., messages), and actions may include (local) updates of the knowledge, remote actions (potentially structured by transactions), synchronization actions, and possibly "blackbox" actions. Since, for being autonomous, sources may rely on different data models with different modelling assumptions and languages, an external description of the interface is also needed.

This view of the Web raises new issues, and requires new languages, the aspects of which will be outlined in the remainder of the paper. Moreover, we provide a systematic outline of the intended research steps towards handling reactivity and evolution on the Web. For reaching this goal, our approach will be to start from existing technologies and extend them step by step, just as demanded by the specific needs encountered. In Section 2 we discuss query languages for the Web. Then, in Sections 3 to 6 we discuss languages for reactivity, evolution and communication on the (non-semantic) Web. Evolution and Reasoning on the Semantic Web are then discussed in Section 7.

2 Web Query Languages

2.1 Today's Web Query Languages

Query languages similar to (but different from) those used and developed for databases are needed on the Web for easing the retrieval of data on the Web, or easing the specification of (materialized or non-materialized) views (in the database sense) and for expressing complex updates, especially intensional updates (i.e., updates of intensional predicates expressed in terms of queries) that must then be translated into updates of the actual sources.

In addition to user-defined queries, queries play an important role also for the Web itself: dynamic Web pages (i.e., whose content is generated at query time) are non-materialized views (i.e., views that are computed at data retrieval time). Here, closedness of the language is obviously necessary: it is defined over Web contents, and its output must be valid Web contents.

Design of Web Query Languages. Early Web query languages developed from query languages for semistructured data, most of them have then been migrated to XML. The design decisions and experiences of these languages have to be considered when designing query and manipulation languages for the Web as a whole. An overview of these can be found in [19]:

Logic programming-style rule-based languages (immediately including an update language) have been presented with e.g. *WSL/MSL (Wrapper/Mediator Specification Language)* [12], and F-Logic [16]. *Lorel* [20] and StruQL [11] followed the SQL/OQL-style clause-based design; StruQL was then developed into XML-QL [10]. Other languages developed from the area of tree matching and transformations, e.g., *UnQL* [7], *XDuce* [13], or *YATL* [9].

The standard XML languages developed from the experience with the above-mentioned languages and from the SGML area. XPath has been established as an addressing language. It is based on path expressions for navigation, extended with filters. It serves as the base for many other W3C languages in the XML world. XQuery extends XPath with SQL-like clause-based constructs `FLWOR: FOR ... LET ... WHERE ... ORDER BY ... RETURN` to a full-fledged query language. Variables are bound in the `FOR` and `LET` clauses to the answer sets of XPath expressions. The `WHERE` clause expresses further conditions, and the `RETURN` clause creates an XML structure as a result. While the selection parts in the `FOR` and `LET` clauses are XPath-based, an XML pattern in XML-QL style is used in the `RETURN` clause where the variables are embedded into literal XML and constructors. Proposals for extending XQuery with update constructs (XUpdate) have been published, e.g., in [22]. As a *transformation* language, XSLT follows a completely different idea for providing information from the Web: Here, transformation rules specify how information is extracted from a data source.

Recent non-Standard XML Query Languages. The Lixto system [3] also uses a graphical interface for querying HTML, where a query is developed by interactively selecting nodes from a browser presentation. Internally, Lixto uses a logic programming language for XML called *Elog*, based on flattening XML data into Datalog. XPathLog [17] combines first-order logic, and XPath expressions extended with variable bindings. It is completely XPath-based both in the rule body (query part) and in the rule read (update part), thereby defining an update semantics for XPath expressions.

Xcerpt [5] is a *pattern-based* language for querying and transforming XML data. Its basic form follows a clean, rule-based design where the query (matching) part in the body is separated from the generation part in the rule head. XML instances are regarded as terms that are matched by a term pattern in the rule body, generating variable bindings. An extension with updates, called *XChange*, is currently designed as a clause-based language.

