

# XChange: A Reactive, Rule-Based Language for the Web

François Bry  
University of Munich  
Institute for Informatics  
Oettingenstr. 67, D-80538 München  
Email: bry@pms.ifi.lmu.de

Michael Eckert  
University of Munich  
Institute for Informatics  
Oettingenstr. 67, D-80538 München  
Email: eckert@pms.ifi.lmu.de

Paula-Lavinia Pătrânjan  
University of Munich  
Institute for Informatics  
Oettingenstr. 67, D-80538 München  
Email: patranjan@pms.ifi.lmu.de

**Abstract**—Reactivity on the Web is an emerging research issue covering: updating data on the Web, exchanging information about events (such as executed updates) between Web sites, and reacting to combinations of such events. Reactivity plays an important role for upcoming Web systems such as online marketplaces, adaptive Web and Semantic Web systems, as well as Web services and Grids. This article introduces the high-level language XChange for programming reactive behavior and distributed applications on the Web.

## I. INTRODUCTION

Many resources on the Web and Semantic Web are dynamic in the sense that they can change their content over time as new information comes in or information becomes out-of-date. Often, changes must be mirrored by other Web resources by propagating updates.

Reactivity on the Web is the ability of Web sites to detect happenings, or events, of interest that have occurred on the Web and to automatically react to them through reactive programs. Events may have various levels of abstraction ranging from low-level, such as insertions into XML or RDF documents, to high-level application-dependent ones. For example, in a tourism application, events of interest include delays or cancellations of flights, and new discounts for flights offered by an airline. Reactions to such events include notifying colleagues about delays, looking for and booking another flight, or booking flights from a particular airline.

This article presents the reactive, rule-based language XChange [BBEP05], [BEP06a], [BEP06b], [XCh], which provides the following benefits over the conventional approaches of using general-purpose programming languages to implement reactive behavior on the Web:

- XChange reactive rules are highly declarative. They allow programming on a high abstraction level, and are easy to analyze for both humans and machines.
- The various parts of a rule all follow the same paradigm of specifying patterns for XML data, thus making XChange an elegant, easy to learn language.
- Both atomic and composite events can be detected and relevant data extracted from events. Composite events, temporal combinations of events, are an important requirement in composing an application from different services.

- XChange embeds an XML query language, Xcerpt [SB04], [Xce], allowing to access and reason with Web resources.
- XChange provides an integrated XML update language for modifying Web resources.
- XChange reactive rules enforce a clear separation of persistent data (Web resources) and volatile data (events). The distinction is important for programmers: the former relates to state, while the latter reflects changes in state.
- XChange’s high abstraction level and its powerful constructs allow for short and compact code.

The remainder of this article is structured as follows. We first introduce the paradigms driving the design of XChange (Section 2). Next we give a flavor of XChange programs (Section 3). We conclude the article with an outlook on future research directions (Section 4).

## II. PARADIGMS

Clear paradigms that a programming language follows provide a better language understanding and ease programming. This section introduces the paradigms upon which XChange is designed.

*Event vs. Event Query:* An event is a happening to which each Web site may decide to react in a particular way or not to react at all. In order to notify Web sites about events and to process event data, events need to have a data representation. In XChange, events are represented as XML documents.

Event queries are queries against event data; they serve a double purpose: detecting events of interest and (through composite event queries) temporal combinations of them and selecting data items from the representation of events.

*Volatile vs. Persistent Data:* XChange reflects the novel view over Web data that differentiates between volatile data (event data communicated on the Web between XChange programs) and persistent data (data of Web resources such as XML or HTML documents). This clear distinction between volatile and persistent data aims at easing programming and avoiding the emergence of a parallel “Web of events.”

*Rule-Based Language:* An XChange program is located at one Web site and contains reactive rules, more precisely Event-Condition-Action rules (ECA rules) of the form Event Query — Web Query — Action. Such an ECA rule has the

```

ON
  xchange:event {{
    flight-cancellation {{
      flight-number{var N},
      passenger{{ name {"Christina Smith"} }} }} }}
FROM
  in { resource { "http://www.example.com/lufthansa.xml", "xml" },
    flights {{
      flight {{ number { var N } }} }} }
DO
  xchange:event [
    xchange:recipient [ "http://sms-gateway.org/us/206-240-1087/" ],
    text-message [ "Your flight", var N, "has been cancelled." ] ]
END

```

Fig. 1. ECA rule to notify Mrs. Smith of a flight cancellation

following meaning: When events answering the event query are received and the Web query is successfully evaluated, the action is performed. Both event query and Web query can extract data through variable bindings, which can then be used in the action. XChange embeds the Web query language Xcerpt (developed in the REVERSE working group I4 on “Reasoning-aware Querying”) for expressing Web queries and for specifying deductive rules (e.g., for views over heterogeneous data, and advanced reasoning tasks).

*Pattern-Based Approach:* Event queries, Web queries and actions follow the same approach of specifying patterns for data queried, updated, or constructed.

