Evolution of Distributed Web Data: An Application of the Reactive Language XChange

François Bry Michael Eckert Hendrik Grallert Paula-Lavinia Pătrânjan
University of Munich, Institute for Informatics
http://www.pms.ifi.lmu.de, {bry,eckert,grallert,patranjan}@pms.ifi.lmu.de

Abstract

Many data sources on the Web evolve in the sense that they change their content over time, typically as a reaction to some event. Such changes often need to be mirrored in data on other Web nodes: updates need to be propagated. To respond to the need for evolution and reactivity both locally and globally, the language XChange has been developed. In this work, we demonstrate its applicability to a concrete scenario of distributed Web sites of a scientific community with mutual data dependencies.

1. Introduction

Many data sources on the Web and Semantic Web are evolving in the sense that they change their content over time in reaction to events bringing new information or rendering existing information out-of-date [2]. Often, such changes must be mirrored in data on other Web nodes: updates need to be propagated.

The reactive, rule-based language XChange [4] has been developed to respond to the need for both local (at a single Web node) and global (distributed over several Web nodes) evolution and reactivity on the Web. Borrowing ideas from active database systems [7], XChange is a language of Event-Condition-Action (ECA) rules. XChange is tailored for the distributed nature of the Web and for common Web data formats by allowing event-based communication and by embedding the versatile Web query language Xcerpt [6, 8].

The demonstration described in this article shows how XChange can be applied to programming reactive Web sites where data evolves locally and, through mutual dependencies, globally. The setting we consider are several distributed Web sites of a fictitious scientific community of historians called the Eighteenth Century Studies Society (ECSS). ECSS is subdivided into participating universities, thematic working groups, and project management. Universities, working groups, and project management have each their own Web site, which is maintained and administered locally. The different Web sites are autonomous, but cooperate to evolve together and mirror relevant changes from other Web sites. For example, Web sites maintain information about personal data of members; a change of member data at a university entails further changes at the Web sites of the management and some working groups.

2. The Language XChange

ECSS's distributed Web sites are realized using the language XChange. XChange provides the following benefits over conventional approaches based on general-purpose programming languages to implement reactive behavior as needed in the demo:

(i) XChange reactive rules are highly declarative. They allow programming on a high abstraction level and are easy to analyze for both humans and machines.

(ii) 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.

(iii) 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.

(iv) XChange embeds an XML query language, Xcerpt, allowing to access and reason with Web data.

(v) XChange provides an integrated XML update language for modifying Web data.

(vi) 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*.

(vii) XChange's high abstraction level and its powerful constructs allow for short and compact code.

We give in this section a very short introduction to the language XChange emphasizing general ideas especially those related to distributed programming with ECA rules. The demo itself will be described in the next section.

```
ON    xchange:event {{
        change-member {{
          memberId { var ID },
            newData {{
              workingGroups {{
                without wg {"Church and Reformation"}
      }} }} }} }}
FROM in { resource {"http://ecss.org/members.xml"},
        members {{
          member {{
            id { var ID },
              desc wg {"Church and Reformation"}
      }} }} }
DO    xchange:event {
        xchange:recipient { "http://churchreform.net" },
          delete-member {
            memberId { var ID }
      } }
END
```

Figure 1. ECA propagating the removal of a member from a working group

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*. Such an ECA rule has the 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. With this, we can see that both event and Web queries serve a double purpose of detecting *when* to react and –through binding variables– *how* to react. For querying data, as well as for updating data, XChange embeds and extends the Web query language Xcerpt.

XChange programs at different Web sites can coordinate each other by sending and receiving events. Events are communicated in a push-manner as XML messages. Push communication has several advantages over pull communication: it allows faster reaction, avoids unnecessary network traffic through periodic polling, and saves local resources.

Example The rule in Figure 1 runs at the management's Web site and reacts to changes in the working group affiliation of a member. The event query detects `change-member` events, where the member is not part of the working group "Church and Reformation." The Web query then tests if this member has previously been a member of this working group. If this is the case, an event message is sent to the working group's Web site, requesting the member to be deleted.

We now take a closer look at the individual parts of XChange rules and explain the example in more detail, starting with how XML data of both events and Web resources is queried.

2.1. Queries as Patterns

Both event queries and Web queries are based on describing *patterns* for XML data. For conciseness, query patterns as well as construction patterns and update patterns are represented in a term-like syntax. 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 brackets $[\,]$ or braces $\{\,\}$) does not match with terms that contain additional subterms without corresponding subterms in the query t.

