Xcerpt and XChange: Deductive Languages for Data Retrieval and Evolution on the Web

François Bry, Paula-Lavinia Pătrânjan, and Sebastian Schaffert
Institute for Informatics, University of Munich, http://www.pms.ifi.lmu.de

Abstract: In this article, two deductive languages are introduced: the language *Xcerpt*, for querying data and reasoning with data on the (Semantic) Web, and the language *XChange*, for evolution and reactivity on the (Semantic) Web. A small application scenario is given as a motivation for these languages.

1 Motivation

The *Semantic Web* is an endeavor aiming at enriching the existing Web with meta-data and data (and meta-data) processing so as to allow computer systems to actually *reason* with the data instead of merely *rendering* it. To this aim, it is necessary to be able to *query* and *update* data and meta-data. Existing Semantic Web query languages (like DQL [2] or TRIPLE[3]) are special purpose, i.e. they are designed for querying and reasoning with special representations like OWL or RDF, but are not capable of processing generic Web data. On the other hand, the language *Xcerpt* [BS02] presented in this article is a general purpose language that can query any kind of XML data, i.e. standard Web as well as Semantic Web data, and at the same time provides advanced reasoning capabilities. It could thus serve to implement a wide range of different reasoning formalisms. Likewise, the maintenance and evolution of data on the (Semantic) Web is necessary: the Web is a "living organism" whose dynamic character requires languages for specifying its evolution. This requirement regards not only updating data from Web resources, but also the propagation of changes on the Web. These issues have not received much attention so far, existing update languages (like XML-RL Update Language [LLW03]) and reactive languages developed for XML data offer the possibility to execute just simple update operations and, moreover, important features needed for propagation of updates on the Web are still missing. The language *XChange* also presented in this article builds upon the query language Xcerpt and provides advanced, Web-specific capabilities, such as propagation of changes on the Web (*change*) and event-based communication between Web sites (*exchange*), as needed for agent communication and Web services.

[1]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).

[2]DAML Query Language, http://www.daml.org/dql.

[3]TRIPLE Language, http://triple.semanticweb.org.

2 Flavours of Xcerpt: Querying Data on the Web

Xcerpt is a declarative, rule-based query language for Web data (i.e. XML documents or semistructured databases) based on logic programming. An Xcerpt program contains at least one *goal* and some (possibly zero) *rules*. Rules and goals consist of query and construction patterns, called *terms*. Terms represent tree-like (or graph-like) structures. The children of a node may be either *ordered* (as in standard XML) or *unordered* (as is common in databases). An *ordered term specification* is expressed by square brackets, an *unordered term specification* by curly braces. Single (square or curly) braces denote terms matching such terms containing one matching subterm for each subterm, without additional (non-matching) subterms (*total term specification*). Double braces denote terms matching such terms containing one matching subterm for each subterm, possibly with additional (non-matching) subterms (*partial term specification*).

Data terms are used to represent XML documents and the data items of a semistructured database. They are similar to *ground* functional programming expressions and logical atoms. A *database* is a (multi-)set of data terms (e.g. the Web).

Example 1. The following two data terms represent a train timetable and a hotel reservation offer.

At site `http://railways.com`:

```
travel {
  last-changes-on { "2004-09-10" },
  currency { "EUR" },
  train {
    departure {
      station { "Munich" },
      date { "2004-09-23" },
      time { "21:30" }
    },
    arrival {
      station { "Ulm" },
      date { "2004-09-23" },
      time { "22:45" }
    },
    price { "25" }
  },
  train {
    departure {
      station { "Ulm" },
      date { "2004-09-25" },
      time { "14:34" }
    },
    arrival {
      station { "Munich" },
      date { "2004-09-25" },
      time { "15:49" }
    },
    price { "25" }
  },
  ...
}
```

At site `http://hotels.net`:

```
voyage {
  currency { "EUR" },
  hotels {
    city { "Ulm" },
    country { "Germany" },
    hotel {
      name { "Comfort Blautal" },
      category { "3 stars" },
      price-per-room { "55" },
      phone { "+49 88 8219 213" },
      no-pets {}
    },
    hotel {
      name { "Inter City" },
      category { "3 stars" },
      price-per-room { "57" },
      phone { "+49 88 8156 135" }
    },
    hotel {
      name { "Maritim" },
      category { "4 stars" },
      price-per-room { "106" },
      phone { "+49 88 8123 414" }
    },
    ...
  },
  ...
}
```

Query Terms are patterns matched against Web resources represented by data terms. They are similar to the latter, but augmented by *variables* (for selecting data items), possibly with *variable restrictions* using the $\rightsquigarrow$ construct (restricting the possible bindings to certain subterms), by *partial term specifications* (omitting subterms irrelevant to the query), and by additional query constructs like *subterm negation* (keyword `without`), *optional subterm specification* (keyword `optional`), and *descendant* (keyword `desc`).