Comparison of Design Concepts. The existing languages for handling semi-structured data and XML differ in several facets in terms of the concepts they use, e.g., access mechanisms and homogeneity of the query and generation part (patterns, path expressions), underlying data model (graph vs. tree), nature of the underlying theoretical framework (logic-based or denotational semantics), and last but not least clause-based or logic-programming-style syntax and semantics. All languages discussed above are *rule-based* and *declarative*, generating variable bindings by a matching/selection part in the "rule body" and then using these bindings in the "rule head" for generating output or updating the database. This rule-based nature is more or less explicit: F-Logic, MSL/WSL (Tsimmis), XPathLog, and Elog use the ":-" Prolog syntax, whereas UnQL, Lorel, StruQL, XML-QL, XQuery/XUpdate, and Xcerpt cover their rule-like structure in an SQL-like clause syntax. These clausal languages allow for a straightforward extension with update constructs.

2.2 Requirements on Query Languages for the Web

Obviously, a Web query language must allow to include explicitly multiple data sources. For this, queries should also be robust against certain changes in a data source, e.g., splitting a data source over several ones that are linked by XLink/XPointer. The execution model must be modular in the sense that it combines answers contributed by multiple data sources. It must consider that data sources provide different, possibly restricted, functionality (answering arbitrary queries, providing access to bulk data, answering a fixed set of data). Thus, closedness is also an important property: query languages are used to define views that act themselves as virtual data sources, and that must be in the same format as the other sources.

In the context of the Web, end even more of the Semantic Web, querying goes much further can "only" accessing the base data, as can e.g. be done by SQL or XML query languages. Query answering on the Semantic Web in general also requires to use meta-data, e.g., expressed using RDF or OWL, but maybe also thesauri (as used in information retrieval and computational linguistics), and thus must support *reasoning* about the query, data, and metadata. The actual forms of reasoning to be applied with RDF, OWL, thesauri, etc., and in general on the forthcoming Semantic Web are rather unclear today – this is one of the goals of current research.

For providing a base for such research, flexible, and extensible query languages including "lightweight reasoning facilities" appear to be a good choice for a first step towards generic tools for the Semantic Web. For several reasons, we propose to use rule-based languages for querying:

Rules provide a natural modularization, and the rule-based paradigm covers a lot of sub-paradigms for designing a wide range of language types as described above: SQL and XQuery are actually a form of simple rules (for defining views), XSLT is rule-based, and languages like Datalog provide a multitude of semantics for any kinds of reasoning. Coming back to the modularization issue, a simple form of non-monotonic negation in presence of recursion (such as restriction to stratified programs) seems to be sufficient for a deductive query language for the Semantic Web. Additional expressive power can be restricted and extended by choosing suitable semantics – e.g. recursion is needed for computing closure relations (as those of RDF and OWL) and can be easily "bought" with using a fixpoint semantics. In [6] it is shown how the rule-based language Xcerpt can be used for reasoning on Web meta-data such as RDF and OWL data. In it, recursive rules are a convenient way to express common forms of reasoning on the Semantic Web e.g. for traversing several edges relating remote concepts. Such complex semantics of rule languages are well investigated in logic programming.

Additionally, rule-based languages can easily be extended to updates and in general to the specification of dynamics on the Web. Here, research on ECA rules and active databases can be applied and extended.

As another consequence of their modularity, rule-based languages are relatively easy to understand and program.

Modular Design of Language Processors. The use of rule-based languages allows also for a modular design of language processors. These consist then of two components: one for the interpretation of rule heads and bodies, and one for implementing the global semantics for suitably evaluating a set of rules.

The language base will most probably be provided by XML, where the individual languages are distinguished by their syntax, i.e., the namespaces and their element names and attribute names. Thus, a generic rule processor will be able to handle arbitrary rules (not only query rules, but also updates and general reaction rules) and to apply appropriate processing to each type of rules. This way, also new languages can be integrated at any time.

