INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

Project thesis

Propagation of updates in distributed Web data: A use case for the language XChange

Hendrik Grallert

Aufgabensteller:	Prof. Dr. François Bry
Betreuer:	Prof. Dr. François Bry
	Paula-Lavinia Pătrânjan
	Michael Eckert
Beginn:	01.11.2005
Abgabe:	09.08.2006

Contents

1 Introduction

The World Wide Web (or simply the Web) is often perceived as a hypertext-based, distributed information system, which presents the user with documents, called Web pages. The network paradigm usually associated with the Web is that of a client/server architecture, with servers hosting collections of Web pages, and Web browsers as user clients.

This view is not necessarily complete. Contemporary Web applications include e-commerce, business to business, and logistics systems as well as larger information portals or e-learning systems that involve new concepts exceeding simple document retrieval. One of these concepts is reactivity, the ability to detect events and respond in a timely manner. Events in this context are happenings on the Web, such as changes in persistent data repositories (Web sites). Events are communicated as messages propagating data about the event.

The language XChange has been developed to allow for reactivity on the Web. XChange programs detect events specified by event queries, and react by updating data repositories and further propagation of events. This report describes a use case for XChange.

1.1 Motivation and goals

As of now, the language XChange has been designed in [5]. A prototype implementation is also available. Additionally, this and related work has brought up a series of proposals for use cases:

- Travel organization is the example used throughout the work which originally described the language XChange (See [5]). The underlying idea is that of a personal travel organizer, a Web based application that communicates with the computer systems of hotels, air lines, or travel agencies to plan a journey and adapt it according to changes such as belated flights or sudden alterations in the intended route.

- The distributed system of a car rental company has been proposed as a use case in [6].

- [1] contains the description of a use case for an information system of a scientific research community, the "Rewerse Information System and Portal".

As neither of these use cases has been implemented yet for use with the prototype, the development of a distributed XChange application still remained to be done. This was the objective of the work described in this report. The goals of the report are:

- to develop a use case for the language XChange,

- to describe the implementation of that use case,

- to describe problems of the prototype discovered during implementation,
- to suggest additional features or constructs for future versions of the language XChange.

The scenario chosen for the use case is a distributed information system of a fictitious scientific research community, the Eighteenth Century Studies Society (ECSS). The implemented application is intended for demonstration purposes of XChange.

1.2 Organization of this report

This introduction continues with a short overview over paradigms, features, and concepts of the language XChange (section 1.3) as well as the scenario which serves as background for the use case (section 1.4). Section 2 explains the implementation in XChange and discusses the employed language constructs in an informal way[1] and by means of examples from the use case. Auxiliary tools that have been developed for demonstration purposes are presented in section 3. The conclusion contains a discussion of discovered problems of the prototype and desirable language constructs. A description of how to set up the use case in a network and the complete source code of the implementation can be found in the appendix.

1.3 Framework for the use case: The language XChange

XChange aims at realizing reactivity on the Web. It is a high-level language that permits Web sites to detect events occurring on the Web and to respond to them in a timely manner. XChange programs consist of Event-Condition-Action (ECA) rules, which allow an clear specification of event queries, Web queries, and subsequent actions. Among the concepts that imbued the development of the language are the idea of a clear separation of persistent and volatile data, applicability to the distributed, non-hierarchical character of the Web, and the need for an embedded query language. New features of XChange also include the possibility to detect composite events, and to perform complex updates in response to events.

Reactivity on the Web

Reactivity on the Web is increasingly needed by different kinds of Web-based systems, for example information portals or distributed business applications. XChange as a reactive language therefore combines the ability to send and receive messages about events, and to react on single events or on a series of events by updating data – e.g. updating the inventory of a library in reaction to lending processes.

[1] It is not intended to give a comprehensive and detailed description of the language here. For such information, see [5].

Advantages of XChange as a higher-level language

While reactive behavior could be achieved with lower-level technologies or languages not specially developed with the requirements of reactivity in mind, such as HTTP, SMTP, or SOAP, using a higher-level language like XChange allows for outright concentration on the principal task and a most elegant way in getting it done. A higher level of abstraction, an intelligible syntax even for complex updates, and the use of ECA rules which closely resemble real-life sequences of actions and reactions make XChange a convenient and cost-efficient language for distributed, reactive Web applications.

The notion of events

An event in this context is considered as some happening on the Web. The data representation of such events, i.e. the messages informing about their taking place are also called events. XChange events are XML documents. They comprise of an envelope, for example containing information about the sender or receiver, and some application specific data as the body of the message.

ECA rules

XChange programs are located on one specific Web site and consist mainly of Event-Condition-Action (ECA) rules. These rules are made up of three parts: event part, condition part, and action part. Every event incoming at the Web site is queried by an event query given in the first part of the ECA rule. This event query states a pattern that the XML event message in question must match. Besides detecting events of interest, the event query also selects data from the event for later use in the other parts of the rule. Next, the condition part is evaluated. Here data resources on the Web can be queried, for example to check if they need to be updated. If this query also is answered in the affirmative, the rule fires, that is, the action is performed as specified in the third part of the rule.

Using ECA rules for a reactive language like XChange is advantageous for several reasons. Often, processes modeled in programming resemble the ECA schema in a natural way. For example the rule: “When the library in Munich requests a rare book as an inter-library loan from the library in Cologne, and if that book is available there, sent it to Munich” can be easily implemented in XChange. Furthermore, messages informing about events are a well known paradigm in network communication, and data from events can be employed in the condition and action part of the rule.

In addition to ECA rules, there are deductive rules in XChange. These can be employed to construct views over persistent data.

Separation of persistent and volatile data

The underlying idea of a clear separation of persistent and volatile data in XChange becomes obvious if one considers the different character of the data queried in the condition part of ECA rules and that queried in the event part. Persistent data are Web resources that are constantly available – probably under a certain address – and that can be altered. Put in a more abstract way, one could describe them as denoting a state of the world. Volatile data on the other hand is event data transmitted over the Web and cannot be manipulated once sent. It could be described as informing about a change of the state of the world.

Distributed, non-hierarchical infrastructure

XChange servers on the Web generally have equal status and control in communicating events, following the peer-to-peer communication model. Events are announced to other nodes using a push strategy, that is, servers send events to possibly interested Web sites. There is no periodical polling of other servers for events (which would be a "pull" strategy). Incoming events are processed at each separate server and evaluated with respect to the locally installed set of rules.

