Flavours of XChange, a Rule-Based Reactive Language for the (Semantic) Web

James Bailey[1], François Bry[2], Michael Eckert[2], and Paula-Lavinia Pătrânjan[2]

[1] NICTA Victoria Laboratory, Dept. of Computer Science and Software Engineering, University of Melbourne, 3010, Australia http://www.cs.mu.oz.au/~jbailey/
[2] University of Munich, Germany, http://pms.ifi.lmu.de/

Abstract. This article introduces XChange, a rule-based reactive language for the (Semantic) Web. Stressing application scenarios, it first argues that high-level reactive languages are needed for both Web and Semantic Web applications. Then, it discusses technologies and paradigms relevant to high-level reactive languages for the (Semantic) Web.

1 Introduction

A common perception of the Web is that of a distributed repository of hypermedia documents, with clients (in general browsers) that download documents, and servers that store and update documents. This perception is not fully accurate: Many Web applications rely on the updating of server data in response to requests of clients, or client data in response to requests of servers. This article first argues that complementing HTTP, the Web's communication infrastructure, with high-level languages for updates and reactivity is needed for both standard Web and Semantic Web applications. It then introduces XChange, a novel high-level language for updates and reactivity on the (Semantic) Web based on Event-Condition-Action rules.

Many Web applications build upon servers that update data according to client requests or actions. This is the case of e-Commerce systems that receive, process and buy orders, of e-Learning systems that select and deliver teaching materials depending on students' test performances, and of communication platforms such as wikis, where several users modify the same documents. Conversely, some Web applications also build upon clients that update data according to server requests. This is the case with so-called *cookies,* i.e. descriptions on a client of the states of a connection to a server, or when a client keeps, after a connection to a server, data collected during the connection, e.g. a railway or airline electronic ticket.

Many Web applications also build upon complex reactions to messages or events, exchanged not only between clients and servers but also (via servers) between clients. This is the case of Web-based communication platforms (contributors are informed of other contributors joining in or leaving a session), of Web-based business management systems (business travel applications, planning and reimbursement in large companies rely upon complex work-flows of actions and messages), of Web-based systems offering context-dependent services (e.g. a time and location dependent car park directory adapting the information it delivers and reacting to changes such as clients changing places and car parks announcing their free parking capacities), etc.

Updates and reactivity are as much “Semantic Web issues”, as they are “standard Web issues”. The applications mentioned above might involve both standard Web and Semantic Web data such as HTML, XML, RDF, Topic Maps, and OWL data, as well as inference from RDF triples.

Updates and reactivity on the Web are realised using HTTP/1.1; its communication paradigm is a *client-server model* of *request-response interactions*. Message *headers* give rise to the specification of network communication and system parameters. Although HTTP provides a communication infrastructure to implement updates and reactivity on the Web, more abstract and higher-level languages are needed that i) abstract away network communication and system issues, ii) ease the specification of complex updates of Web resources (e.g. XML, RDF, and OWL data), iii) are convenient for specifying complex flows of actions and reactions on the Web.

2 Technologies and Paradigms

Atomic Events, Event Messages, and Composite Events. Web applications require a number of different kinds of *atomic events:* i) events exchanged between Web nodes for a node to trigger reactions at remote nodes, ii) events local to a node, to help express local reactivity, e.g. local updates, and iii) system events for reacting to the functioning, or non-functioning, of the encompassing “system(s)”. A natural assumption is that events exchanged between nodes are expressed in XML as *event messages*. Reacting to *complex events* is essential. Complex events have received considerable attention in active databases [12, 11]. However, differences between (generally centralised) active databases and the Web, where central clock and management are missing, message deliveries between Web nodes can be delayed, and user-centered (instead of system-centered) paradigms are expected, necessitate new approaches.

Temporal Dependencies often have to be expressed when composite events are specified, e.g. “depend on an event E_1 occurring *before* an event E_2”, or “depend on an event E occurring within a time interval I”. Sophisticated temporal notions and temporal event composition constructs are needed.