3 Local Behavior: Answering Queries and Being Reactive

The simplest activity on the Web is query answering. From the point of view of the user, querying is a static issue: there is no actual temporal aspect in it (except possibly a delay). Nevertheless, from the point of view of the resources, there comes reactivity into play: when answering queries, the resources must answer in reaction to a query message, and in case that the query is answered in cooperation of several sources, they must also send messages to other resources.

Such cooperation is in general required when considering communities of resources on the Web. Even more, cooperation and communication is already necessary when a resource contains references in any form to another resources. Two types of cooperation for query answering can be distinguished:

Distributed Query Answering of Explicit Queries. Explicit queries in any XML query language can be stated against XML resources that are specified by a DTD or an XML Schema. Such resources can contain references to other XML resources by XLink elements with XPointers. A model for answering queries in the presence of XLink references has been investigated in [18]. The actual evaluation of a query in this model results in (re)active communication of result sets and/or subqueries between resources. This behavior is a simple form of reactivity.

Query Answering by Web Services. Web services can also be used for answering (often only a specific set of) queries. For this, they can either use their local knowledge (e.g., facts and derivation rules), or they can also contact other resources for answering a query (e.g., a travel agency service that uses the schedules of trains and flights from several autonomous sources). In this case, the behavior of the Web service is actually not simply query-answering, but can be seen as a general (re)active behavior, given by an external interface description (e.g., in WSDL) that can provide much more functionality than only query answering. Such Web services are instances of general *reactivity*: there is a message, and the resource performs some actions. Moreover, Web services may have an internal state (e.g., reservation systems) that they update.

For including general reactivity of Web services into the intended reasoning on the Semantic Web, a formal specification of their behavior is required. Here, we distinguish two types of Web services, according to the possibilities of reasoning about their behavior (and updating it): Web services where a formal specification of their behavoir exists –either directly rule-based or by any formal method– can be subject to reasoning, whereas other Web services will be considered as black boxes.

Note that, so far, we have described a mainly non-changing, but *active* Web consisting of nearly isolated passive and locally active Web resources. In this setting, evolution on the Web takes place by: either local changes in data sources via updates (e.g. changing a flight schedule); or local evolution of Web services by reaction to events due to their own, specified, behavior. However, this scenario already raises a number of issues deserving investigation, and will serve as our base case for *reactivity*. From this base case, independent, different dimensions can be derived. Namely: the definition of a global model theory and reasoning about global constraints and consistency aspects based on computing answers to queries; testcases for basic communication of resources via *messages* and *reaction rules*; expressing and reasoning about general non-state-changing reactivity on the Web; extension from queries to updates (where updates require the evaluation of a query e.g. for addressing the actual item to be updated). Such updates on the XML level will provide our simplest case of *dynamics*.

4 Updates and Evolution of the Web

Evolution of the Web is a twofold aspect: on today's Web, evolution means mainly evolution of individual Web sites that are updated locally. In contrast, considering the Web as a "living organism" that *consists* of autonomous data sources, but that will *show* a global "behavior" (as already investigated above for query answering) leads to a notion of evolution of the Web as *cooperative evolution* of its individual resources.

In this context, update languages are needed for several tasks, besides simply updating of the contents of a Web site by its owner, or as a reaction on actions of some user (e.g. when a user books a flight, the reservation of a seat must be stored in the underlying database). The updated site can be either the local database of the respective Web site, but most likely, this is an update to a remote database (e.g. in the above example when the booking is done via a travel agency, and the reservation must be forwarded to an update of the carrier company) – seen as a *view update*. But update languages must also allow for modifying the data and *behavior* of a Web site, e.g. in adaptive Web systems (the user modelling data must be updated, and the behavior of the system will change).

Thus, when "updating the Web", two aspects must be taken into account: there are "local" updates that apply to a given Web site only; and there are updates in a "distributed" scenario, that must be propagated from one Web site to another.

In the first case, an update language according to the data model of the respective Web site is required. In the second, there must be a declarative, semantic framework for generically specifying the update, and additionally, a framework how to *communicate* the update (e.g. to Web sites that provide a materialized view on others' data) is required.