Xcerpt embedded

XChange embeds the Web query language Xcerpt. This is useful both for querying and reasoning with persistent data from Web resources and for querying incoming events. Xcerpt follows a pattern-based approach in querying and constructing XML data. Xcerpt thus is employed in formulating all parts of XChange rules, resulting in a high uniformity and ease of use.

Composite events

Composite event queries allow the detection of a temporal correlation between incoming events. If for example a discount is offered for a customer of an online shopping system, but this discount is restricted to the duration of a month after he created his customer account, XChange can determine if an order is still within this period with the help of a composite event. In this case, it would be checked if both events – registration and order – occurred within a given time period.

Complex updates

Besides simple insertions, deletions, and alterations of data, XChange supports complex updates. These are – possibly ordered – conjunctions or disjunctions of simple updates. A basic example for a complex update is an update of local data combined with recording this update in a log. But also more intricate interrelations, such as the combined booking of a flight and hotel stay, which depend on each other, can be realized with complex updates.

1.4 A portal for a distributed research community: The Eighteenth Century Studies Society (ECSS)

The background for the use case presented in this report is a fictitious scientific research community, the Eighteenth Century Studies Society (ECSS). It is supposed to be an international network of scholars with a special research interest in the European 18th century. The members of the ECSS are situated at the universities of Budapest/Hungary, Edinburgh/UK, and Cologne and Munich, Germany. Research activities focus on pan-European developments in the 18th century, i.e. the enlightenment, ecclesiastical reform, or anti-absolutist and democratic movements. About 100 members cooperate in the seven working groups of the society.

A first geographical layer of organization is provided by the following participating institutions ("participants"):

- The Department of History, Central European University, Budapest
- The College of Humanities, Edinburgh University
- Philosophische Fakultät, Universität zu Köln (Cologne)
- Historisches Seminar, Ludwig-Maximilians-Universität, Munich

Moreover, the ECSS is structured factually by its working groups. There are groups on the following topics:

- The Enlightenment
- Nation and Government
- Church and Reformation
- Revolution and Democratization
- Media and Communication
- Economical Issues
- Theory and Method

Members of the different participating institutions gather in each working group. A management office has been set up to take care of issues concerning the society as a whole.

A special service of the society for its members is the acquisition, storage, and provision of old, expensive or otherwise rare books for research purposes. Therefore, libraries have been established at all participating universities. Members can lend these books at their local library, or – if the desired book is not available locally – commission inter-library loans from other locations. The management office is charged with the task of keeping track of the books.

Communication within the society and to the public is maintained by a distributed Web based information system. The management office, the participants, and the working groups each have their own Web node (server) in this information system.

The nodes of the participants contain Web sites displaying information about those employees of the universities which are members of the ECSS, about their working group allocations, and publications. As membership in the society depends on being an employee of one of the participants, any changes in member data are initiated at the participants nodes. For this purpose, they offer Web forms for adding, changing, and deleting employee records. The Web sites of the participants also host special pages informing about the inventory of the ECSS library, and the local availability of books. The lending process is initiated at the nodes of the participants, too.

The management node provides Web sites informing about all aspects of the society. A complete list of members and their publications can be found on the Web site of this node. Moreover, there is a number of Web forms available supporting the organization of meetings of the society. Members can register for such meetings here, or propose papers and panel moderators. The management node also keeps a list of the books owned by the society and their current locations. Finally, it has subscribed certain newsletters to be able to inform the working groups about interesting developments in their areas of research.

The working group nodes also provide member and publication lists of their own. Additionally, they gather information and news of interest for the working group on their home pages.

As becomes obvious from the description of the structure of the information system, there are a number of dependencies of data stored on the different nodes:

- Member data, which is entered at the participant nodes, is also used by the management office and the working group nodes.
- Inter-library loans of books imply a coordination between the libraries situated at the participants and the management office, which keeps track of the current locations of books.
- News gathered from newsletters by the management office must be distributed to the working groups, corresponding to their interests.

These dependencies necessitate the propagation of data between the different Web nodes of the system, and updates of the different data repositories. Because of these needs arising from the distributed structure of the society's system, it is very well suited for implementation with XChange.

2 Implementation in XChange

The previous section has described the structure of the ECSS information system and portal and also has given an overview over the necessary propagation of data and updates. This section turns to the implementation of the rules that perform updates on these events or propagate changes by raising new events. It has a double purpose in that it documents the functionality of the use case on the one hand and explains the employed language constructs in an informal way[2] on the other hand.

2.1 Member management

Member data is used by all nodes and on all Web sites of the ECSS. The task of member management is to keep track of changes in the particulars of members, to inform the system about new members, and to remove obsolete member data. The members of the ECSS are employees of the participating universities. Thus the insertion, deletion, and alteration of member data is initiated at the Web sites of the participants. Once the data is updated locally on these sites, changes are then propagated to the management node, and from there to the working group nodes.

2.1.1 Adding members: insertion of data

The most basic update that is accomplished with XChange in this use case is the insertion of a new member on the node of a participant. This requires only a local update. The local update will of course be propagated later on to other servers, but this is treated in another section, 2.1.4 Information of other nodes in the system: propagation of updates. This section explains all steps of the local update, from entering the data via a Web form over detecting the resulting event with the event query of an appropriate XChange rule to performing the update in the action part of that rule.

The data collected about each member in this use case consists of the first name and surname, title, e-mail address, phone number, office room number, and ECSS working group allocation. This data is supplied by users via a Web form at the site of the participant employing the new member:

[2]For more comprehensive information about the language XChange, grammar rules for language constructs etc., see [5].

Submitting the form generates an XML event which is sent to the local XChange server. (The event is generated by a PHP script. See section 3.3 Subsidiary PHP functions.) The XML event corresponding to the new record of John Smith (see previous figure) looks as follows:

```
<xchange:event>
  <xchange:sender>
    http://history.lmu.de/procEmployee.php</xchange:sender>
  <xchange:recipient>
    http://history.lmu.de/</xchange:recipient>
  <xchange:raising-time>
    2006-03-15T12:45:00</xchange:raising-time>
  <type>insert employee</type>
  <employee>
    <surname>Smith</surname>
    <firstName>John</firstName>
    <title>Dr.</title>
    <email>john.smith@lmu.de</email>
    <phone>089 / 2180 - 001</phone>
    <office>010</office>
    <ecssWorkingGroups>
      <ecssWorkingGroup>
        The Enlightenment
      </ecssWorkingGroup>
```

```
      <ecssWorkingGroup>
        Church and Reformation
      </ecssWorkingGroup>
    </ecssWorkingGroups>
  </employee>
</xchange:event>
```

For a discussion of the structure of XChange events, see section 2.1.4 Information of other nodes in the system: propagation of updates. For now it suffices to note that all the data entered at the Web form is communicated in the XML event to the local XChange server.