Construct Terms serve to reassemble variables (the bindings of which are specified in

query terms) so as to construct new data terms. Again, they are similar to the latter, but augmented by *variables* (acting as place holders for data selected in a query) and the *grouping construct* `all` (which serves to collect all instances that result from different variable bindings). Occurrences of `all` may be accompanied by an optional sorting specification.

Construct-Query Rules (short: rules) relate a construct term to a query consisting of AND and/or OR connected query terms. They have the form `CONSTRUCT` *Construct Term* `FROM` *Query* `END`. Rules can be seen as "views" specifying how to obtain documents shaped in the form of the construct term by evaluating the query against Web resources (e.g. an XML document or a database). Queries or parts of a query may be further restricted by arithmetic constraints in a so-called condition box, beginning with the keyword `where`.

Example 2. The following Xcerpt rule is used to gather information about the hotels in Ulm where a single room costs less than 70 Euro per night. Hotels where no pets are allowed are not of interest (specified using the Xcerpt `without` construct).

```
CONSTRUCT
  answer [ all var H ordered by [ P ] ascending ]
FROM
  in {
    resource { "http://hotels.net"},
    voyage {{
      hotels {{
        city { "Ulm" },
        desc var H ⇝ hotel {{ price-per-room { var P }, without no-pets {}
                    }}
      }}
    }}
  } where var P < 70
END
```

An Xcerpt query might contain one or several references to *resources*. Xcerpt rules might be *chained* like active or deductive database rules to form complex query programs.

Xcerpt supports namespaces and references: in contrast to most Web and Semantic Web query languages, Xcerpt automatically performs dereferencing. In its current version, Xcerpt does not support some particularities of XML. E.g. Xcerpt abstracts out the various referencing mechanisms of XML (e.g. global parameter entities, links, ID-IDREF attributes) into one single reference mechanism. This limitations of the current version of Xcerpt are no limitations of the Xcerpt approach, for they can be easily lifted and at no conceptual cost.

With Xcerpt data can be retrieved from every Web site, links in retrieved data can be followed, and, since Xcerpt supports dereferencing (e.g. of IDREF attributes) non-tree, e.g. cyclic graph data can be queried. These three features make Xcerpt amenable to query graph structures (located at one single Web site or distributed over the Web) like e.g. RDF or Topic Maps structures as well as OWL relationships.

3 Flavours of XChange: Exchanging Events on the Web

The language XChange aims at establishing reactivity, expressed by *reaction rules*, as communication paradigm on the Web. With XChange, communication between Web sites is peer-to-peer, i.e. all parties have the same capabilities and can initiate communication, and synchronisation can be expressed, so as to face the fact that communication on the

Web might be unreliable and cannot be controlled by a central instance.

XChange Events. The framework in which events are raised/notified (i.e. the Web) has some characteristics, i.e. no central clock, no central management, no central synchronisation, which need to be taken into consideration. An event raised at a site may have different notification times at its reception sites. An XChange event has two time stamps: raising time (i.e. when the event has been raised) and notification time (i.e. when the event has been received). The approach taken in XChange excludes broadcasting of events on the Web, before an event is raised its recipient Web site(s) is (are) determined. A Web site that has raised an event must know which Web sites to send the event to and a reception Web site might need to know which site has send an event it receives. Thus, an XChange event has a raising URI (i.e. where the event has been raised) and one or more notification URIs (i.e. where the event is to be sent).

XChange events are XML data, hence a generic data exchange between Web sites is supported by XChange, making easier the transfer of parameters (e.g. raising time, notification URI(s)) and thus the execution of actions in a user-defined synchronised manner. Examples of events: delay notification, update of a train table.

Example 3. A train has 30 minutes delay, the control point that has observed this event raises it to `http://railways.com`, as:

```
delay { to { "http://railways.com" },
        train { departure { station { "Munich" }, date { "2004-09-23" },
                            time { "21:30" } },
                minutes-delay { "30" } }
 }
```

XChange Event-Raising Rules. The notified sites (e.g. `http://railways.com` in the above example) process the incoming events, thus only the information of interest for these sites is used (e.g. the time stamps may be used to decide whether the events are too old or not), in order to execute (trans)actions or to raise other events. The processing of events is specified in XChange by means of event-raising rules, event-driven update rules, and event-driven transaction rules (discussed in the next section). The body of an event-raising rule may contain Xcerpt queries to incoming events, Xcerpt queries to XML resources (local or remote), and conditions that variables (specified in the queries to incoming events or XML resources) should satisfy. The head of an event-raising rule contains resource specifications (i.e. the resources to be notified) and event specifications (used to construct event instances).