As already mentioned above, local update languages are in general extensions of query languages: the query language is used (i) for determining what data is updated, and (ii) for determining the new value. Note that in both cases, the respective answer is in general not a single data record or a single value, but can be a set or – in the XML case – a substructure. Moreover, the value of (ii) can be dependent on the current item of the answer computed in (i).

Example 1 *Consider a Web site of an online shop, where for all articles the prices without and with VAT are given. An update of the VAT rate to this page would require to update the prices for* all *items of an answer set with an individually computed new price. (obviously, a good design in reality would only store the net price and compute the gross price by multiplying it with the current percentage for the VAT.)*

So, the local update languages will be directly based on the local query languages – in most cases, languages for operating on XML data such as XQuery/XUpdate or Xcerpt/Xchange.

Update languages for the *Web* are to be based on query languages for the Web. Using XML as the common date model, these and local languages can most likely be the same. So, the problem in general (distributed) updates is not the update language itself, but lays in the communication and propagation of updates between resources.

Update propagation consists of (i) propagating an update, and (ii) processing/materializing the update at another Web resource. The latter, we have just seen, is solved by local update languages, so the remaining problem turns out how to communicate updates on the (Semantic) Web. This problem again is twofold: (i) infrastructure and organization, and (ii) language. The former will be dealt with later in Section 6 since it is not directly a language-related issue. Next, we investigate language aspects of this communication.

5 Languages for Reactive Behavior on the Web

Following a well-known and successful paradigm, we propose to use rules, more specifically, *reactive rules* according to the *Event-Condition-Action* (ECA) paradigm for the specification of reactivity. An important advantage of them is that the *content* of the communication can be separated from the *generic semantics* of the rules themselves. Cooperative and reactive behavior is then based on events (e.g., an update at a data source where possibly others depend on). The depending resources detect events (either they are delivered explicitly to them, or they poll them via the communication means of the Web; see next section) Then, conditions are checked (either simple data conditions, or e.g. tests if the event is relevant, trustable etc.), which are queries to one or several nodes and are to be expressed in the proposed query language. Finally, an appropriate action is taken (e.g., updating own information accordingly). This action can also be formulated as a transactions whose ACID properties qensure that either all actions in a transaction are performed, or nothing of is done.

Thus, the language for the ECA rules must comprise a language for describing events (in the "Event" part), the language for queries (in the "Condition" part), and a language for actions (including updates) and transactions (in the "Action" part). An important requirement here is, that specification and event detection is as much declarative and application-level as possible. The design of event languages should be based on the experiences in the area of *active database systems*.

Events. An (atomic) event is in general any detectable occurrence on the Web, i.e., (local) system events, incoming messages including queries and answers, transactional events (commit, confirmations etc), updates of data anywhere in the Web, or any occurrences somewhere in an application, that are (possibly) represented in explicit data, or signalled as the event itself. For these *atomic events*, it must also be distinguished between the event itself (carrying application-specific information), and its metadata, like the type of event (update, temporal event, receipt of message, ...), time of occurrence, the time of detection/receipt (e.g., to refuse it, when it had been received too late), the event origin or its generator (if applicable; e.g. in terms of its URI).

Reactive rules often do not specify reactions on atomic events, but use the notion of *complex events*, e.g., "when E_1 happened and the E_2 and E_3, but not E_4 after at least 10 minutes, then do A". Complex events are usually defined in terms of an *event algebra* [8, 21]. Thus, a declarative language for describing complex events is required, together with algorithms for handling complex events. This language should not be concerned with what the information contained in the event might be, but only with types of events.

So, several integrated languages have to be defined: the surrounding language for complex events, a language for atomic events and their metadata, and languages for expressing the contents of different types of events.

This point, again, calls for modular design – an event processor must be able to process all kinds of atomic events and to cooperate with the complex event module (which can be rule-driven by itself), and to forward the event *contents* to suitable processors.