The local server of the LMU has an appropriate Event-Condition-Action (ECA) rule in place, which reacts on this event, and performs the update on local data:

```
TRANSACTION
  in { resource { "file:staff.xml" },
    staff {{
      insert var EMPLOYEE }}  }
ON
  xchange:event {{
    type { "insert employee" },
    var EMPLOYEE -> employee {{}}  }}
END
```

Before the details of this specific rule are discussed below, recall the definition of an ECA rule as already presented in section 1.3, ECA rules. ECA rules specify a certain event, on whose occurrence a condition is verified. If the condition is met, the action is performed. XChange programs consist mainly of ECA rules.[3] TRANSACTION denotes the beginning of the action part, or update specification, of the ECA rule, ON opens the event part, or event query, and END closes the rule. The above rule does not include any conditions. If there were conditions, that is, a Web query, in the rule, the condition part would be introduced by the keyword FROM. A complete updating rule in XChange thus looks as follows:

```
TRANSACTION
  <update_specification>
ON
  <event_query>
FROM
  <web_query>
```

[3]There may also be deductive rules to construct views over persistent data, but they are not important for the implementation of this use case.

END

The event query serves to identify those XML events on which the rule will fire. It consists of a pattern of such events given in Xcerpt notation.[4] Whenever the pattern matches an incoming event in structure and contents, the remainder of the rule – condition and action part – are evaluated.

To give a short overview of the notation, the pattern for an XML element in Xcerpt is given by the name of the element followed by a pair of brackets. Element contents are put into quotation marks. The pattern for an element like

```
<title>Dr.</title>
```

therefore is

```
title { "Dr." }
```

Comparing the XML event and the event query of the example, it stands out that there are many more elements in the event than in the pattern. Event and pattern match nonetheless, because an *incomplete* pattern is employed in the query. That means only a rough outline of the expected event is specified, whereas the real event may contain more XML elements. Incompleteness in Xcerpt is indicated by double brackets. In the event query of the above example there are double brackets just after the root element xchange:event, which means that in the incoming event there may be other child elements than type and employee. This is necessary, as indeed there are some more elements in the example, such as sender or recipient. If for some reason a complete pattern needs to be specified, single brackets have to be used.

There is also a difference between curly ({ }) and square ([ ]) brackets in Xcerpt. Elements enclosed in curly brackets are considered to be unordered, while those in square brackets are considered to be ordered. In the above example, as curly brackets are employed, it would therefore not matter if in the incoming event the element employee was given prior to the element type. In case square brackets were used, type had to precede employee for the event to match the pattern. Of course square brackets can also be single or double, so that complete or incomplete ordered patterns are possible. An example for ordered patterns can be found in section 2.2.2 Monitions: composite events.

Besides identifying events, the second purpose of event queries is to bind content from the event to variables for further use in the condition or action parts. If the contents of an element are to be bound to a variable, this variable preceded by the keyword var has to be specified in place of the content. The title of a person as given in the above example could for instance be extracted from the element title as follows:

[4] Xcerpt is the query language XChange is built upon. (See section 1.3, Xcerpt embedded.) To be precise, the syntax used in XChange to specify patterns is not exactly the syntax of Xcerpt, but a very similar one. Certain constructs have been added to that syntax in XChange. For details, see [5].

title { var TITLE }

Note that the upper case of variable names is only for readability in this report.

Sometimes it is necessary to store whole element structures in a variable. For this purpose it is possible to bind an element "from the outside" to a variable by employing the "as" construct (an arrow consisting of a hyphen and a "greater than" sign: −>). In the above example the construct:

var EMPLOYEE −> employee {{}}

binds the employee element with all child elements – thus the double curly braces denoting incompleteness – to the variable EMPLOYEE.

Using a similar XML structure in events and the data to be modified considerably reduces the patterns in rules. For example, in this case, the whole employee element structure can be taken from the event and used in the update specification. As there may be other events incoming that include an employee element, type is necessary in this pattern to restrict the reactions of this rule to appropriate events.

The other part of the rule to be considered is the update specification. It starts out with the keyword in followed by a resource specification and an update pattern enclosed in curly braces. The resource specification gives a URI for the XML document that is to be updated, in our case staff.xml. It is separated by a comma from the update pattern.

The meaning of the update pattern of the above rule is to look for the root element, staff, which may contain several arbitrary child elements. That is, only an incomplete pattern is given, which is indicated by the double curly braces. The insert construct then effects that the contents of the variable EMPLOYEE are fit in as a child element of staff.

Technically spoken, the keyword insert is followed by a construct term. Construct terms are complete patterns including variables in order to construct new data.[5] The construction of the new data term is quite simple in this case, as it is taken unchanged from the variable bound in the event query. It is of course possible to construct wholly new data terms in this place. Consider the case that only first name and surname of the employee need to be inserted into staff.xml, and that they were bound to the variables FIRSTNAME and SURNAME in the event query. The update term then looks as follows:

```
insert employee {
         surname { var SURNAME },
         firstName { var FIRSTNAME }  }
```

In this code, the elements are constructed explicitly, while only the contents are taken from variables.

[5]For in dept information on construct terms, see [5], p.35, and on their use in insertion specification, p. 96.

In any case the constructed term is inserted into the existing persistent data, which here is an XML file named staff.xml. The following shows an excerpt of (possible) contents of this file after the update (using the original update specification):

```
<staff>
  <employee>
    <surname>Smith</surname>
    <firstName>John</firstName>
    <title>Dr</title>
    <email>john.smith@lmu.de</email>
    <phone>089 / 2180 - 001</phone>
    <office>010</office>
    <ecssWorkingGroups>
      <ecssWorkingGroup>
        The Enlightenment
      </ecssWorkingGroup>
      <ecssWorkingGroup>
        Church and Reformation
      </ecssWorkingGroup>
    </ecssWorkingGroups>
  </employee>
  <employee>
     ...
  </employee>
   ...
</staff>
```

With this insertion all steps of the local update are complete. The data of a new member, which has been provided via a Web form, has been sent to the local XChange server. An ECA rule registered with the server detected the incoming event because its structure and contents matched the incomplete pattern specified in the event query of the rule. Using data selected and bound to a variable in the event query, the update part of the rule has constructed a new data item for the insertion into the persistent member data stored on the server in the file staff.xml.

