Reactivity on the Web: Paradigms and Applications of the Language XChange

François Bry
Institute for Informatics
University of Munich
Oettingenstr. 67
D-80538 Munich, Germany
francois.bry@ifi.lmu.de

Paula-Lavinia Pătrânjan
Institute for Informatics
University of Munich
Oettingenstr. 67
D-80538 Munich, Germany
paula.patranjan@ifi.lmu.de

ABSTRACT

Reactivity on the Web is an emerging issue. It is essential for upcoming Web systems such as online marketplaces, adaptive, Semantic Web systems as well as Web services and Grids. This article first introduces the paradigms upon which the high-level language *XChange* for programming reactive behaviour and distributed applications on the Web relies. Then, it briefly presents the main syntactical constructs of XChange. Finally, it sketches the implementation in XChange of a reactive Web-based application.

Categories and Subject Descriptors

D.3.3 [**Software**]: Programming Languages—*Language Constructs and Features*

Keywords

Web, reactive languages, event-condition-action rules

1. MOTIVATION

Many resources on the Web and the Semantic Web are dynamic in the sense that they can change their content over time. E.g. a Web-based information system on flights could report delays on departures or arrivals, and flight cancellations. A Web-based personalised organiser might be conceived so as to automatically react to such changes affecting its owner. A delayed arrival might result in issuing an email to some other person or in cancelling a hotel reservation. Thus, a strong motivation exists for having means to specify and/or request updates to (local or remote) Web sites and to propagate the updates over related Web resources. Existing update languages (like XML-RL Update Language [5] and XPathLog [6]) and reactive languages [7] developed for XML data support *simple* update operations. However, important features needed for propagation of more *complex* updates on the Web are still missing. The language *XChange* presented in this article aims at filling this gap. It builds upon the query language Xcerpt [8] and provides advanced, Web-specific capabilities, such as propagation of changes on the Web (*change*) and event-based communication between Web sites (*exchange*).

[0]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. XCHANGE: PARADIGMS

2.1 Events and Event Queries

An *event* is a happening (e.g. update of a Web resource) on which each Web site (through a reactive program) may decide to react in a particular way or not to react to at all. An important distinction is made between *persistent data* (i.e data of Web resources) and *volatile data* (i.e. events). To query persistent data *standard queries* are used (e.g. expressed using a query language like XQuery [10], or Xcerpt [8]), to query volatile data *event queries* are used. Standard and event queries might well be very similar. However, event queries are likely to refer to time or event sequences, while standard queries, related to persistent data, are likely not to refer to such notions. Based on this distinction, the metaphor of XChange for reactivity on the Web is that of *speech* for volatile data, i.e. events, and *written text* for persistent data, i.e. Web content retrieved via standard queries. Speech cannot be modified. If one has communicated some information in this way he/she can correct, complete, or invalidate what he/she has told – through further speech. In contrast, written text can be updated in the usual sense. Likewise, volatile data (i.e. events) is *not* updatable but persistent data (i.e. Web content) is updatable. To inform about, correct, complete, or invalidate former volatile data, new *event messages* (i.e. data containing informations about events that have occurred) are communicated between Web sites. Since events are volatile (like speech), event queries cannot be posed by remote Web sites (but only by a Web site referring to the events it receives). Otherwise, events would have to be made persistent – confusing the clear picture volatile vs. persistent data thus making programming much more complicated.

2.2 Communication Paradigms

Peer-to-Peer. With XChange, the communication between Web sites is based on a *peer-to-peer* communication model, i.e. all parties have the same capabilities and every party can initiate a communication session. Event messages are directly communicated between Web sites without a centralised processing of events. XChange assumes no instance controlling (e.g. synchronising) communication on the Web, even though such instance can be realised using XChange.

Self-Synchronising Reactive Programs. In the absence of central synchronisation on the Web and faced with communication unreliability, XChange programs have the capability to synchronise themselves with other XChange programs on the Web.