*Push Strategy for Event Communication:* XChange uses a push strategy for communicating events to interested Web sites. This has several advantages over pull communication: it allows faster reaction, avoids unnecessary network traffic through periodic polling, and saves local resources.

*Processing of Events:* Event queries are evaluated locally at each Web site against the stream of incoming events. For efficiency reasons, an incremental evaluation is used for (composite) event queries.

*Bounded Event Lifespan:* Event queries are such that no data on any event has to be kept forever in memory, that is, the event lifespan is bounded. Hence, design enforces that volatile data remains volatile.

### III. A FLAVOR OF XCHANGE

An XChange program is located at one Web site and consists of one or more (re)active rules of the form Event query — Web query — Action. Every incoming event is queried using the event query (introduced by keyword ON). If an answer is found and the Web query (introduced by keyword FROM) has also an answer, then the specified action (introduced by keyword DO) is executed.

Rule parts communicate through variable substitutions. Substitutions obtained by evaluating the event query can be used in the Web query and the action part, those obtained by evaluating the Web query can be used in the action part.

*Example:* The site <http://airline.com> has been told to notify Mrs. Smith’s travel organizer by SMS of delays or cancellations of flights she travels with. This can be expressed as XChange rule in Figure 1. Note that both data from the event (number of the canceled flight) and an XML document on the Web are accessed.

We now look closer at the three parts of an XChange rule, starting for ease of presentation with Web queries in the condition part.

*Web Queries:* The condition part of XChange rules queries data from Web resources such as XML documents or RDF documents. Such Web queries are expressed in Xcerpt, a Web and Semantic Web query language. Xcerpt has query patterns, called query terms, for querying Web resources, and construction patterns, called construct terms, for re-assembling data selected by queries into new data items. Only query terms are used in the condition part of XChange ECA rules; however, XChange programs can contain deductive rules expressed as CONSTRUCT *construct-term* FROM *query-term* END.

Such deductive rules are similar to views in relational databases, and data derived by them can “feed” into other deductive rules and into the condition part of ECA rules.

For conciseness, XChange and Xcerpt represent data, query patterns, and construction patterns in a term-like syntax. The following depicts an XML document with information about flights, its representation as data term, and a query extracting a flight number.

```

<flight-cancellation>
  <number>UA917</number>
  <date>2006-02-20</date>
</flight-cancellation>

flight-cancellation [
  number ["UA917"],
  date ["2006-02-20"]
]

flight-cancellation {{
  number [ var N ]
}}

```

```

in { resource { "http://airline.com" },
    flights {{
        last-change { var L replaceby "2006-02-20" },
        flight {{
            number { var N },
            delete departure-time {{ }},
            delete arrival-time {{ }},
            insert news { "Flight has been cancelled!!" } }} }}

```

Fig. 2. Update to the flight database at <http://airline.com>

In the term syntax, square brackets denote that the order of the children of an XML element is relevant, curly braces denote that the order is not relevant.

Both partial (i.e. incomplete) or total (i.e. complete) query patterns can be specified. A query term  $t$  using a partial specification denoted by double brackets or 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 subterms of data terms that are bound to the variables.

The results of a query are bindings for the free variable in that query. In the example,  $N$  is bound to "UA917".

*Event Queries:* Events are represented as XML messages in XChange, and these event messages are exchanged between Web sites. Each Web site monitors the incoming event messages to check if they match an event query of one of its XChange rules.

Atomic event queries detect occurrences of single, atomic events. They are query patterns for the XML representation of the events. This means that the same pattern-based approach is used to query Web data and data in atomic events.

Often, situations that require a reaction by a rule are not given by a single atomic event, but a temporal combination of events, leading to the notion of composite events and composite event queries. Support for composite events is very important for the Web: In carefully developed applications, designers have the freedom to choose events according to their goal. They can thus often do with only atomic events by representing events which might be conceptually composite with a single atomic event. In the Web's open world, however, many different applications which have not been engineered together are integrated and have to cooperate. Situation that require a reaction might not have been considered in the original design of the applications and thus have to be inferred from many atomic events.

A composite event query consists of (1) a connection of (atomic or composite) event queries with event composition operators and (2) an optional temporal range limiting the time interval in which events are relevant.

Composition operators are denoted with keywords such as *and* (both events have to happen), *andthen* (the events

have to happen in sequence), *or* (either event can happen), *without* (non-occurrence of the event in a given time frame). Limiting temporal ranges can be specified with keywords such as *before* (all events have to happen before a certain time point), *in* (all events have to happen in an absolute time interval), *within* (all events have to happen within a given length of time).

*Actions:* The Web is a dynamic, state-changing system. To act in this world, XChange rules support the following primitive actions: executing simple updates to persistent Web data (such as the insertion of an XML element) and raising new events (i.e., sending a new event message to a remote Web site or oneself). To specify more complex actions, compound actions can be constructed as from the primitive actions.

*Updating Web Data:* An XChange update term is a (possibly incomplete) pattern for the data to be updated, augmented with the desired update operations. An update term may contain different types of update operations: An insertion operation specifies an Xcerpt construct term that is to be inserted, a deletion operation specifies an Xcerpt query term for deleting all data terms matching it, and a replace operation specifies an Xcerpt query term to determine data terms to be modified and an Xcerpt construct term as their new value.