2.1.2 Changing member information: modification of existing data

Once a member has been added to the data of a participant's node, the data can be changed, for example because of some alteration in the member's particulars. This is done using a Web form very similar to the one for inserting a new member. The Web form can be accessed from the page displaying the details of a member:

The whole process of initiating a change via the Web form and thereby sending an event triggering a local update is very similar to the insertion of a new member presented in the previous section. In this case the XML event passed by the form contains the new data (in an employee element) as well as the surname and first name of the member previous to the change. The latter act as a key to identify the member record which is to be changed[6] and is read in from staff.xml by the same script that processes the form and sends the event[7]. The event thus looks as follows:

```
<xchange:event>
  <xchange:sender>
    http://history.lmu.de/procEmployee.php
  </xchange:sender>
  <xchange:recipient>
    http://history.lmu.de/
  </xchange:recipient>
  <xchange:raising-time>
    2006-03-15T12:45:00
  </xchange:raising-time >
  <type>change employee</type>
  <surname>Smith</surname>
  <firstName>John</firstName>
  <employee>
```

[6]To avoid problems with several members that happen to have the same name, one could of course use some other key, such as a unique ID number.

[7]See section 3.3 Subsidiary PHP functions for details.

```
    <surname>Smith</surname>
    <firstName>John</firstName>
    <title>Dr</title>
    <email>john.smith@lmu.de</email>
    <phone>089 / 2180 - 001</phone>
    <office>020</office>
    <ecssWorkingGroups>
      <ecssWorkingGroup>
        Church and Reformation
      </ecssWorkingGroup>
      <ecssWorkingGroup>
        Revolution and Democratization
      </ecssWorkingGroup>
    </ecssWorkingGroups>
  </employee>
</xchange:event>
```

The ECA rule for changing an employee record, which fires upon receipt of the above event, is quite concise:

```
TRANSACTION
  in { resource {"file:staff.xml"},
    staff {{
      employee {{
        surname { var SURNAME },
        firstName { var FIRSTNAME }  }}
      replaceby var EMPLOYEE }}  }
ON
  xchange:event {{
      type { "change employee" },
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      var EMPLOYEE -> employee {{}}  }}
END
```

In the event query of this rule, the variables SURNAME and FIRSTNAME are bound directly to the content inside of the corresponding XML elements, while EMPLOYEE is bound from the outside to the employee element. The surname and first name information gathered from the event are employed in the update pattern to identify the employee record that is to be changed. The actual substitution is specified by the construct

```
employee {{ ... }} replaceby var EMPLOYEE
```

which replaces the existing employee element with the new data from the variable EMPLOYEE.

Generally, replaceby operates with two terms, a query term, given on the left side of replaceby, and a construction term on the right side. The query term constitutes a pattern for those data terms that have to be replaced. In the example, the query term is:

```
employee {{
  surname { var SURNAME },
  firstName { var FIRSTNAME }  }}
```

All employee elements with the corresponding first name and surname thus match for replacement. Note that the whole replaceby construct in the update pattern is used within the staff element (which is the root element of this XML data). This further restricts replacements to employee elements inside staff.

The construct term on the right side of the replaceby construct in this example only consists of the variable EMPLOYEE. As explained in section 2.1.1 Adding members: insertion of data, construct terms are patterns incorporating variables in order to construct new data terms. Thus, if needed in some instance, they can of course be more elaborate than in this example.[8]

The important difference in the rule for changing member data, compared with the one adding a new member, therefore is the replaceby construct. As the insert construct it takes a construct term to specify the structure of the new data. Additionally it requires a query term, which identifies the parts of the persistent data that have to be replaced. For this purpose, the event contains some "key" information additional to the new member data.

2.1.3 Removing members: deletion of data

Member records can of course also be deleted. The page showing the member details on a participant's node offers the possibility to delete a member:

[8] For a more comprehensive discussion of construct terms, see [5], p. 35.

A member is selected for deletion, if the [Delete] link on the member details page is clicked. Consequently, an event as follows is generated and sent to the local XChange server:

```
<xchange:event>
  <xchange:sender>
    http://history.lmu.de/procEmployee.php
  </xchange:sender>
  <xchange:recipient>
    http://history.lmu.de/
  </xchange:recipient>
  <xchange:raising-time>
    2006-03-15T12:45:00
  </xchange:raising-time >
  <type>delete employee</type>
  <surname>Smith</surname>
  <firstName>John</firstName>
</xchange:event>
```

In the body of the event, as in the case for changing particulars of members discussed in the previous section, again the first name and surname of the employee are given as a key to its record.

An ECA rule employing the delete construct of XChange removes the member from staff.xml:

```
TRANSACTION
  in { resource {"file:staff.xml"},
    staff {{
      delete employee {{
               surname { var SURNAME },
               firstName { var FIRSTNAME }  }}  }}  }

ON
  event {{
    type { "delete employee" },
    surname { var SURNAME },
    firstName { var FIRSTNAME }  }}
END
```

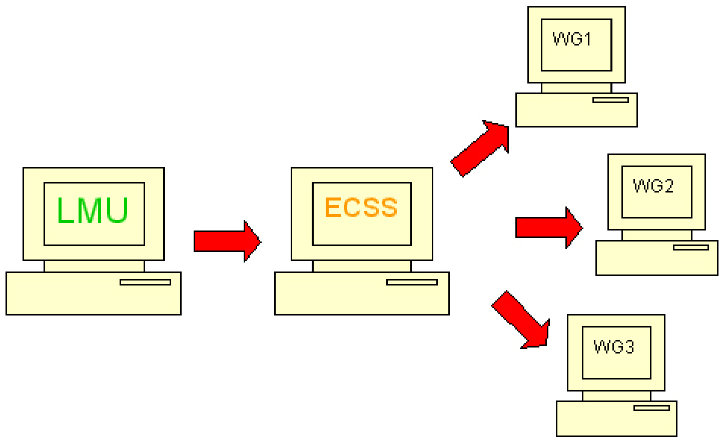
In the update pattern again the employee element is identified by the surname and first name given in the event. The pattern in turn is incomplete, as the remaining child elements of employee are not considered. The prefixed delete construct removes the elements thus selected from staff.xml.

In technical terms, delete precedes a query term, a pattern identifying all data terms that have to be deleted. In the example the query term is incomplete and requires an employee element with at least the child elements surname and firstName with matching contents, provided by the variables.

