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 DOL or TRIPLES) 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 formal programming expressions and logical atoms. A database is a (multi-)set of data terms (e.g. XML documents).

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 descendend.

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 placeholders for data selected in a query) and the grouping construct all (which serves to collect all instances that result from different variable bindings).

Constructions

Web Data Terms

```
Xcerpt Rule to retrieve a list of hotels with a price less than 70€: 

answer [ all var H ordered by [ P ] ascending ]

| FROM | in, resource [ "http://hotels.net" ],
|      | voyage [ [ hotel { [ town = "Ulm", 
|      |      desc var H -> hotel { 
|      |      price-per-room [ var P ], 
|      |      without no-pets () } } ] 
|      | ] ] |
|      | where var P < 70 |}
```

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

XChange: Evolution of Data 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 can 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 consequently (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).

```
BEGIN

CONSTRUCT

answer [ all var H ordered by [ P ] ascending ]

FROM

in, resource [ "http://hotels.net" ],
voyage [ [ hotel { [ town = "Ulm", 
      desc var H -> hotel { 
      price-per-room [ var P ], 
      without no-pets () } } ] 
|      | ] ] |

|       where var P < 70 |}

END

```

```
BEGIN

XChange provides the capability to specify relations between complex updates and execute the updates consequently (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).

```
BEGIN

XChange provides the capability to specify relations between complex updates and execute the updates consequently (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).
```