Example 4. Mrs. Smith uses a travel organiser that plans her trips and reacts to happenings that might influence her schedule. The site `http://railways.com` has been told to notify her travel organiser of delays of trains Mrs. Smith travels with:

```
RAISE
  delay {
    to { "http://travelorganiser.com/Smith" },
    train { departure { var M, estimated-time { var DT + var Min } },
            arrival { var U, estimated-time { var AT + var Min } }
     }  }
ON
  delay {{
    train {{
      departure {var M ⇝ station{"Munich"}, var Date ⇝ date{"2004-09-23"},
                 time { var DT ⇝ "21:30" } }, minutes-delay { var Min } }}
   }}
FROM
  in {
    resource { "http://railways.com" },
    travel {{ train {{ departure {{ var M, var Date, time {var DT} }},
                       arrival {{ var U ⇝ station {"Ulm"}, time {var AT}
            }} }}
     }}  }
END
```

4 Flavours of XChange: Propagating Changes on the Web

XChange provides the capability to specify relations between complex updates and execute the updates conformly (e.g. when booking a trip on the Web one might wish to book an early flight *and* of course the corresponding hotel reservation, *or* a late flight *and* a shorter hotel reservation). As the updates are to be executed on the Web, network communication problems could cause failures of update execution. To deal with such problems, an explicit specification of synchronisation of updates is possible with XChange.

XChange Updates. The XChange update language uses rules to specify intensional updates, i.e. a description of updates in terms of queries. The notion of update rules is used to denote rules that specify (possibly complex) updates (i.e. insertion, deletion, and replacement). The body of an XChange update rule may (and generally does) contain an Xcerpt query, which specifies bindings for variables and conditions that variables should satisfy. The head of an XChange update rule contains resource specifications for the data that is to be updated, update specifications, and relations between the desired updates. An XChange update specification is an (incomplete) pattern for the data to be updated, augmented with the desired update operations.

Example 5. At `http://railways.com` the train timetable needs to be updated as reaction to the event given in Example 3:

```
UPDATE
  in {
    resource { "http://railways.com" },
    travel {{
      last-changes-on { var L replaceby var RTime },
      train {{
        departure {{ station { var DS }, var Date, time { var DT },
                     insert estimated-time { var DT + var Min } }},
        arrival {{ time {var AT}, insert estimated-time {var AT + var Min}
           }}  }}
     }}  }
ON
  delay {{
    raising-time { var RTime },
    train {{ departure { station { var DS }, var Date, time { var DT } },
             minutes-delay { var Min } }}
   }}
END
```

XChange Transactions. As sometimes complex updates need to be executed in an all-or-nothing manner (e.g. when booking a trip on the Web, a hotel reservation without a flight reservation is useless), the concept of transactions (one or more updates treated as one unit) is supported by the language XChange. In case of transaction abort a rollback mechanism, which undoes partial effects of a transaction, is to be used. XChange transactions are transactions executed on user requests or as reactions to incoming XChange events. The latter transactions are specified in the head of XChange event-driven transaction rules.

Example 6. The travel organiser of Mrs. Smith uses the following rule: if the train of Mrs. Smith is delayed such that her arrival will be after 23:00h then book a hotel at the city of arrival and send the telephone number of the hotel to her husband's address book. The rule is specified in XChange as:

```
TRANSACTION
  and [
    update {
      in { resource { "http://hotels.net" },
      reservations {{
        insert reservation { var H, name { "Christina Smith" },
                             from {"2004-09-23"}, until {"2004-09-24"} }
      }}
    } },
    update {
      in { resource { "address-book://addresses/my-husband" },
      addresses {{ insert my-hotel { phone { var Tel },
                             remark {"I'm staying in Ulm over night!"} } }}
    } }
  ]
ON
  delay {{
   from { "http://railways.com" },
   train {{ arrival { station { var City ⇝ "Ulm" },
                      estimated-time { var ETime } } }}
  }} where var ETime after 23:00
FROM
  in {
   resource { "http://hotels.net" },
   voyage {{ hotels {{ city {var City},
                       desc var H ⇝hotel {{ price-per-room {var P},
                                            phone {var Tel} }}
    }} }}
  } where var P < 70
END
```

5 Conclusion

This paper gives flavours of two declarative languages, Xcerpt for querying and XChange for updating, aiming at an easier programming of Web and Semantic Web applications. A prototypical implementation[4] of Xcerpt and a visual rendering of Xcerpt programs (visXcerpt [BBSW03]) are already available, aa well as a specification of Xcerpt's semantics [BS03]. The first steps have been made towards the development of XChange.

[4] Xcerpt Prototype, `http://demo.xcerpt.org`.

References

[BBSW03] Berger, S., Bry, F., Schaffert, S., und Wieser, C.: Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data. In: *Int. Conf. on Very Large Databases (VLDB)*. 2003.

[BS02] Bry, F. und Schaffert, S.: Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In: *Proc. Int. Conf. on Logic Programming (ICLP)*. LNCS 2401. Springer-Verlag. 2002.

[BS03] Bry, F. und Schaffert, S.: An Entailment for Reasoning on the Web. In: *Proc. of Rules and Rule Markup Languages for the Web (RuleML03)*. LNCS. Springer-Verlag. 2003.

[LLW03] Liu, M., Lu, L., und Wang, G.: A Declarative XML-RL Update Language. In: *Proc. Int. Conf. on Conceptual Modeling (ER 2003)*. LNCS 2813. Springer-Verlag. 2003.