Thus deleting a member in comparison to the insertion and replacement operations considered in the preceding sections only introduces a new language construct, delete. The other steps of the local update, with an XChange event generated after some user interaction on a participant's Web site, and a rule firing on the local XChange server subsequent to the receipt of the event, are the same.

2.1.4 Information of other nodes in the system: propagation of updates

Remember that in the information system of the ECSS, member data is entered at the participant nodes, but used on the management node and working group nodes, too. Thus the data on the latter nodes is dependent on that administrated on the former. (For a description of the sever structure and data dependencies of the information system, see section 1.4 A portal for a distributed research community: The Eighteenth Century Studies Society (ECSS).) As a consequence of local insertion, deletion and alteration of member data at a participant's site, updates therefore must be propagated to the management node as well and from there to the working group nodes. For this purpose, special ECA rules at a participant's node react to the same events that trigger the local updates. They inform the management node of the update by raising a new event addressed to this node. On the management node the update is reproduced with local data – there is a list of all members of the ECSS and their details on the management node –, and additionally the events informing about updates are again passed on, this time to the working group nodes. There, lists of members of the working groups are kept, and modified on receipt of such events.



The process of propagation is depicted above. The LMU node at the left is a participant node. Updates of member data are initiated here. The arrows represent the flow of events. Events are generated by the LMU node after local updates, and sent to the ECSS management node. From there, updates of member data are propagated to the working group nodes (WG1 - WG3).

While rules for *updating* the data on the management node and on the working group nodes are nearly identical to those of the participants' nodes (and therefore are not discussed separately in this report), the following, different rule *propagates* an alteration of member data from a participant's node to the management node:

```
RAISE
  xchange:event {{
    xchange:sender { "http://history.lmu.de/" },
    xchange:recipient { "http://www.ecss.org/" },
    type { "change member" },
    var SURNAME,
    var FIRSTNAME,
    var EMPLOYEE }}
ON
  event {{
    type { "change employee" },
    var SURNAME -> surname {},
    var FIRSTNAME -> firstName {},
    var EMPLOYEE -> employee {{}}  }}
END
```

The action part in this case is opened by the keyword RAISE instead of TRANSACTION, indicating that a new event will be raised. The XML structure of the event to be constructed is again given in Xcerpt syntax. As can be seen in this example, an XChange event generally has a root element labeled event. Obligatory child elements are sender and recipient, which belong to the namespace xchange. The URI of the node raising this event is given as sender. The content of recipient is the URI of the management node's XChange server.

Additional obligatory elements for XChange events are raising−time, reception−time, and id. The raising time is the point in time at which the event is generated, and is automatically inserted by XChange upon raising the event. The time information has the format

YYYY−MM−DDThh:mm:ss

This is the ISO 8601 standard format for dates and times. (See [4] for details.) reception−time and id are inserted by the XChange server running on the site of the recipient – they need not to be considered during event generation. The complete syntax of an XChange event upon reception thus is:

```
<xchange:event xmlns:xchange="http://xcerpt.org/xchange">
  <xchange:sender>sender</xchange:sender>
  <xchange:recipient>recipient</xchange:recipient>
  <xchange:raising-time>raising-time</xchange:raising-time>
  <xchange:reception-time>
    reception-time</xchange:reception-time>
  <xchange:id>id</xchange:id>
  event data
</xchange:event>
```

The event data is application specific. In our example, the elements type, surname, firstName, and employee with its child elements are the event data. The type element indicates what action the recipient is supposed to take. Although this could be determined solely by the structure and contents of the remainder of the event data, stating it explicitly allows for shorter and clearer rules.

To summarize, the propagation of events requires a special ECA rule, that is, an event-raising rule, which begins with the keyword RAISE. In the action part of such rules the event to be raised is constructed by stating an event term in Xcerpt notation. Data in variables from the event query of the rule – and of course from the condition part, if there is one – can be used in these event terms. XChange events have a certain format with some obligatory elements such as sender and recipient.

As has been said above, updates on the management node are performed by a set of rules registered there very similar to those running at the participants' nodes. In addition, rules for raising events like the one presented in this section arrange for further propagation to the working group nodes. But propagation of changes in member data includes one peculiar case: Changes of the working group allocation of an ECSS member at a participant's node will effect deletions and insertions of members at the working group nodes – if a member changes his working group allocation, the complete member record is deleted at the Web site of his old working group and added to that of the new one. To decide on such updates, a special set of rules is installed at the management node.

Consider the case that a member was allocated to the working group "Church and Reformation" and then changes to "Revolution and Democratization". The event sent from the participant's node to the management node looks as follows:

```
<xchange:event>
  <xchange:sender>http://history.lmu.de/</xchange:sender>
  <xchange:recipient>http://www.ecss.org/</xchange:recipient>
  <xchange:raising-time>
    2006-03-15T12:45:00
  </xchange:raising-time>
  <type>change member</type>
  <surname>Smith</surname>
  <firstName>John</firstName>
  <member>
    ...
    <workingGroups>
      <workingGroup>
        Revolution and Democratization
      </workingGroup>
    </workingGroups>
  </member>
</xchange:event>
```

In this event, the surname and firstName elements outside the member element again serve as a key to identify the member record. The child elements of member only contain the new data, that is, those elements that have been changed. If, for example, the title, phone number, office room number etc. of this member remain the same, they need not be inserted in the event. In this case, at least the working group allocation has changed and is set to only one working group, "Revolution and Democratization", by this event.