The example in Figure 2 updates the flight timetable at <http://airline.com> (in reaction to the flight cancellation event seen earlier). The variable  $N$  is already bound to the flight number of the canceled flight.

*Raising New Events:* Events to be raised are specified as (complete) patterns for the event messages, called event terms. An event term is simply an Xcerpt construct term restricted to having a root labeled event and at least one sub-term recipient specifying the URI of the recipient.

The following is an example of an event term to notify a passenger on his PDA about the flight cancellation:

```

xchange:event {
  xchange:recipient {
    "http://www.johnqpublic.com/pda/" },
  cancellation-notification { var N }
}

```

*Specifying Complex Actions:* The primitive actions described by update terms and event terms alone do not let you do very much; only in their combination they can become powerful. XChange hence allows specifying complex actions as combinations of (primitive and complex) actions. Such

```

ON
  flight-cancellation {{
    number [var N]   }}
FROM
  in { resource { "http://airline.com/passengers.xml" },
      passengers {{
        booked-for { var N },
        name { var P },
        contact { var C }   }}   }
DO
  and {
    in { resource { "http://airline.com/waitinglist.xml" },
        waitinglist {{
          flight {{
            replaces { var N },
            insert all passenger{ var P }   }}   }}   },
    xchange:event {
      all xchange:recipient { var C },
      message ["Your flight ", var N, " has been cancelled. ",
              "You have been placed on the waiting list."],
      waitinglist {"http://airline.com/waitinglist.xml"}   }
  }
END

```

Fig. 3. Example of a complete XChange rule

combination of actions is to be executed in a transactional all-or-nothing manner.

Actions can be combined with disjunctions and conjunctions. Disjunctions specify alternatives, only one of the specified actions is to be performed successfully. (Note that actions such as updates can be unsuccessful, i.e., fail). Conjunctions in turn specify that all actions need to be performed. The combinations are indicated by the keywords `or` and `and`, followed by a list of the actions enclosed in braces or brackets.

The list of the actions can be ordered (indicated by square brackets []) or unordered (indicated by curly braces {} ). If the actions are ordered, their execution order is specified to be relevant. If the actions are unordered, their execution order is specified as irrelevant, thus giving more freedom for parallelization.

*Putting It All Together: An Example:* Having seen the three parts of an XChange rule, events, conditions, and actions, Figure 3 gives an example of a complete XChange rule. The rule below reacts upon the flight cancellation (event), extracts as a query to Web data the affected passengers (condition), and notifies the affected passengers (event raising action) as well as placing them on a waiting list (update action).

#### IV. CONCLUSIONS

This article has presented the high-level language XChange for realizing reactivity on the Web. XChange introduces a novel view over the Web data by stressing a clear separation between persistent data (data of Web resources, such as XML or HTML documents) and volatile data (event data communicated on the Web between XChange programs). XChange's language design enforces this clear separation.

XChange is an ongoing research project [XCh]. The design, the core language constructs, and the semantics are completed and a proof-of-concept prototype has been implemented. An implementation of use cases with XChange indicates the language's applicability and relative ease of use [Rom06].

Issues deserving further attention in XChange are automatic generation of ECA rules (e.g., from data dependency specifications), efficient evaluation of rule sets and in particular event queries, visual rendering of XChange programs, and means to structure and organize large rule programs [BE06].

#### REFERENCES

- [BBEP05] James Bailey, François Bry, Michael Eckert, and Paula-Lavinia Pătrânjan. Flavours of XChange, a rule-based reactive language for the (Semantic) Web. In *Proc. Int. Conf. on Rules and Rule Markup Languages for the Semantic Web*, number 3791 in LNCS, pages 187–192. Springer, 2005.
- [BE06] François Bry and Michael Eckert. Twelve theses on reactive rules for the Web. In *Workshop "Reactivity on the Web" at Int. Conf. Extending Database Technology*, 2006. (Invited paper).
- [BEP06a] François Bry, Michael Eckert, and Paula-Lavinia Pătrânjan. Querying composite events for reactivity on the Web. In *Proc. Intl. Workshop on XML Research and Applications*, number 3842 in LNCS, pages 38–47. Springer, 2006.
- [BEP06b] François Bry, Michael Eckert, and Paula-Lavinia Pătrânjan. Reactivity on the Web: Paradigms and applications of the language XChange. *Journal of Web Engineering*, 5(1):3–24, 2006.
- [Rom06] Inna Romanenko. Use cases for reactivity on the Web: Using ECA rules for business process modeling. Master's thesis, Institute for Informatics, University of Munich, 2006.
- [SB04] Sebastian Schaffert and François Bry. Querying the Web reconsidered: A practical introduction to Xcerpt. In *Proc. of Extreme Markup Languages Conf.*, 2004.
- [Xce] Xcerpt. <http://xcerpt.org>.
- [XCh] XChange. <http://www.pms.ifi.lmu.de/projekte/xchange>.