

# ***Xcerpt and XChange: Deductive Languages for Data Retrieval and Evolution on the (Semantic) Web***

Christoph Wieser

`http://www.pms.ifi.lmu.de`

Institute for Informatics, University of Munich

# Contents

## 1. Motivation

## 2. Data Retrieval: The Language [Xcerpt](#)

Related Work

Underlying Ideas

Constructs

## 3. Evolution of Data: The Language [XChange](#)

Related Work

Underlying Ideas

Constructs

## 4. Conclusion

# 1. Motivation

- data retrieval is key issue for many Web-based technologies/applications
- query languages for the Semantic Web are special purpose
  - ⇒ need for a general purpose language for **querying and reasoning** with any kind of XML data
- many resources on the Web and the Semantic Web are dynamic (they can change their content over time)
- general purpose programming languages are not easy to understand and use by the novice practitioners
  - ⇒ need for a language for specifying the **evolution of data** on the (Semantic) Web

## 2. Xcerpt – Related Work

- query languages for XML/Web, e.g. XQuery (2003), XSLT (1999), XPath (1999)
  - path-oriented
  - patterns serve to re-assemble path-selections
  - ⇒ strong intertwining of query and construction parts
  - ⇒ queries tend to be complicated
- query languages for Semantic Web, e.g. DQL (2002), TRIPLE (2001)
  - special purpose (i.e. querying and reasoning with special representations like OWL or RDF)
  - restricted to a specific reasoning algorithm
- logic programming

## 2. Xcerpt – Underlying Ideas

- rule-based transformations and queries
- patterns instead of paths for querying data
- patterns for creating results
- strict separation of query and construction parts
- “connected” via variables (as in logic programming)
- revisited logic programming (simulation unification)

# 2. Xcerpt – Data Terms

- represent semistructured databases

*Example 1.* An Xcerpt data term found at site <http://hotels.net>:

```
accommodation {
  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" }
    ],
    ...
  ],
  ...
}
```

## 2. Xcerpt – Query Terms

- (incomplete) patterns for querying data
- are very similar to goals in logic programming and SQL selections

... but have additional properties

- **partial queries** (answers might have additional subterms to those in the query term)
- **ordering** (answers might have a different subterm ordering than the query)
- **“descendant”** (a query term might specify subterms at an unknown depth)
- **“as”** (or  $\rightsquigarrow$ , variables with pattern restriction may be specified)
- **“optional”** (variables are bound to optional subterms if they exist, otherwise a default value may be used)
- **“without”** (states that a certain subterm must not be found in the database)

## 2. Xcerpt - Query Terms

*Example 2.* Partial vs. total matching:

```
accommodation {{
  hotels {{
    country { "Germany" }
  }}
}}
```

matches with the database

```
accommodation {
  hotels {
    country { "Germany" }
  }
}
```

does not match with the database

```
accommodation {
  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 {}
    ],
  },
}
```

## 2. Xcerpt - Construct Terms

- patterns for the result of a query
- are **templates** that are filled in with values gathered in a query
- contain variables (but no  $\rightsquigarrow$  restrictions)

*Example 3.* The construct term ...

```
results {  
  result { var CITY , var HOTEL }  
}
```

... may yield the following data term as result

```
results {  
  result {  
    city { "Ulm" },  
    hotel [ name { "Inter City" },  
            category { "3 stars" },  
            price-per-room { "57" },  
            phone { "+49 88 8156 135" }  
          ]  
  }  
}
```

# 2. Xcerpt - Construct-Query Rules

*Example 4.* A construct-query rule used to gather information about hotels in Ulm where a single room costs less than 70 Euro per night. Hotels where no pets are allowed are not of interest.

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

Construct-Query rules are range-restricted

Rules can be chained to realise complex programs

# 3. XChange – Related Work

- update languages for XML/Web (e.g. XML-RL Update Language, XUpdate, XPathLog)
  - simple updates to Web resources
  - no propagation of changes on the Web
  - no update language as standard
- (so-called) ECA languages for XML/Web/Semantic Web (e.g. Active XQuery)
  - reaction: simple updates, no transactions
  - no means for synchronisation on the Web
  - no processing of composite events
  - no ECA language as standard
- active databases offer useful concepts, but...
- characteristics of the Web need to be taken into consideration!