A main difference wrt. active database systems is here that on the Web with its autonomous, often a priori unknown nodes, more information about time points and identity of event generatoring instances must be present since there is no centralized, common synchronization.

An aspect here that will become important and has to be tackled, is integrability of the event meta language, the event contents languages, and the query language. It is desirable that the specification of complex events can be combined with requirements on the state of resources at given intermediate timepoints, e.g. "when at timepoint t_1, a cancellation comes in and somewhere in the past, a reservation request came in in a timepoint when all seats were booked, then, the cancellation is charged with an additional fee". In this case, the complex event handler has to state a query at the moment when a given event arrives. For being capable of describing these situations, a logic (and system) that deals with sequences of events and queries is required. Such approaches have e.g. been presented in *Transaction Logic* [4] and in *Statelog* [15]; we will also investigate the use of Evolving Logic Programs [1] for this purpose.

The view described up to this point is to result in a rule-based infrastructure for implementing and reasoning about evolution on the Web. This infrastructure relies on the detection of (complex) events and the communication of messages (represented in XML), and on reaction rules that define the behavior of resources upon detection of events. Local knowledge is defined by facts, derivation rules, and reaction rules. All of this local knowledge is encoded in XML, and is updatable, in the sense that the update language to be developed must be capable of changing both fact, derivation rules and reactive rules. Here we may rely on the studies done in the context of logic programming about updating derivation and reactive rules (e.g. in [2]).

Several research steps must be taken in order to reach these results, besides the definition and implementation of a global language of rules, events, and messages covering evolutionary and reactive behavior (in the lines described above). Namely, one further needs to find and investigate suitable formalizations for describing the autonomous, distributed nature of this knowledge, and its change in time, and to express a global semantics and reasoning principles about the *Web Evolution and Reactivity* rule language in these formalisms.

For using the resulting infrastructure, possible communication structures inside groups of nodes on the Web must be identified and classified. These structures are essential for having global strategies of propagating the knowledge among the various nodes, and will be the subject of the next section.

6 Communication Structure and Knowledge Propagation

Communication on the Web as a living organism consisting of autonomous sources and groups or communities of sources requires knowledge about capabilities and behavior of participating resources. Thus, every resource must be distinguished by its capabilities, both from the point of view as a knowledge base, and from a directly functional point (i.e. its interface, e.g. considered as a Web service). In general, for establishing and maintaining communities, metadata about available resources must be communicated, before actual data can be handled. This general case requires the definition of semantic properties of nodes (as on the Semantic Web), and is discussed in Section 7. However, several aspects of the structure for communication and propagation of data can be discussed, and deserve investigation, even on a non-Semantic Web infrastructure as the one described up to now.

Without metadata about resources, communication concerning the *evolution* of the Web is bound to definite links between resources. In this setting, evolution takes place if a resource (or its knowledge) evolves locally, and another resource that depends upon it also evolves (as a reaction).

Example 2 *Consider the Website of a vendor of computer parts, where for each article some technical data and the price is given, and another resource that compares the offers of several vendors and shows e.g., the cheapest vendor for all parts etc. In case that a vendor page changes, the comparison page must also change.*

For this to be possible without the recourse to a Semantic Web, the roles of resources must be predefined. Here resources can be classified according to their role as information providers/consumers: Some resources only provide information: these are data "sources" that are updated locally by their maintainers (e.g., the flight schedule database of the *Lufthansa*). We call such resources *information sources*. Other resources *combine* information from other resources according to their own application logic, and that in course provide this information (e.g. travel agencies providing packages of flights and train connections). We call such resources *information transformers*. A third kind of resources only uses information from others but are not (or only in small scale) used as information providers (e.g., statistics about the average load of flights).

Moreover there must exist knowledge about other resources and their roles, for evolution to take place. The above roles of resources induce different handlings of the *communication paths*. One possibility is to have a *registered communication path*, where the "provider" knows about the "client" who uses its data. In another possibility the user of the information knows his "provider", but not vice versa.