Query terms contain variables for selecting subterms of data terms that are bound to the variables. Accordingly, the result of a query are bindings for the free variables in that query.

More advanced constructs for describing query patterns are available, e.g., in our example we use `without` to query the *absence* of subterms and `desc` to query for subterms that are not immediate children of the parent term but *descendants* at arbitrary depth. Non-structural conditions (e.g., comparisons on integer variables) can be specified by adding a `where`-clause to queries.

2.2. Event Queries

Each Web site monitors the incoming event messages (XML representations of events) to check if they match an event query of one of its XChange rules. Atomic event queries are single query terms (as described above) and detect and react to single incoming event messages. In the example rule, we have an atomic event query which binds the variable ID to the content of the `memberId`-element.

Often, situations that require a reaction by a rule are not given by a single atomic event, but a temporal combination of events. For this, XChange supports composite event queries [3], which are built by combining event queries with composition operators like `andthen` (ordered sequence of events), `and` (unordered conjunction of events), or `without` (absence of events) and temporal restrictions like `within` (all events happen within a given length of time).

2.3. Web Queries

The condition part of XChange rules queries data from Web resources such as XML documents or RDF documents.

The keyword `in` together with `resource` is used to specify the URIs of the documents that are queried.

Queries can be combined into conjunctions (`and`) and disjunctions (`or`), thus allowing to access multiple documents and complex conditions. Also, negation (as failure) is supported (`not`).

2.4. Actions: Updates and Raising New Events

The (re)action part of XChange rules has 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.

Raising New Events Events to be raised are specified as a construction pattern for the new event messages. Construction patterns (also called construct terms) are similar to query patterns; however only complete patterns (single brackets/braces) can be used. Variables are replaced by the bindings obtained previously in the event and Web queries.

Grouping and aggregation is supported through constructs like `all` ct `group-by var` X, which will be "replaced" by one construction of ct for each binding of the variable X.

Construction patterns for events must contain an element `xchange:recipient` which specifies the recipient Web node's URI. Note that this can be a variable.

Updates Updates to Web data are specified as so-called update terms. An update term is a (possibly incomplete) query 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 a construct term that is to be inserted, a deletion operation specifies a query term for deleting all data terms matching it, and a replace operation specifies a query term to determine data items to be modified and a construct term for their new value.

Complex Actions 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.

3. Demo Description

Our demonstration applies the ECA rule language XChange to the distributed Web data of ECSS, as described in Section 1. Data evolves locally and updates are propagated globally by means of event messages. The Web sites of universities, working groups, and management are autonomous, but cooperate to evolve together and mirror relevant changes from other Web sites.

The different Web sites maintain XML data about members, publications, meetings, library books, and newsletters. Data is often shared, for example a member's personal data is present at his home university, at the management node, and in the working groups he participates in. Such shared data needs to be kept consistent among different nodes; this is realized by communicating changes as events between the different nodes using XChange ECA rules.

Events that occur in this community include changes in the personal data of members, keeping track of the inventory of the community-owned library, or simply announcing information from email newsletters to interested working groups. These events require reactions such as updates, deletion, alteration, or propagation of data, which are implemented using XChange rules. The rules run locally at the different Web nodes of the community, allowing for the processing of local and remote events.

For a concrete example, consider changing a member's personal data including his working group affiliation. The information flow is depicted in Figure 2. The initial change is entered by using a Web form at the member's home university LMU. The form generates event message $m1$. One ECA rule ($r1$) reacts to this event and locally updates the member's data at LMU accordingly. Another ECA rule ($r2$) forwards the change to the management node.

The management node has rules for updating its own local data about the member ($r3$) and for propagating the change to the affected working groups ($r4$ for adding, $r5$ for deleting a member). In the example, the member changes the working group affiliation from WG2 to WG3. Accordingly, event $m4$ is sent to WG3 by rule $r4$ and $m3$ is sent to WG2 by $r5$.

The working groups finally each have rules reacting to deletion and insertion events ($m2$ and $m3$) to perform the requested updates (here: $r6$ at WG2 and $r7$ at WG3).

In this description we have restricted ourselves for space reasons to this one example of changing member data. The demonstration realizes full member management of the community, a community-owned and distributed virtual library (e.g., lending books, monitions, reservations), meeting organization (e.g., scheduling panel moderators), and newsletter distribution. These other tasks are also implemented by ECA rules that are in place at the different nodes. For presentation purposes, the demonstration includes facilities for displaying the rules of each node and logging received and sent events.