Push Strategy. For communicating events on the Web two strategies are possible: the *push* strategy, i.e. a Web site informs (possibly) interested Web sites about events, and the *pull* strategy, i.e. interested Web sites query periodically (poll) persistent data found at other Web sites in order to determine changes. Both strategies are meaningful. The pull strategy is supported by languages for standard queries (e.g. XQuery or Xcerpt), i.e. queries to persistent data. Therefore, so as to complement the framework, XChange offers the *push* strategy. The push strategy requires event queries to be incrementally evaluated (by so-called *event managers*). In the case of XChange, this is done at every (XChange-aware) Web site.

2.3 Transactional Reactivity

Complex Updates. An *elementary update* is a change (i.e. insert, delete, replace) within a persistent data item (e.g. XML or RDF data). *Complex updates* expressing ordered or unordered conjunctions, or disjunctions of (elementary or complex) updates are offered by XChange. Such updates are required by real applications. 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. Since it is sometimes necessary to execute such complex updates in an *all-or-nothing manner* (e.g. when booking a trip, a hotel reservation without a flight reservation is useless), XChange has a concept of transactions [9].

Transactions and ACID Properties. XChange transactions obey the ACID properties [9] (Atomicity, Consistency, Isolation, and Durability). Atomicity and isolation are considered in XChange, the issues of consistency and durability for transactions are currently not investigated in the project. XChange will build on standard solutions from database systems.

Transactional Events. Transactional events (i.e. commit, abort, request) are offered by XChange. They are needed for supporting transactions.

2.4 Authentication, Authorisation, and Accounting

XChange in its present stage of development does not offer specific means for security (especially authentication and authorisation). However, such extensions are neither incompatible with the current version of XChange nor precluded in future versions of the language. The protocols of a Grid architecture (such as Globus [4]) would provide with convenient means for such an extension. Extending XChange with accounting functionalities is a promising perspective for future research. Vice versa, XChange could be seen as a (core of a) high-level reactive language for Grids.

2.5 Relationship Between Reactive and Query Languages

A working hypothesis of the XChange project is that a reactive language for the Web should build upon, more precisely embed, a Web query language. There are two reasons for this. First, specifications of reactive behaviour often refer to actual Web contents - calling for querying Web contents. Second, reactive behaviour necessarily refers to (more or less recent) events - calling for querying events. For reasons of uniformity, it is highly desirable both for users and for system developers that the languages used for querying Web contents and querying events are as close as possible to each other. Note, however, that querying events calls for constructs not needed for querying Web contents.

3. THE WEB QUERY LANGUAGE XCERPT

The Web query language Xcerpt is *embedded* in XChange. Xcerpt is a pattern and rule-based language for querying Web contents (i.e. persistent data). Xcerpt uses (query) *patterns* for querying Web contents, and (construction) *patterns* for constructing new data items. *Terms* are used for denoting query patterns (i.e. *query terms*), construction patterns (i.e. *construct terms*) and also for denoting data items of Web contents (i.e. *data terms*). Common to all terms is that they represent tree or graph-like structures. The children of a node may either be *ordered*, i.e. the order of occurrence is relevant, or *unordered*, i.e. the order of occurrence is irrelevant. In the term syntax (used in this article as it is more readable as the Xcerpt's XML syntax), an *ordered term specification* is denoted by square brackets [ ], an *unordered term specification* by curly braces {}.

Data Terms represent data items (i.e. XML documents) that are found on the Web. In an Xcerpt program the Web contents to be queried are specified using the keyword `resource` followed by the Web address(es) where the data is to be found.

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

At `http://airline.com`:

```
flights {
 last-changes{"2004-08-20"},
 currency{"EUR"},
 flight {
  number{"AI2011"},
  from{"Paris"},
  to{"Munich"},
  date{"2004-08-23"},
  departure-time{"10:30"},
  arrival-time{"12:00"},
  class{"economy"},
  price{"75"}
 },
 flight {
  number{"AI2021"},
  from{"Paris"},
  to{"Munich"},
  date{"2004-08-23"},
  departure-time{"17:30"},
  arrival-time{"19:00"},
  class{"economy"},
  price{"80"}
 },
...
}
```