Additionally, information transformers can be classified according to their way how their knowledge is represented, and how it is maintained. An information transformer may not materialize its knowledge but, rather, answer queries only by transforming them into queries against sources/transformers via its application logic. Alternatively, an information transformer may maintain an own (materialized) database that is populated with data that is derived from information from other sources/transformers.

The application logic of non-materializing transformers is in general specified Datalog-like where their *knowledge base* is represented via *derivation rules*. There is no internal knowledge except the rule base, thus, answers are always computed from the current state of the used transformers or sources. There is no knowledge maintenance problem; on the other hand, such resources totally depend on the 24/7 availability of their information providers.

The application logic of materializing transformers can either be specified in a Datalog-style, or by *reactive rules* that say how to update their information according to incoming messages (or a combination of both). In the latter case, the resource is more independent from its providers, but the *view maintenance problem* comes up.

Using peer-to-peer-communication, the propagation of knowledge between sources –which is obviously desirable to keep resources up-to-date– can then be done in two distinct ways:

- Push: an information source/transformer informs registered users of the updates. A directed, targeted propagation of changes by the *push* strategy is only possible along registered communication paths. It takes place by explicit *messages*, that can be update messages, or just information about what happened.
- Pull: resources that obtain information from a fixed informer can *pull* updates by either explicitly asking the informer whether he executed some updates recently, or can regularly update themselves based on queries against their providers. Communication is based on queries and answers (that are in fact again sent as messages). Here, *continuous query* services can be helpful.

For putting to work this form of propagation of knowledge on the Web, even at a non-semantic level, much work remains to be done. Firstly we have to extend the basic approach (queries) described in Section 3 to updates: updates of XML data (using XQuery/XUpdate, XPathLog, or XChange). Here, we have to deal also with updates "along" XLink references, extending the querying approach described in [18]. In a next research step, the communication of changes due to evolution of a resource to other resources by the *push* strategy will be studied and implemented. Here, especially the representation of messages in combination with appropriate reaction rules of the "receiving" source has to be designed. Also, *active* rules that implement the *pull* strategy will be investigated, together with suitable strategies (e.g. change-log in XML) of information providers.

Note that both in the case of push and pull strategies, the actual reactivity, i.e., how the instance that is informed reacts on an *event*, is expressed by ECA rules as described in the previous section.

7 Evolution and Reasoning on the Semantic Web

The aim of the Semantic Web endeavor is to provide a *unified*, *integrated* view on the Web on a conceptual level by enriching today's Web data with meta-data. This meta-data attaches explanations or formalizations of the base data meaning and thus allows for an automatic retrieval of data or of services offered on the Web in terms of the data or service descriptions provided by the meta-data.

In our research we assume that the Semantic Web will consist of sets of resources (or *communities*) that can be accessed by using a common semantic level (i.e., conceptual model, ontology, and access mechanisms). For an application area, there can be several such communities. Queries are then stated against the semantic level that forwards them towards the individual sources (via a mapping), and that combines the results also according to the semantic level. Communities can have different intra-community structure, which in course has implications to the interface towards the user. In a centralized community, there exists a central resource that does the mapping tasks, and that serves as user interface for querying. In a non-centralized community, each source performs the mapping task by itself when it is queried, and (sub)queries are forwarded to relevant nodes in the community to be answered. The mapping between the communities must be done by special *bridge* resources that belong to two or more communities and that *mediate* between the ontologies. These bridge resources will use GLAV-like [14] bidirectional ontology mappings between participating communities for mapping queries, answers, and also communication messages between the communities.