Event Messages vs. Web Resources. Event messages (volatile data)and standard Web resources (persistent data) should be kept in two separate data kinds, since otherwise the development of the “reactive” Web may be insufficiently distinguishable from the (Semantic) Web.

Event-Condition-Action Rules (*ECA rules*) fit well with the widespread and intuitive view of the Web as a distributed repository of documents. Indeed, ECA rules build on queries by the use of “conditions”. Thus, ECA rules building on a Web or Semantic Web query language are a natural paradigm for reactivity on the Web.

Distributed Processing and Communication. On the Web, reactive programs call for distributed processing. Reactive languages making each node capable of controlling its own reactive behaviour fit the decentralised management of the (Semantic) Web.

3 XChange in a Nutshell

XChange is a language of *ECA rules.* Each rule consists of three parts: i) an *"event"*-part, more precisely an event query, accessing event messages and (local) system events, ii) a Web query referred to as the *"condition"* accessing standard Web data, and iii) an *"action"* expressing iii.a) single updates, iii.b) messages to be sent to Web nodes, or iii.c) transactions i.e. a group of actions to be realised in an all-or-nothing manner. Figure 1 presents an XChange rule sending SMS notifications of delayed flights.

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

Fig. 1. An XChange ECA rule

The *atomic events* of XChange are happenings (e.g. an update of a possibly remote Web resource) to which each Web node (through a reactive program) may or may not react. XChange has *explicit events* and *implicit events. Explicit events* are raised by a user or by an XChange program at a Web node and sent to this and/or other Web nodes as *event messages.* XChange's event messages are (arbitrary) XML documents within an *event message envelope* expressed itself as a (specific) XML document.

Figure 2 presents an XChange event message in the *term syntax* of Xcerpt [4, 2] and XChange; the (arbitrary) content is surrounded by a fixed envelope. Nesting messages with their former envelopes makes it possible to track the origin of messages, removing envelopes before forwarding messages hides their origin.

Implicit events are local events such as updates of Web resources and system events. Events are transmitted from one Web node to another via event messages. Thus, an event sent from one Web node to another is necessarily explicit. *Composite events* are defined in XChange as *answers* to composite *event queries*, cf. [3, 7]

XChange makes a strict distinction between *persistent data,* i.e. Web resources, and *volatile data,* i.e. *by definition* events. XChange relies on the query language Xcerpt [2, 13] for accessing persistent data i.e. Web resources. XChange uses a novel query language especially tuned to events for accessing volatile data, i.e. events. This *event query language* [7] builds upon Xcerpt and extends it with constructs for *temporal event composition.* Event messages can be turned into Web resources, and Web resources might be included in event messages.

```
xchange:event [
   xchange:sender { "http://www.pms.lmu.de/" },
   xchange:recipient { "http://ruleml.org" },
   xchange:recipient { "http://www.cs.mu.oz.au/~jbailey/" },
   xchange:raising-time { "2005-06-29T18:15:00" },
   info { "Here is an article for RuleML'06!" },
   article [ title {"Flavours of XChange"}, authors [...], body [...] ]
]
```

Fig. 2. An XChange Event Message

XChange's *communication model* is *peer-to-peer,* i.e. all Web nodes have the same communication capabilities and every party can initiate a communication with every other Web node. Two basic *communication strategies* are possible on a network: a *push strategy* where senders inform recipients of messages they want to send to them, and a *pull strategy* where (potential) recipients keep querying all (potential) senders for messages. Arguably, the pull strategy is convenient for querying (persistent) Web resources, while the push strategy is convenient for querying (volatile) events. XChange relies on the push strategy for event queries and on the pull strategy for Web resource queries. XChange's message communication is *asynchronous,* i.e. XChange's 'send operation' is *non-blocking:* the execution of an XChange program immediately continues after a 'send operation' without waiting for the message transmission, an acknowledgment of receipt, or a reply. Note that blocking sending can easily be implemented using XChange rules.