At `http://hotels.net`:

```
accomodation {
 currency{"EUR"},
 hotels {
  city{"Paris"},
  country{"France"},
  hotel {
   name{"Ambassade"},
   category{"2 stars"},
   room-price{"62"},
   phone{"+331888219"},
   no-pets {}
   },
  hotel {
   name{"Winston"},
   category{"3 stars"},
   room-price{"60"},
   phone{"+331828156"}
   },
   hotel {
    name{"Royale"},
    category{"4 stars"},
    room-price{"120"},
    phone{"+331778123"}
     },
    ... },
    ... }
```

Query Terms are (possibly incomplete) patterns that are matched against Web contents represented by data terms. *Partial* (incomplete) or *total* (complete) query patterns can be specified. A query term t using a partial specification (denoted by *double* square brackets [[ ]] or curly braces {{}}) for its subterms matches with all such terms that (1) contain matching subterms for all subterms of t and that (2) might contain further subterms without corresponding subterms in t. In contrast, a query term t using a total specification (denoted by *single* square brackets [ ] or curly braces {}) does not match with terms that contain additional subterms without corresponding subterms in t. Query terms contain *variables* for selecting data items (i.e. subterms of data terms are to be bound to the variables). Variable restrictions can be specified using the $\leadsto$ construct (read *as*), which restricts the bindings of the variables to those terms that are matched by the restriction pattern.

Example 2. The following Xcerpt query term is used to query the data at `http://airline.com` about flights from Paris to Munich.

```
in { resource {"http://airline.com"},
   flights {{ var F ⇝ flight {{
                              from{"Paris"},to{"Munich"}  }}
  }}   }
```

Query terms are "matched" with data or construct terms by a non-standard unification method called *simulation unification* [8] dealing with partial and unordered query specifications.

Construct Terms serve to reassemble variables (the bindings of which are specified in query terms) so as to construct new data terms. They are similar to data terms, 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).

Construct-Query Rules (short rules) relate a construct term (introduced by the keyword `CONSTRUCT`) to a query (introduced by the keyword `FROM`) consisting of AND and/or OR connected query terms. Queries or parts of a query may be further restricted by arithmetic constraints in a so-called condition box (introduced by the keyword `where`).

Example 3. The following Xcerpt rule gathers informations about the hotels in Paris with a price limit.

```
CONSTRUCT
  answer [ all var H ]
FROM
 in { resource {"http://hotels.net"},
  accomodation {{
    hotels {{ city {"Paris"},
              var H⇝hotel {{ room-price{var P} }}  }}
  }}   } where var P < 90
END
```

An Xcerpt program consists of one or more rules. Xcerpt rules may be *chained* to form complex query programs, i.e. rules may query the results of other rules. More on Xcerpt can be found in [8] and at `http://xcerpt.org`.

4. XCHANGE: LANGUAGE CONSTRUCTS

4.1 Events and Event Messages

XChange offers *explicit* events, *implicit* events, and *system* events. *Explicit events* are explicitly raised by a user or by a (predefined) XChange program. They are raised at a Web site and sent to one or more (same or other) Web sites through *event messages*. *Implicit events* are local events not expressed through event messages (e.g. local updates of persistent data or system clock events). Events are transmitted from a Web site to another through event messages. Thus, such events are necessarily explicit. *System events* (e.g. system clock events) are events that are coming from the encompassing "system" and might be useful to handle together with explicit and/or implicit events. A system event might be explicit or implicit depending whether it is transmitted from a Web site to another or not.

Event messages communicate events between (same or different) Web sites. An XChange *event message* is an XML document with a root element labelled `event` and the four parameters (represented as child elements as they may contain complex content): `raising-time` (i.e. the time of the event manager of the Web site raising the event), `reception-time` (i.e. the time at which a site receives the event), `sender` (i.e. the URI of the site where the event has been raised), and `recipient` (i.e. the URI of the site where the event has been received). An event message is an envelope for an arbitrary XML content. Thus, multiple event messages can (but not necessarily) be nested making it possible to create trace histories. Note that XChange messages are compatible with the messages and the "message exchange patterns" of SOAP [11].