Upon receipt of the above event, two kinds of rules fire on the management node. One rule updates the local list of members, while other rules propagate the change to the working group nodes. Which working group nodes have to be informed is determined by comparing the event data with the local data on the management node prior to the change. The above event for instance informs the management node that the working group allocation of John Smith has changed to one single working group, which is "Revolution and Democratization". Prior to the update on the management node, the member record there still contains the old working group allocation, which includes the working group "Church and Reformation". In our example the data on the management node prior to the update looks like this:

```
<members>
  <member>
    <surname>Smith</surname>
    <firstName>John</firstName>
    <title>Dr</title>
    <email>john.smith@lmu.de</email>
    <phone>089 / 2180 - 001</phone>
    <office>010</office>
    <workingGroups>
      <workingGroup>Church and Reformation</workingGroup>
    </workingGroups>
  </member>
  ...
</members>
```

Thus "Church and Reformation" has to be informed about the deletion of one of its members, while "Revolution and Democratization" has to be instructed to add John Smith as a new member. In order for this to work, it is necessary to *first* evaluate the rule for propagation and *then* that for the local update. In this use case, this is accomplished by placing the rule for propagation before the rule for the update in the file containing the rules. The current XChange prototype then processes the propagation rule first. Note that this is not a language feature of XChange, and may not work in upcoming versions of the XChange prototype.[9]

The following rule detects that John Smith is no longer a member of "Church and Reformation" and sends an event to the working group node requesting the deletion of this member:

```
RAISE
  xchange:event {{
    xchange:sender { "http://www.ecss.org/" },
    xchange:recipient
              { "http://www.church-reformation.net/" },
    type { "delete member" },
    var SURNAME,
    var FIRSTNAME,
ON
  xchange:event {{
    type { "change member" },
    var SURNAME -> surname {},
```

[9]Evaluating rules in a set order is proposed as a language feature in this report (See section 4.4, Order of evaluation of rules.)

```
    var FIRSTNAME -> firstName {},
    member {{
      workingGroups {{
        without workingGroup
                { "Church and Reformation" }  }}  }}  }}
FROM
  in { resource { "file:members.xml" },
    members {{
      member {{
        var SURNAME,
        var FIRSTNAME,
        workingGroups {{
         workingGroup
           { "Church and Reformation" }  }}  }}  }} }
END
```

There are two further constructs in this rule compared to the ones considered above. First, this rule includes a condition part, or Web query, identified by the keyword FROM. Second, in the event query a subterm is negated using without. Due to this negation this rule only reacts on those incoming events that do not contain a working group allocation to “Church and Reformation”. In such cases it has to be checked if the named person was a member of that working group so far, which is done in the Web query. The queried resource is specified just as in update patterns with the keyword in. Also, the pattern is given in the same way as in update specifications. If this condition is met, a deletion event is sent to the server of “Church an Reformation”.

So additional to the introduction of event-raising rules and the format of XChange events already summarized above, this section explained some new aspects: Conditions, or Web queries, marked by the keyword FROM in XChange, are taken on again in the following section, 2.2 A virtual library. Patterns for example in event queries can be refined by Xcerpt constructs such as without, allowing for subterm negation. Finally it might be necessary at times to consider the order in which the rules of an XChange server fire. For the time being and with the current prototype, this can be determined by the order of the rules in the rules file.

2.2 A virtual library

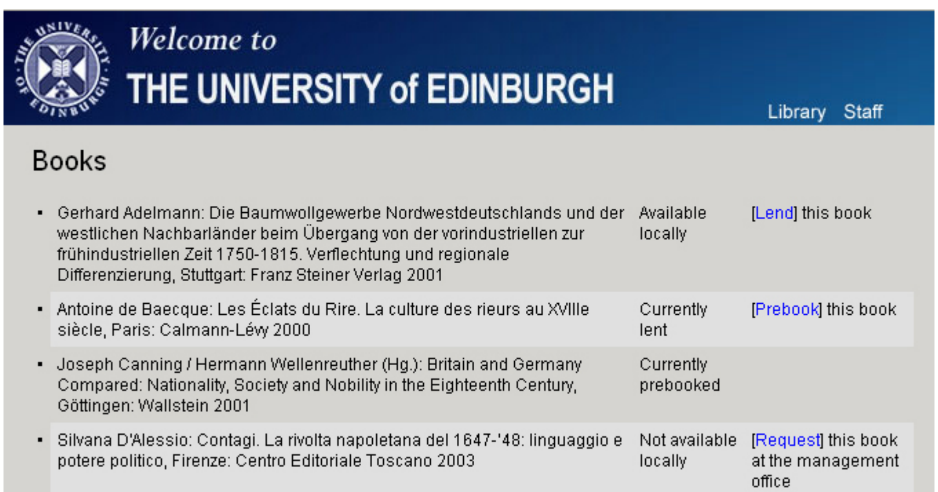
The ECSS owns a small selection of old, expensive, or otherwise rare books which are kept for research purposes of its members. These books are stored at the participants’ libraries in Munich, Cologne, Edinburgh, and Budapest, and sent around whenever a member at a different participating institution needs one of them. Inter-library loans of the local libraries, local lending, and other organizational processes are administered by means of a set of Web sites distributed over the servers of the participants and the management office. While the participants’ nodes provide interfaces for ordering a book from another library or lending it locally, the management node serves with a central list of all books owned by the society. Thus, data concerning inter-library loans is kept at the management node, while data about local availability remains at the participants' nodes. General book details as the author and title for all books are stored both at the participants' nodes as well as at the management node.

2.2.1 Keeping track of books with a central list: using conditions

The Web site of each participant provides a list of all books currently owned by the society, and information about their local availability. Books can be:

- locally available at a participant,
- lent to a member of that participant,
- prebooked by another local member, and
- not available locally

The following screenshot shows the library Web site of the University of Edinburgh, with different availabilities for books, and thus different options for lending, prebooking, or requesting as an inter-library loan:



This information is stored at the participants' nodes in the file books.xml, with the following XML structure:

```
<books>
  <book>
    <callNumber>01</callNumber>
    <title>
```

```
      Gerhard Adelmann: Die Baumwollgewerbe
      Nordwestdeutschlands ...
    </title>
    <state>available</state>
    <emailPrebooked>none</emailPrebooked>
    <emailLent>john.smith@lmu.de</emailLent>
  </book>
  <book>
    ...
  </book>
  ...
</books>
```

Each book is identified by its call number. The state can be one of not available, available, lent, and prebooked, whereas not available means the book is currently situated at another participant, while the others are local states. Additionally the e-mail addresses of the members who have currently lent or prebooked a title are stored. Lending, prebooking, returning of books at a participant's Web site, and even requesting a book which is not available locally triggers very similar corresponding events as follows:

```
<xchange:event>
  <xchange:recipient>
    http://history.ed.ac.uk/
  </xchange:recipient>
  <type>lend book</type>
  <callNumber>01</callNumber>
  <email>john.smith@lmu.de</email>
</xchange:event>
```

The structure of these events is the same in all four cases with the exception of the type of the event, which is one of request book at management node, lend book, prebook book, and return book. If the book is not away at another participant, the corresponding rules are simple local updates, as shown in this rule for lending a book:

```
TRANSACTION
  in {
    resource {"file:books.xml"},
    books {{
      book {{
        callNumber { var CALLNUMBER },
        state { "available" replaceby "lent" },
        emailLent { "none" replaceby var EMAIL }  }}  }}  }
```

```
ON
  xchange:event {{
      type { "lend book" },
      callNumber { var CALLNUMBER },
      email { var EMAIL } }}
END
```

The rules for prebooking and returning a book only differ in terms of the values replaced in the state element, and in that the e-mail address of the prebooker may also be subject to change. For other functionality that is connected with the prebooking and returning of books, like sending monitions to belated lenders, or sending notifications to prebookers in case of a return, see the following sections, 2.2.2 Monitions: composite events, and 2.2.3 Prebooking: complex updates.