The issue of query languages for the Semantic Web, and the research steps we intend to take in this direction, have already been briefly discussed in Section 2. But the Semantics Web also raises new issues, and opens new possibilities for studying and building an infrastructure for evolution in the Web:

Example 3 (Update on the Semantic Web) *Consider an update in the timetable of Lufthansa that adds a daily flight from Berlin to Timbuktu at 12:00, arriving at 20:00. There will be several other resources for whom this information is relevant. First there are german travel agencies etc. who use the Lufthansa schedule for their offers. For these, it is probably the case that they are directly informed, or that they query the LH timetable regularly. There will also be other resources that have no immediate connection to the LH resource. E.g., a taxi service at Timbuktu can have the rule "when a european airline provides a flight to Timbuktu, we should provide a car at the airport with a driver who speaks the language of the country where the plane comes from". Here, the resource should somehow be able to detect updates somewhere on the Semantic Web that are relevant for itself.*

For this, it is clear that we need a "global" language for updating and communicating updates, and communication strategies how to propagate pieces of information through the Semantic Web. For languages, we will investigate possibilities of lifting the above sketched ones in order to be able to work on the ontological level. For studying communication strategies, i.e. how to propagate pieces of information through the Semantic Web, we focus on two scenarios: Firstly, in "controlled" communities, the participating resources are known a priori and contribute to a common task (e.g. in a tourist information system, which is one of the prospective testbed scenarios in REWERSE). Second, in "dynamic" communities the participating resources are not completely known a priori, but the whole Semantic Web is considered as a potential information source. In this case, peers that provide a given information must be searched and then queried to answer a message. Here, search for a given "service" must be organized in a CORBA-like way - but, in general without a central repository.

The latter case allows to use the union of *all* information throughout the Web. In it, the *Semantic*-property of the Web is crucial for automatically mediating between several actual schemata. Moreover, such communities provide a natural notion of community-global state, as the union of the knowledge in each community resource. For maintaining the consistency of the global state in this environment of autonomous resources, appropriate communication and reasoning mechanisms are needed. These must be able to foster communication of updates between the participating resources.

Reasoning on (and about) the Semantic Web is an area that deserves further research on various topics, among which the ones below that we will study in the course of the REWERSE project; especially the following issues:

Uncertainty and Incomplete Knowledge. The Semantic Web gives the user (and its resources) an "infinite" feeling (no instance can ever claim to collect all knowledge of the Web as a whole). Thus, "global" reasoning inside the Web must be based on a "as-far-as-I-know"-logic: all reasoning *on* the Web has to cope with uncertainty and incomplete knowledge, and, in case of derived facts in general also with nonmonotonic "belief" revisions (negation, aggregates).

Dealing with Quality of Information. Obviously, the Web (and also the Semantic Web) contains inconsistent information - both by simply erroneous or outdated information. Additionally, many information sources provide opinions, beliefs, conjectures and speculations. Often, these should also be handled (e.g., for answering questions like "what stocks in automobile industry are recommended to buy?"). Even, information *about* (here, we do not mean metadata, but e.g., evaluations of the trustworthyness and the quality, or comments about statements of other resources) resources is available. Thus, reasoning has to cope with inconsistencies, incompleteness, and with modal information.

Inconsistencies and Consistency Management. Inconsistencies can occur inside the information of a resource (to be avoided), between resources of a "community" (in most cases, also to be avoided), and across the whole Web (immanent). The latter ones cannot be avoided, but must be *handled* by each resource.

For consistency management, in addition to simply "importing" updates, resources must be able to apply additional reasoning. There may be the case that the receiver of an update does not want to implement an update because he does not believe it. Then, the community-global knowledge becomes either inconsistent (when dealing with facts), or there are different "opinions" or "presentations" (e.g., when handling news from different news services). Additionally, a resource can come to the decision that one of its informers must be replaced by another one. So, the communities can be subject to internal restructuring (still sharing the same ontology, but with different communication paths).

The Semantic Web itself also has to act in a self-cleaning way: Resources that provide erroneous or outdated information can be notified, requested for update, and potentially sanctioned by others. For doing this, instances with powerful reasoning capabilities (and equipped with some authority!) are needed. Communities inside the Semantic Web will turn out to be useful for this kind of *quality management*.