Example 4. Assume that a flight has been cancelled. The control point that has observed this event raises it and sends to `http://airline.com` the following event message:

```
event {
   sender {"control://controlpoint-A20"},
   recipient {"http://airline.com"},
   raising-time {"2004-08-23T12:00:25"},
   cancellation {
     flight {number{"AI2021"},date{"2004-08-23"} }  }
 }
```

XChange excludes broadcasting of event messages on the Web (i.e. sending event messages to all sites of a portion of the Web), since indiscriminate sending of event messages to many Web sites is not adequate for a non-centrally managed structure such as the Web.

4.2 Event Queries

The capability to detect and react to *composite events*, e.g. sequences of event instances that have occurred possibly at different Web sites within a specified time interval, is needed for many Web-based reactive applications. However, (to the best of our knowledge) existing languages for reactivity on the Web do *not* consider the issues of detecting and reacting to such composite events[1]. One of the novelties introduced by XChange is the processing of *composite events*. To this aim, XChange offers *composite event queries*. An XChange *event query* may be *atomic*, i.e. one event query term (similar to an Xcerpt query term), or *composite*. An *atomic event query* refers to one single event. Three dimensions are distinguished for *composite event queries*: *temporal range*, *event composition*, and *occurrence*.

Note that *composite event instances* (detected using composite event queries) do not have time stamps, like atomic event instances do. Instead, a composite event instance inherits from its components a beginning time (i.e. the reception time of the first received event instance that is part of

[1] [1] considers "composite events". However, this notion refers in [1] to updates of several elements of a single XML document. The XChange notion of composite events goes beyond such updates of an XML document.

the composite event instance) and an ending time (i.e. the reception time of the last received event instance that is part of the composite event instance).

Temporal Range. A time interval can be specified for (atomic or composite) event queries meaning that event instances are considered relevant only if they occur in the given time interval. Such a time interval has always a lower bound (the time point of event query definition, if not explicitly given) and might have an upper bound (i.e. the time interval is *finite* in the sense that it contains a finite number of reference granules [3]). Lower bounds are indispensable for efficiency reasons. They make it possible to release each event at each Web site after a finite time – thus keeping the distinction persistent vs. volatile data. Time intervals for event queries can be specified using the constructs: `within TimeInterval`, `before TimePoint`, `after TimePoint`, and `during Duration`.

Example 5. An XChange composite event query that detects insertion of discounts for flights from Paris to Munich, but only before 20th of August 2004:

```
event {{
  flight {{ from {"Paris"}, to {"Munich"},
           new-discount { var D } }}
   }} before "2004-08-20T10:00:00"
```

The temporal range constructs of XChange are aligned to the temporal and calendar type system currently developed for Xcerpt and XChange [3].

Event Composition. XChange has constructs for:

1. *temporally ordered conjunctions* of event queries (i.e. querying for successive, in terms of time, occurrences of events) – keyword `andthen`. A total specification (i.e. single square brackets) expresses that between any two component instances of a detected composite instance no other event instance occurred. In contrast, a partial specification (i.e. double square brackets) expresses that between two such component instances other events have (possibly) occurred.
2. *conjunctions* of event queries (i.e. querying for occurrences of events where the order in which event instances occur is not important) – keyword `and`.
3. *disjunctions* of event queries (i.e. querying for event instances - answers to one of the event queries) – keyword `or`, and *exclusive disjunctions* of event queries within a *finite* time interval – keyword `xor`.
4. *negations* of event queries within a *finite* time interval (i.e. querying for non-occurrence of event instances in the given finite time interval) – keyword `not`.
5. *overlapping* for *composite* event queries (i.e. querying for overlapping of composite event instances on the time axis of the incoming events) – keyword `overlap`.
6. *meet* for *composite* event queries (i.e. for an answer, the ending time of each component event instance is the beginning time of the successive component) – keyword `meets`.
7. *if-then-else* for event queries (i.e. *if* querying for an event A detects an instance of it, *then* query for an event B, *else* query for an event C). A generalisation of this control construct (i.e. a *case* construct) is also offered by XChange.
8. *arbitrary* event queries (i.e. each event instance would be an answer to such a query) – keyword `any`.