If a book is not locally available, an event has to be sent to the management node. There exists a virtual library, i.e. a central list which keeps track of the whereabouts of every book. The event raising rule at the participants which sends such an request to the management node is the following:

```
RAISE
  xchange:event {{
    xchange:recipient { "http://www.ecss.org/" },
    requestingLibrary { "LMU" },
    type { "request book at management node" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
ON
  xchange:event {{
    type { "request book at management node" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
END
```

So the event is basically the same as discussed above and simply passed on. It is only supplemented by an element requestingLibrary, which identifies the participant to which the book has to be sent once it is located. The e-mail address of the member that initiated this book search is included, so that he can be notified once the transfer of the book to his participant is complete.

The virtual library at the management node keeps information about all books currently owned by the society, and their locations in an XML file:

```
<books>
  <book>
    <callNumber>01</callNumber>
    <title>
```

```
      Gerhard Adelmann: Die Baumwollgewerbe
      Nordwestdeutschlands ...
    </title>
    <library>LMU</library>
  </book>
  <book>
    ...
  </book>
  ...
</book>
```

The library element here states where a book currently is situated. It may be one of LMU, CEU, and Edinburgh. There is a separate rule at the management node for each participant, which checks for incoming requests for books from this participant. To determine if a request for a book should be passed on to a certain participant, conditions, or Web queries, are used in these rules. The rule for requesting a book currently at the LMU therefore looks like this:

```
RAISE
  xchange:event {{
    xchange:recipient { "http://history.lmu.de/" },
    type { "request book at participant node" },
    callNumber { var CALLNUMBER },
    email { var EMAIL },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
ON
  xchange:event {{
    requestingLibrary { var REQUESTINGLIBRARY },
    type { "request book at management node" },
    callNumber { var CALLNUMBER },
    email { var EMAIL }  }}
FROM
  in {
    resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        library { "LMU" }  }}  }}  }
END
```

If an event requesting a book with a certain call number reaches the management node, the condition of this rule – denoted by the keyword FROM – is evaluated next. Thus it will be determined if the book with that call number has the location LMU in the list of the virtual library. If this condition is met, the action part will be executed, that is, an event will be sent to the LMU with the instruction to deliver the said book to the requesting participant's library. So the condition part of a rule allows to check persistent data sources such as the list of the virtual library for a certain state.

A similar examination of persistent data takes place at the participant node which is asked to deliver the book ("delivering library"). When a request for a book arrives at a delivering library, and the book will be delivered, this has to be communicated to the requesting participant as well as to the management node. As a requirement, the book must be available – and not lent or prebooked – at the delivering library. This is again checked using a condition in the ECA rule, as can be seen in the rule that informs the management node about the delivery:

```
RAISE
  xchange:event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY },
    deliveringLibrary { "LMU" }  }}
ON
  xchange:event {{
    type { "request book at participant node" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
FROM
  in {
    resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        state { "available" }  }}  }}  }
END
```

Here, when an event requesting a book comes in, and if this book is available locally, the management node is notified about the transfer of that book from the LMU library to the requesting library. Note that such an examination of persistent data using a Web query is not necessary if the persistent data is to be updated by the same rule, as is the case in the rule changing the state of a transferred book at the delivering library:

```
TRANSACTION
  in {
    resource { "file:books.xml" },
    books {{
      book {{
```

```
        callNumber { var CALLNUMBER },
        state {
          "available" replaceby "not available" }  }}  }}  }
ON
  xchange:event {{
    type { "request book at participant node" },
    callNumber { var CALLNUMBER }  }}
END
```

This rule only updates the element state if its current value is available. This "condition" is enforced by the structure of the query term on the left side of replaceby. Other steps that complete this transaction are a notification from the delivering library to the requesting library, and then local updates of the availability of that book at the former requesting library now possessing it, and of its location at the virtual library list of the management node. These are performed with rules very similar to those already presented.

The main intention of the outline of the processes involved in inter-library loans that has been given in this section was to introduce further examples for the use of conditions in XChange rules. Conditions are queries of persistent Web resources. Besides checking the state of persistent Web data, they can of course also be employed to select certain data items and bind them to variables for use in the action part of the rule. While the Web resource to be queried is specified in the same way as resources in update specifications, the patterns used in conditions very much resemble those of event queries. This uniformity in all parts of the rules is one of the reasons for the ease of use of XChange.

2.2.2 Monitions: composite events

The processes of a library in some cases demand reactions to more complex events than just a single incoming message. An example for this are monitions. If a book is lent, the lending ECSS member will after some time first receive a notification about the end of the lending period. If the book is not returned within this period, a first monition will be sent, and later a second one. If the book is not returned, the library account of the corresponding member may be disabled. In all these cases it has to be checked if several events happened – or not happened, respectively – within a certain period of time. The issue of time and time-periods in which events must happen in order for an XChange rule to fire will be discussed in a later section in more detail (See 2.3.1 Registration of panel moderators: working with time). Event composition will be in the focus of this section.