Temporal Aspects. Mainly for reasoning *about* evolution and reactivity in the Web, a logic for reasoning about several states is needed. We need to be able to express evolution and reactivity of individual resources, and of the global state. Here again, two issues will be investigated:

For describing and reasoning about immediate reactions, e.g., for update propagation and consistency maintenance, only a low number of successive states is needed. Here, a detailed reasoning on the states is required, and a logic in the style of classical temporal logics, LTL or CTL with a Kripke semantics based on a given semantics for individual states, could be used.

On the other hand, reasoning about activities involving a higher number of states for describing *transaction*-like activities on the Web, mostly takes place on a higher level of abstraction. Here, the temporal aspect is dominating over the actual data, and a logic in the style of Transaction Logic [4] is useful.

8 Conclusions

This article has outlined aspects of querying and updating resources on the Web and on the Semantic Web and stressed approaches the authors are investigating in the framework of the Rewerse research project (cf. `http://rewerse.net`). Much remains to be done. Firstly, a querying framework has to be selected and enhanced into a formalism for specifying evolution and reactivity on the Web and Semantic Web. This has to be done considering conventional Web applications and query languages as well as the emerging Semantic Web query languages as well as Semantic Web use cases. Secondly, the approaches proposed have to be checked against these use cases as well as against other use cases developed in other contexts, possibly stressing other concerns.

Acknowledgement

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, A. Brogi, J. A. Leite, and L. M. Pereira. Evolving logic programs. In S. Flesca et al, editor, *JELIA'02*, pages 50–61. Springer LNAI, 2002.
2. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming*, 45(1–3):43–70, 2000.
3. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *Intl. Conference on Very Large Data Bases (VLDB)*, 2001.
4. A. J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133(2):205–265, 1994.
5. F. Bry and S. Schaffert. Towards a declarative query and transformation language for XML and semistructured data: Simulation unification. In *Intl. Conf. on Logic Programming (ICLP)*, pages 255–270, 2002.
6. F. Bry, T. Furche, P. Pătrânjan, and S. Schaffert. Data retrieval and evolution on the (semantic) web: A deductive approach. Technical Report PMS-FB-2004-13, University of Munich, May 2004.
7. P. Buneman, S. Davidson, G. Hillebrandt, and D. Suciu. A query language and optimization techniques for unstructured data. In *ACM Intl. Conference on Management of Data (SIGMOD)*, pages 505–516, Montreal, Canada, 1996.
8. S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In VLDB, 1994.
9. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion. In *ACM Intl. Conference on Management of Data (SIGMOD)*, 1999.
10. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A query language for XML. `http://www.w3.org/TR/1998/NOTE-xml-ql`, 1998.
11. M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. STRUDEL: A web-site management system. In *ACM SIGMOD*, pages 549–552, 1997.
12. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2), 1997.
13. H. Hosoya, B. Pierce. Xduce: A typed XML processing language. *WebDB 2000*.
14. M. Lenzerini. Information integration. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.
15. Bertram Ludäscher. *Integration of Active and Deductive Database Rules*. DISDBIS 45, infix-Verlag, Sankt Augustin, 1998. PhD thesis, Universität Freiburg.
16. B. Ludäscher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing semistructured data with FLORID: A deductive object-oriented perspective. *Information Systems*, 23(8):589–612, 1998.
17. W. May. A rule-based querying and updating language for XML. In *Workshop on Databases and Programming Languages (DBPL 2001)*, Springer LNCS 2397, 2001.
18. W. May. Querying linked XML document networks in the web. In *11th. WWW Conference*, 2002. Available at `http://www2002.org/CDROM/alternate/166/`.
19. W. May. Xpath-logic and xpathlog: A logic-programming style XML data manipulation language. *Theory and Practice of Logic Programming*, 4(3), 2004.
20. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.
21. M. P. Singh. Semantical considerations on workflows: An algebra for intertask dependencies. In *Proc. DBPL-5*, volume 5, 1995.
22. I. Tatarinov, Z. Ives, A. Halevy, and D. Weld. Updating XML. In *ACM Intl. Conference on Management of Data (SIGMOD)*, pages 133–154, 2001.