Example 6. An XChange composite event query for detecting the occurrence of a flight cancellation event and (as successive event) the non-occurrence (during two hours from the cancellation notification) of a notification of an accomodation granted by the airline:

```
andthen [
    event {{ cancellation {{
                 flight {{var Number}} }}  }},
    not event {{ sender {"http://airline.com"},
                   accomodation-granted {{}}  }}
   ] during "2 hours"
```

Occurrence. XChange event queries can query for *multiplicities*, e.g. to detect event istances that occur at least or at most a number of times in a given time interval, for *repetitions*, i.e. to detect every n-th event instance in a given time interval, and for *rankings*, i.e. to detect event instances having a given rank, or position, in the incoming "flow" of events.

Example 7. Mrs. Smith is on a business trip in Paris. She uses a travel organiser that plans her trips and reacts to happenings that might influence her schedule. The following travel organiser's event query is used to detect if she receives at least three important messages from her secretary.

```
event {{  secretary-message {{ important {{ }} }}
     }} at least "3" times
```

4.3 Transactions

An XChange transaction specification is a group of update specifications and/or explicit event specifications (expressing events that are constructed, raised, and sent as event messages) that are to be executed in an *all-or-nothing manner*. An XChange *update specification* is a (possibly incomplete) *pattern* for the data to be updated, augmented with the desired update operations. The notion of update terms is used to denote such patterns containing update operations for the data to be modified. An update term may contain different types of update operations.

Example 8. At `http://airline.com` the flight timetable needs to be updated as reaction to the event given in Example 4:

```
in { resource {"http://airline.com"},
 flights {{
   last-changes {var L replaceby "2004-08-23"},
   flight {{ number{"AI2021"}, date{"2004-08-23"},
      delete departure-time {{}},
      delete arrival-time {{}},
      insert news{"Flight has been cancelled!!"} }}
  }} }
```

Intensional updates, i.e. a description of updates in terms of (standard or event) queries, can be specified in XChange as the language inherits the querying capabilities of the language Xcerpt. This eases considerably the specification of updates, e.g. for specifying *modification* of the discounts for *all* flights offered by a specific airline.

As mentioned in Section 2.3 XChange supports *complex updates* (e.g. ordered conjunction of atomic or complex updates, meaning that all specified updates are to be executed and in the specified order). In XChange, the keywords `and` and `or` denote conjunction and disjunction of updates, respectively. Like Xcerpt, XChange uses square brackets and curly braces for expressing that the order of evaluation is of importance and of no importance, respectively.

4.4 (Re)Active Rules

An XChange program is located at one Web site and consists of one or more (re)active rules of the form *Event query – Standard query – Transaction/Raised events*. Every occurrence of an event is queried using the *event query* (introduced by keyword `ON`). If an answer is found and the *standard query* (introduced by keyword `FROM`) has also an answer, then the action is executed (i.e. a transaction is executed – keyword `TRANSACTION`, or explicit events are raised and sent to one or more Web sites – keyword `RAISE`). There are two kinds of XChange rules: *event-raising rules* (i.e. the head of the rules specifies explicit events) and *transaction rules* (i.e. the head of the rules specifies transactions).

Example 9. The site `http://airline.com` has been told to notify Mrs. Smith's travel organiser of delays or cancellations of flights she travels with:

```
RAISE
   event {
     recipient{"http://travelorganiser.com/Smith"},
     cancellation-notification { var F }  }
  ON
   event {{
     sender{"http://airline.com"},
     cancellation {{
       var F⇝flight {{ number{"AI2021"},
                         date {"2004-08-23"} }}  }}
  }}
  END
```

Example 10. The travel organiser of Mrs. Smith uses the following rule: if the return flight of Mrs. Smith is cancelled then look for and book another suitable flight. The rule is specified in XChange as:

```
TRANSACTION
  in { resource {"http://airline.com/reservations/"},
    reservations {{
     insert reservation{var F,name{"Christina Smith"} }
    }}
  }
ON
  event {{ sender {"http://airline.com"},
    cancellation-notification {{
      flight {{ number{"AI2021"},date {"2004-08-23"} }}
    }} }}
FROM
    in { resource {"http://airline.com"},
      flights {{
        var F ⇝ flight {{
             from {"Paris"}, to {"Munich"},
             date{"2004-08-23"}, departure-time{var T}  }}
     }}  } where var T after "14:00"
END
```