XChange programs are processed in a *distributed* manner. Each (XChange-aware) Web node processes, possibly by delegation to another Web node, the XChange programs locally specified. XChange relies neither on "super-peers", nor on central services, such as a central synchronisation point.

XChange ensures a *local control of events*, as well as of *event memorisation.* A Web node might reject an update request from a remote Web node (sent in an event message), e.g. because of a lack of credentials. Furthermore, the events memorised at a Web node only depend on the XChange event queries posed *at that node.* The time during which an atomic event, e.g. an event message or a local implicit event, is kept in memory at a Web node only depends on the event queries posed *at that node.* By design, XChange composite event queries can be evaluated without keeping any event forever in memory.

XChange event queries have a *declarative semantics.* XChange event query evaluation is *data-driven,* incoming events are used for *incrementally evaluating* queries. In contrast, the evaluation of queries against Web resources, e.g. Xcerpt queries and XChange conditions, is in general query-driven.

4 Related Work And Conclusion

Allen's Temporal Relations [1] and the composite events of *Active Databases* [6, 5, 8] have been important inspirations in defining XChange's temporal event composition operators. As opposed to XChange, *high-level reactive languages for the Web* formerly developed, e.g. [10], support only *simple* update operations on XML (and RDF) documents. They offer no means to specify several updates to be executed in a given order or in an *all-or-nothing* manner. Another related system is Xyleme [9], a system for monitoring and subscription on the Web with 'alerters' monitoring simple updates of Web resources and a 'monitoring query processor' for complex event detection. Its reactive functionality is highly tuned to its specific application field.

XChange significantly differs from and/or extends over the above-mentioned approaches with i) its structured event messages, ii) its distinction between Web resources and event messages, iii) its logical variables possibly shared by its event queries, conditions, and actions, iv) its declarative semantics, and v) its communication and distributed processing models.

Acknowledgments 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 (http://rewerse.net)

References

1. J. F. Allen. Maintaining Knowledge About Temporal Intervals. *Comm. ACM*, 26:832–843, 1983.
2. J. Bailey, F. Bry, T. Furche, and S. Schaffert. Web and Semantic Web Query Languages: A Survey. In *Reasoning Web*, LNCS 3564. Springer-Verlag, 2005.
3. F. Bry and P.-L. Pătrânjan. Reactivity on the Web: Paradigms and Applications of the Language XChange. In *Proc. 20th ACM Symp. Applied Computing*, 2005.
4. F. Bry and S. Schaffert. A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In *Proc. Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.
5. A. Buchmann, A. Deutsch, and J. Zimmermann. The REACH Active OODBMS. In *Proc. ACM SIGMOD Int. Conf. on the Management of Data*, 1995.
6. S. Chakravarthy and D. Mishra. SNOOP: An Expressive Event Specification Language for Active Databases. *Data and Knowledge Engineering*, 14(1), 1994.
7. M. Eckert. Reactivity on the Web: Event Queries and Composite Event Detection in XChange. Master's thesis, Inst. for Informatics, Univ. Munich, Germany, 2005.
8. R. Meo, G. Psaila, and S. Ceri. Composite Events in Chimera. In *Proc. 5th Int. Conf. on Extending Database Technology*, 1996.
9. B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda. Monitoring XML Data on the Web. In *Proc. ACM SIGMOD Conf. on the Management of Data*, 2001.
10. G. Papamarkos, A. Poulovassilis, and P. Wood. Event-Condition-Action Rules Languages for the Semantic Web. In *Proc. Workshop on Semantic Web and Databases*, 2003.
11. N. W. Paton. *Active Rules in Database Systems*. Springer-Verlag, 1999.
12. J. Widom and S. Ceri. *Active Database Systems*. Morgan Kaufmann, 1996.
13. Xcerpt. http://xcerpt.org.