Composite Event Queries for Reactivity on the Web

James Bailey
Dept. of Computer Science
University of Melbourne
Victoria, 3010
Australia
jbailey@cs.mu.oz.au

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
patranja@pms.ifi.lmu.de

ABSTRACT

Reactivity on the Web is an emerging issue. The capability to automatically react to events (such as updates to Web resources) is essential for both Web services and Semantic Web systems. Such systems need to have the capability to detect and react to complex, real life situations. This presentation gives flavours of the high-level language *XChange*, for programming reactive behaviour on the Web.

Categories and Subject Descriptors

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

Keywords

Web, reactive languages, event-condition-action rules, composite events

1. INTRODUCTION

Reactivity on the Web is gaining importance and is recognised as a solution for many everyday life problems. For example, a Web service provided by an airline could report delays on departures or arrivals, and flight cancellations. A Web-based personalised organiser might be conceived so as to automatically react to (possibly combinations of) reports which affect its owner. Such reports can be sent from different Web services (like a weather forecast service). A delayed arrival might cause either an email to be sent to some other person or the cancellation of a hotel reservation. Reactive languages formerly developed for the Web support *simple* update operations on XML documents, i.e. there is no support for specifying and executing (two or more) updates in a desired order and in an *all-or-nothing* manner. Moreover, these languages have the capability to react only to single event instances and do not provide constructs for querying for complex combinations of event instances.

The issue of reacting to so-called *complex events*, i.e. (possibly time-related) combinations of event instances, has received considerable attention in the field of active databases (cf. e.g. [3]). Thus, useful concepts can be "borrowed" from active databases when investigating reactivity on the Web. However, differences between (generally centralised) active databases and the Web, where a central clock and a central management are missing, necessitate new approaches. In particular, complex events reflecting a user-centered (and not a system-centered) view are needed for the Web. One such approach is proposed by the language *XChange* [1] and is introduced in this presentation. XChange builds upon the Web query language Xcerpt [2] and provides constructs for detecting complex (or composite) events on the Web.

[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`).

WWW 2005, May 10–14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

2. EVENTS AND EVENT QUERIES FOR REACTIVITY ON THE WEB

2.1 Atomic and Composite Events

Informally, an *atomic event* is a happening (e.g. an update of a possibly remote Web resource) to which each Web site (through a reactive program) may decide to react in a particular way or not to react to at all. XChange distinguishes between two kinds of atomic events: *explicit* events and *implicit* events. *Explicit events* are explicitly raised by a user or by a (predefined) XChange program. They are raised at a Web site and sent internally or to other Web sites through *event messages*. *Implicit events* are local events not expressed through event messages (e.g. local updates of data or system clock events). Events are transmitted from one Web site to another through event messages. Thus, an event sent from one Web site to another is necessarily explicit.

Composite events are defined in XChange through *composite event queries* (see Section 3) – they are *answers* to composite event queries. This is a novel way of defining composite events, but the authors consider it the only intuitive one.

2.2 Event Query vs. Web Query

Volatile vs. Persistent Data. An important distinction is made between *persistent data* (data of Web resources) and *volatile data* (events). To query persistent data, *standard queries* (like Xcerpt queries [2]) are used. To query volatile data, *event queries* are used. Standard and event queries can be very similar. However, event queries are more likely to refer to time or event sequences.

Incremental Aspects. Event queries need to be evaluated in an *incremental* manner, as data (events) that are

queried are received in a stream-like manner and are not persistent.

2.3 Metaphor: Speech vs. Written Text

The metaphor of XChange for reactivity on the Web is that of *speech* for volatile data and *written text* for persistent data. Speech cannot be modified. If one has communicated some information in this way one can correct, complete, or invalidate what one has told – through further speech. In contrast, written text can be updated in the usual sense. Likewise, volatile data (events) is *not* updatable but persistent data (Web content) is updatable. To inform about, correct, or invalidate former volatile data, new *event messages* (see below) are communicated between Web sites

2.4 Communication of Events

Event Messages. *Event messages* communicate information about events between the same or different Web sites. An XChange *event message* is an XML document containing at least information about the *sender*, the *recipient*, the *raising* and *reception times* of the event message.

Peer-to-Peer. For communicating data between Web sites XChange uses the *peer-to-peer* communication model, that is all parties have the same capabilities and every party can initiate a communication session.

Push Strategy. For propagating events on the Web, two strategies are possible: the *push* strategy, where a Web site informs possibly interested Web sites about events, and the *pull* strategy, where interested Web sites query periodically persistent data found at other Web sites in order to determine changes. Both strategies are useful. As the pull strategy is supported by languages that query persistent data, XChange offers the *push* strategy.

2.5 Local Control of Event Memorisation

An essential aspect of XChange is that each Web site controls its own event memory usage. In particular, the size of the event history kept in memory depends only on the event queries posed at this Web site. The time period for which an atomic event is kept in memory at a Web site is automatically detected from the event queries locally posed. By design, XChange composite event queries are such that no data on any event need to be kept for ever in memory. If this is necessary for some applications, events should be explicitly stored as persistent data.

3. COMPOSITE EVENT QUERIES

An XChange *event query* may be *atomic* or *composite*. An *atomic event query* refers to one single event, it represents a pattern for the single incoming event that is of interest. *Composite event queries* are offered by XChange for detecting *composite events*, a novel language ability. Two dimensions are distinguished for *composite event queries*: *temporal range* and *event composition*.

Temporal Range. A *time interval* can be specified for event queries, restricting relevant event query instances to those occuring in the given time interval. Such a time interval always has a lower bound (the time point of event query definition, if not explicitly given) and an upper bound (the time interval is *finite*). These bounds make it possible to release each event at each Web site after a finite time. Time intervals can also be specified in XChange as relative to occurrences of other events, by means of *durations*.

Event Composition. Real application scenarios have determined the introduction into the language of a set of constructs along the event composition dimension. For example, constructs are offered for detecting temporally ordered *conjunctions* of events (e.g. flight cancellations followed by a notification saying that no accommodation is granted by the airline), *all* events that have occurred between occurrences of other events, *disjunctions* of events, *exclusion* (negation) of events, or *quantified occurrences* of events (e.g. detecting every second email from my secretary).

Processing of Event Queries. XChange assumes no central *processing of event queries* as such an approach is not suitable on the Web. Instead, event queries are processed locally at each XChange-aware Web site by means of an *event manager* with ability to also *release* event query instances after a finite time.

4. TRANSACTIONS AND REACTIVE RULES IN XCHANGE

Complex Updates. An *elementary update* is a change (insert, delete, replace) to a persistent data item (XML or RDF document). *Complex updates* expressing ordered or unordered conjunctions, or disjunctions of updates are also offered by XChange. 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.

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.

(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*. If an answer is found and the *standard query* (i.e. Xcerpt query) also has an answer, then the action is executed (i.e. a transaction is executed or explicit events are raised and sent to one or more Web sites).

5. CONCLUSION

This presentation has introduced the language XChange for reactivity on the Web focussing on the ability of the language to detect composite events on the Web. A more detailed discussion on XChange's constructs can be found in [1] and on the query language XChange is integrating, the Xcerpt language, in [2] .

6. REFERENCES

[1] F. Bry and P.-L. Pătrânjan. Reactivity on the Web: Paradigms and Applications of the Language XChange. In *20th Annual ACM Symposium on Applied Computing (SAC'2005)*. ACM Press, 2005.

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

[3] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.