Example 11. If no other suitable return flight is found and the airline does not provide an accomodation, then book for Mrs. Smith a cheap hotel and inform her husband about the changes in her schedule:

```
TRANSACTION
 and [
  in {resource{"http://hotels.net/reservations/"},
    reservations {{
     insert reservation{var H,name{"Christina Smith"},
            from{"2004-08-23"}, until{"2004-08-24"} }
    }} } },
  in {resource{"address-book://addresses/my-husband"},
   addresses {{
     insert my-hotel { phone { var Tel },
       remark {"I'm staying in Paris over night!"}}
    }} } }
   ]
ON
 andthen [
   event {{ sender {"http://airline.com"},
            cancellation-notification {{
                flight {{ number {"AI2021"},
                    date {"2004-08-23"} }} }}
   }},
   not event {{
     sender {"http://airline.com"},
     accomodation-granted {{ hotel {{}} }}
  }}
 ] during "2 hours"
FROM
 in { resource {"http://hotels.net"},
   accomodation {{
     hotels {{ city {"Paris"},
          var H⇝hotel {{ room-price{var P},
                          phone{var Tel} }} }}
    }}  } where var P < 90
END
```

5. XCHANGE: PROJECT STATUS

The XChange project has started ten months ago. As explained in Section 2.5 XChange builds upon the Web query language Xcerpt [8]. A first version of Xcerpt is fully designed and a reference implementation is available (cf. http://xcerpt.org). Currently, the design of an extended core language for XChange is completed and a first (reference) implementation extending that of Xcerpt has begun.

6. CONCLUSION

XChange has been conceived not only for standard Web but also for Semantic Web applications. Such applications of XChange are presented and discussed in [2]. Reactivity on the Web is an emerging issue essential for Semantic Web, Web services, Grids as well as many other Web-based systems. The XChange project aims at contributing to this issue with a high-level programming language.

7. REFERENCES

[1] M. Bernauer, G. Kappel, and G. Kramler, *Composite Events for XML*, Int. Conf. on World Wide Web, 2004.

[2] F. Bry, T. Furche, P.-L. Pătrânjan, and S. Schaffert, *Data Retrieval and Evolution on the (Semantic) Web: A Deductive Approach*, Workshop on Principles and Practice of Semantic Web Reasoning, Springer, 2004.

[3] F. Bry and S. Spranger, *Towards a Multi-Calendar Temporal Type System for (Semantic) Web Query Languages*, Workshop on Principles and Practice of Semantic Web Reasoning, Springer, 2004.

[4] I. Foster, C. Kesselman, and S. Tuecke, *The Anatomy of the Grid. Enabling Scalable Virtual Organizations*, Int. Journal of Supercomputer Applications, 2001.

[5] M. Liu, L. Lu, and G. Wang, *A Declarative XML-RL Update Language*, Int. Conf. on Conceptual Modeling (ER 2003), Springer, 2003.

[6] W. May and E. Behrends, *On an XML Data Model for Data Integration*, Workshop on Foundations of Models and Languages for Data and Objects, 2001.

[7] G. Papamarkos, A. Poulovassilis, and P.T. Wood, *Event-Condition-Action Rule Languages for the Semantic Web*, Workshop on Semantic Web and Databases, 2003.

[8] S. Schaffert and F. Bry, *Querying the Web Reconsidered: A Practical Introduction to Xcerpt*, Int. Conf. Extreme Markup Languages, 2004.

[9] J.D. Ullman, *Principles of Database and Knowledge-base Systems*, vol. 1, Computer Science Press, 1988.

[10] W3 Consortium, *XQuery: A Query Language for XML*, 2001.

[11] W3 Consortium, *SOAP Version 1.2 Part 1: Messaging Framework*, 2003.