# 3. XChange – Underlying Ideas

- declarative language for **updating** data on the Web
- **reactivity** as communication paradigm on the Web
- rule-based language
- **pattern-oriented** update and event specifications
- synchronisation of updates on the Web
- complex updates as **transaction**
- processing of **composite events** (i.e. situations)
- update language **embedding** the query language (not the other way around!)
- builds upon the query language **Xcerpt**

# 3. XChange – Underlying Ideas

## Persistent vs. **volatile** data

- persistent data, i.e. data of Web resources (like written text)
  - (standard) queries are used for querying
  - persistent data is updatable
- volatile data, i.e. **events** (like speech)
  - **event queries** are used for querying
  - volatile data is *not* updatable

## Evolution of persistent data

- local changes
  - **updates** (i.e. insert, delete, replace) to *local persistent* data items (XML documents)
- propagation of changes
  - communication between Web sites: through **event messages** (peer-to-peer, push)
  - **reaction** to incoming events (specified using reaction rules)

# 3. XChange – Constructs

**Example 5.** 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>:

```
accommodation {
  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" }
    ],
    ...
  },
  ...
}
```

# 3. XChange – Updates

## Update

- **elementary** update (specified by an *update term*)
- **complex** update (ordered/unordered conjunctions or disjunctions of updates)

An **update term** is a kind of Xcerpt query term, augmented with update constructs

```
insert <construct-term>  
delete <query-term>  
<query-term> replaceby <construct-term>
```

**Example 6.** The following update term specifies modification of the prices given in EUR to the corresponding values in USD, in the database at <http://hotels.net>:

```
accommodation {{  
  currency { "EUR" replaceby "USD" },  
  hotels {{  
    desc price-per-room { var P replaceby var P * var ExchangeRate }  
  }}  
}}
```

# 3. XChange – Events and Event Queries

## Events

- **explicit** events
- **implicit** events (i.e. standard queries to persistent data, updates of persistent data, transactional events)
- **system** events (e.g. system clock events)

## Event queries

- **atomic** event queries (refer to one single event)
- **composite** event queries
  - temporal range** (i.e. within `TimeInterval`, before/after `TimePoint`, during `Duration`)
  - event composition** (e.g. temporally ordered conjunction, conjunction, disjunction, negation, overlapping, meets, if-then-else)
  - occurrence** (multiplicity, repetition, ranking)

# 3. XChange – Event Messages

An **event message** is an XML document with

- root element labelled **event**, and
- four parameters

<b>raising-time</b>	<i>(time stamp)</i>
<b>reception-time</b>	<i>(time stamp)</i>
<b>sender</b>	<i>(URI)</i>
<b>recipient</b>	<i>(URI)</i>

**Example 7.** A train has 30 minutes delay, the control point that observes this raises the following event and sends it to `http://railways.com`:

```
event {
  sender { "control://controlpoint/de/" },
  recipient { "http://railways.com" },
  raising-time { "2004-09-23T15:00:24" },
  delay {
    train {
      departure { station { "Munich" },
        date { "2004-09-23" }, time { "21:30" } },
      minutes-delay { "30" }
    }
  }
}
```

# 3. XChange – Reaction Rules

## Processing of events

- no central event manager
- a local event manager at each (XChange-aware) Web site
- specified by means of...

...XChange reaction (or **E**vent-**C**ondition-**A**ction) rules

**E**: *Event query*, i.e. query against volatile data

**C**: *Standard query*, i.e. Xcerpt query against persistent data

**A**: *raise event(s)*  $\Rightarrow$  *event-raising* rules, or

*transaction(s)* (i.e. group of updates and/or explicit events, executed in an all-or-nothing manner)  $\Rightarrow$  *transaction* rules

# 3. XChange – Event-Raising Rules

**Example 8.** The site `http://railways.com` notifies the travel organiser of Mrs. Smith of delays of trains she travels with:

```
RAISE
  event {
    recipient { "http://travelorganiser.com/Smith" },
    delay-notification {
      train {
        var Date,
        departure { var M, estimated-time { var DT + var Min } },
        arrival { var U, estimated-time { var AT + var Min } }
      }
    }
  }
}
ON
  event {{
    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
```

# 3. XChange – Transaction Rules

**Example 9.** The travel organiser of Mrs. Smith uses the following rule:

```
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 { "diary://diary/my-husband" },
      diary {{ var Date,
        news {{ insert my-hotel { phone { var Tel },
          remark { "Train delayed! I'm staying in"+var City+"over night!" } } }}
      }}
  }
]
ON
event {{
  sender { "http://railways.com" },
  delay-notification {{ train {{ var Date ~> date {{ } } },
    arrival { station { var City ~> "Ulm" }, estimated-time { var ETime } }
    }} }}
}} where var ETime after 23:00
FROM
in { resource { "http://hotels.net" },
  accommodation {{
    hotels {{ city { var City },
      desc var H ~> hotel [[ price-per-room { var P }, without no-pets { },
        phone { var Tel } ] ] }}
  }}
} where var P < 70
END
```

## 4. Conclusion

**Xcerpt** and **XChange** are ongoing projects

Coordination: François Bry

Xcerpt: Sebastian Schaffert +

visXcerpt: Sacha Berger, Christoph Wieser

XChange: Paula-Lavinia Pătrânjan +

Semantic Web issues: Tim Furche +

Verbalisation of XML and Xcerpt: Uta Schwertel

Temporal reasoning: Stephanie Spranger +

Spatial reasoning: Bernhard Lorenz +

Types for XML data: Sacha Berger

