# Xcerpt and XChange

## *Logic Programming Languages for Querying and Evolution on the Web*

François Bry, Paula-Lavinia Pătrânjan, Sebastian Schaffert
`http://www.pms.ifi.lmu.de`

## Motivation

The *Semantic Web* is an endeavour aiming at enriching the existing Web with meta-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 or TRIPLE) 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* presented here is a general purpose language that can query any kind of XML data and at the same time, being based on logic programming, 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) 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 here builds upon the query language Xcerpt and provides advanced, Web-specific capabilities, such as propagation of changes on the Web (*change*) and event-based communications between Web sites (*exchange*).

## Xcerpt: a Logic Language for Web Querying

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 (maybe zero) *rules*. Rules and goals consist of query and construction patterns, called *terms* in analogy to other logic programming languages.

### Web Data as Terms

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

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

***Figure:*** *Xcerpt Rule to retrieve a list of hotels with a price less than 70€ where pets are not disallowed, ordered by price.*

**Query Terms** are patterns matched against Web resources represented by data terms. They are similar to the latter, but augmented with *variables* (for selecting data items), possibly with *variable restrictions* (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*, *optional subterm specification* and *descendant*.

**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).

```
voyage {
  currency { "EUR" },
  hotels {
    town { "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" }
     },
  ...
  },
...
}
```

***Figure:*** *A Data Term representing a hotel database*

### Construct-Query Rules

Construct-Query rules (short: rules) relate a construct term to a query consisting of `and` and/or `or` connected query terms. Rules can be seen as ``views" specifying how documents shaped in the form of the construct term can be obtained by evaluating the query against Web resources (e.g. an XML document or a database).

## XChange: Evolution of Data on the Web

### 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.

The processing of events is specified in XChange by means of event-raising rules, event-driven update rules, and event-driven transaction rules. *Event-raising rules* specify events that are to be constructed and raised as reaction to incoming (internal or external) events.

### Propagating Changes on the Web

XChange provides the capability to specify relations between complex updates and execute the updates conformly (e.g. in booking a trip on the Web, one might wish to book an early flight *and* of course the corresponding hotel reservation). To deal with network communication problems, an explicit specification of synchronisation operations on updates is needed, a (kind of) control which logic programming languages lack.

*Update rules* are rules specifying (possibly complex) updates. The head of an update rule contains patterns for the data to be modified, augmented with update operations (i.e. insertion, deletion, replacement), called *update terms*, and the desired synchronisation operations.

As sometimes complex updates need to be executed in an all-or-nothing manner (e.g. in 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 XChange. Transactions may be executed on user requests or as reactions to incoming events (the latter transactions are specified using *event-driven transaction rules*).

```
RAISE
  delay {
    to { "http://travelorganizer.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, var DT }},
        arrival {{ var U -> station { "Ulm" },
                    time { var AT } }} }}
     }}
   }
END
```

***Figure:*** *Mrs. Smith uses a travel organizer, which plans her trips and reacts to happenings that might influence her schedule.*
*The site* `http://railways.com` *has been told to notify her travel organizer of delays of trains Mrs. Smith travels with this event-raising rule.*

```
TRANSACTION
  and [
    update {
      to { "http://hotels.net" },
      reservations {{
        insert reservation {
          var H, name { "Christina Smith" },
          from { "2004-09-23" },
          until { "2004-09-24" } }
      }}
    },
    update {
      to { "address-book://addresses/husband" },
      addresses {{
        insert my-hotel {
          phone { var Tel },
          remark { "Staying here over night!" } }
      }}
    }
  ]
ON
  delay {{
    from { "http://railways.com" },
    train {{
      arrival { station { var Town -> "Ulm" },
                estimated-time { var ETime }   }
    }}
  }} where var ETime after 23:00
FROM
  in {
    resource { "http://hotels.net" },
    voyage {{
      hotels {{
        town { var Town },
        desc var H -> hotel {{
          price-per-room { var P },
          phone { var Tel }, without no-pets {}  }}
      }}
    }}
  } where var P < 70
END
```

***Figure:*** *The travel organizer of Mrs. Smith uses the following event-driven transaction rule: if the train of Mrs. Smith is delayed such that her arrival will be after 23:00 then book a hotel at the city of arrival and send the telephone number of the hotel to her husband's address book.*