$r1$: ON change member
DO update LMU data
$r2$: ON change member
DO forward to management
$r3$: ON change member
DO update management data
$r4$: ON change member (w/WG3)
IF was not member of WG3
DO send add member to WG3
$r5$: ON change member (w/o WG2)
IF was member of WG2
DO send remove member to WG2
$r6$: ON remove member
DO update WG2 data
$r7$: ON add member
DO update WG3 data

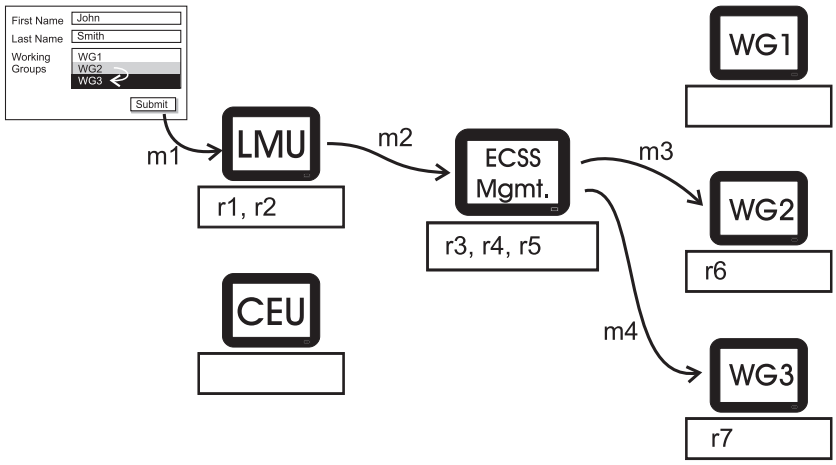

Figure 2. Changing a member's personal data (including working group affiliation)

4. Conclusions

While a similar behavior as the one in the demo could be obtained with conventional programming languages, XChange provides an elegant and easy solution. Evolution of data and reactivity on the Web are easily arranged for by using readable and intuitive ECA rules. Moreover, by employing and extending Xcerpt as a query language, XChange integrates reactivity to events, querying of Web resources, and updating those resources in a single, easy-to-learn language.

The demo presented here has been implemented in the framework of a three months independent study project. The student has had no prior experience with XChange, Xcerpt, and rule-based programming (including ECA rules). Out of the three month, large parts were dedicated to designing the use case from scratch; the actual rule authoring consumed less than one month. Judging from the learning curve, we estimate that adding a new task to the demo (such as managing reports that are delivered to the funding agency of the ECSS) could be done within only two or three days now.

XChange is an ongoing research project [9]. The design, the core language constructs, and the semantics are completed. A prototype implementation is available and used to run the demo described in this article.

We are currently considering automatic generation of ECA rules (e.g., from data dependency specifications or for monitoring integrity constraints), constructs for structuring of rule sets, and efficient evaluation of rule sets. Learning from experience with the current "algebraic" approach to querying composite events (which is similar to the approaches found in active databases [10]), a novel composite event query language is being developed [1]. Issues related to efficient event query evaluation are investigated in this work. Also, other use cases than the one presented here are investigated, in particular in the domain of business processes and business rules [5].

References

[1] F. Bry and M. Eckert. A high-level query language for events. In *Proc. Int. Workshop on Event-driven Architecture, Processing and Systems*. IEEE, 2006. To appear.

[2] F. Bry and M. Eckert. Twelve theses on reactive rules for the Web. In *Workshop "Reactivity on the Web" at Int. Conf. Extending Database Technology*, 2006. (Invited paper).

[3] F. Bry, M. Eckert, and P.-L. 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.

[4] F. Bry, M. Eckert, and P.-L. Pătrânjan. Reactivity on the Web: Paradigms and applications of the language XChange. *Journal of Web Engineering*, 5(1):3–24, 2006.

[5] F. Bry, M. Eckert, P.-L. Pătrânjan, and I. Romanenko. Realizing business processes with ECA rules: Benefits, challenges, limits. In *Proc. Int. Workshop on Principles and Practice of Semantic Web*, LNCS. Springer, 2006.

[6] S. Schaffert and F. Bry. Querying the Web reconsidered: A practical introduction to Xcerpt. In *Proc. of Extreme Markup Languages Conf.*, 2004.

[7] J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, CA, USA, 1996.

[8] Xcerpt. http://xcerpt.org.

[9] XChange. http://www.reactiveweb.org/xchange.

[10] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *Proc. Int. Conf. on Data Engineering*. IEEE, 1999.