In the simplest case, a composite event is the conjunction of two or more events with the keyword and. For example, to detect the lending and return of a certain book in the library, this could be defined in a rule as follows:

```
TRANSACTION
```

```
  ...
ON
  and {
    xchange:event {{
      type { "lend book" },
      callNumber { var CALLNUMBER },
      email { var EMAIL }  }},
    xchange:event {{
      type { "return book" },
      callNumber { var CALLNUMBER },
      email { var EMAIL }  }}  }
END
```

The action of this rule is executed if a certain book – identified by its call number – is lend and returned by a certain member – identified by his e-mail address. Using this notation, XChange does not care if the lending event happened before or after the return event. By employing the keyword andthen and square instead of curly braces, a certain chronology of events can be guaranteed. In the following example the lending has to take place before the return:

```
TRANSACTION
  ...
ON
  andthen [
    xchange:event {{
      type { "lend book" },
      callNumber { var CALLNUMBER },
      email { var EMAIL }  }},
    xchange:event {{
      type { "return book" },
      callNumber { var CALLNUMBER },
      email { var EMAIL }  }}  ]
END
```

If a member lends and returns a certain book twice, the above rule would be triggered three times, because at the second time the book is returned, two preceding lend events are existent.[10] To make sure that only lend and return combinations are detected that belong to the same business transaction of the library, one could use the event ID of the lending event as a library business transaction ID ("LBTID") number. This can be saved at lending time in the book list of the library together with the information that the book is lent. If it is returned later on, this information then would be retrieved from that list using a Web query, and inserted into the return event. Then the above rule can

[10]So events are not "consumed" by matching event queries.

be rewritten as follows:

```
TRANSACTION
  ...
ON
  andthen [
    xchange:event {{
      xchange:id { var LBTID },
      type { "lend book" },
      callNumber { var CALLNUMBER },
      email { var EMAIL }  }},
    xchange:event {{
      type { "return book" },
      lbtid { var LBTID },
      callNumber { var CALLNUMBER },
      email { var EMAIL }  }}  ]
END
```

This rule only detects lending and return events belonging to the same business transaction of the library.

For the purpose of monitions it is necessary to consider even more complex events, and especially event negation. What we are interested in, is the case that a book is lent but not returned after a certain period of time. In general, the keyword without denotes event negation in XChange, but at the moment XChange only allows this together with absolute and finite time intervals, as in the following rule:

```
TRANSACTION
  ...
ON
  without {
    xchange:event {{
      type { "lend book" },
      callNumber { "01" }  }}
  during [2006-08-01T00:00:00 .. 2006-08-31T00:00:00]
END
```

This rule only fires if the book with the call number 01 is never lent between the 1st and the 31st of August 2006. What is needed for monitions is a detection that although a book has been lent, the corresponding library business transaction was never completed with a return event in a certain period of time. Event queries for the detection of such behavior are not possible in XChange at the moment.[11] Therefore monitions are not implemented in the use case.

[11] This is suggested as an additional langauge feature, see section 4.4, Alternative constructs

There is a much simpler instance of a complex event query implemented in the meeting organization parts of on the management node (For this and the use of the within construct, see section 2.3.1 Registration of panel moderators: working with time).

The concept of monitions as presented in this section brought up some new issues: Composite event queries in XChange allow to specify patterns over more than one incoming event. The basic means for event composition are conjunction with the keyword and, and disjunction with the keyword or. Temporally ordered conjunctions are possible with the keyword andthen. Additionally, without denotes event negation, but must be complemented with a finite time interval or a composite event query to specify the period during which the event is excluded. Finally, while developing XChange programs one should remember that events are not consumed by rule firings in XChange.

2.2.3 Prebooking: complex updates

Conjunction – and disjunction – is not only useful in the event query part of an ECA rule but also for the definition of more complex updates in the action part. Consider the case that a book is lent by one member “A” and prebooked by member “B”. The XML data in the file books.xml on the participant's node (see 2.2 A virtual library) then looks as follows:

```
<books>
  <book>
    <callNumber>01</callNumber>
    <title>
      Gerhard Adelmann: Die Baumwollgewerbe
      Nordwestdeutschlands ...
    </title>
    <state>prebooked</state>
    <emailPrebooked>B@lmu.de</emailPrebooked>
    <emailLent>A@lmu.de</emailLent>
  </book>
  ...
</books>
```

If A returns the book, it is automatically lent to the prebooker B. Then this record has to be updated: The state changes from prebooked to lent; the e-mail address of B will be inserted in the element emailLent and replace that of A; the content of the element emailPrebooked will be deleted, as there is no person prebooking that book any longer. Additionally some notification should be sent to B that he now may pick up the book in the library. All in all, these are basically two actions that have to be executed on the same event: First, updating the book record on the participant node, and second, sending a notification to

for exclusions.

the person prebooking. This can be accomplished using a conjunction of update specifications – a complex update – in a rule as follows:

```
TRANSACTION
  and {
    in { resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        state { "prebooked" replaceby "lent" },
        emailLent {
         var EMAILLENT replaceby var EMAILPREBOOKED }
    }} }} },
    in { resource { "file:notifications.xml" },
    notifications {{
      insert notification {{
        type { "prebooked book now available" },
        addressee { var EMAILPREBOOKED },
        title { var TITLE } }} }} } }
ON
  xchange:event {{
    type { "return book" },
    callNumber { var CALLNUMBER } }}
FROM
  in
  {
    resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        title { var TITLE },
        state { "prebooked" },
        emailPrebooked { var EMAILPREBOOKED },
        emailLent { var EMAILLENT } }} }} }
END
```

If a book is returned, in the Web query, this rule first looks up if this book is of state prebooked. Evaluating the condition part, it also retrieves certain data from the file books.xml, like the e-mail addresses of the person previously lending the book, that of the one prebooking, and the title of the book.

The action part of this rule is more interesting. It comprises of a conjunction of two local updates. First, the book record in books.xml is updated to reflect the return of the book by member A and its lending to member B. Additionally a file notifications.xml is amended by a new record containing a notification for member B that the book is now available for him. notifications.xml can, for example, be processed by some page at the participants node displaying personal messages for members, or by a program sending such notifications as e-mails.

The prebooking function of this use case thus provides a very simple example for a complex update. Of course, other constructs of the language XChange as presented in previous sections can be used in complex updates as well. While a conjunction of updates specifies that all updates have to be executed, a disjunction would express that only one of the given updates should be processed. Square brackets allow to define the order in which updates have to take place.

2.3 Meeting organization

The management Web site of the ECSS allows to carry out some tasks connected to the organization of meetings of the society. Members can propose papers there, register for a workshop, or suggest a panel moderator. The panel moderator proposal must be confirmed by the suggested person afterwards. Besides presenting the corresponding functionality of the use case, this section introduces language constructs for working with time and discusses a situation in which composite events without event consumption are useful.

2.3.1 Registration of panel moderators: working with time

XChange provides different ways to add absolute or relative temporal restrictions to event queries. Absolute means that a finite time interval or point in time is given, such as "between the first and the 31st of August 2006", or "before the first of August 2006". A relative temporal restriction on the other hand relates to specifications such as "within 2 months". Used together with composite event queries, this means that the time interval for incoming events matching that query may not exceed two months.

An example for an absolute restriction in this use case is the possibility to register for a workshop. There is a deadline for registration, which is implemented as follows in the corresponding ECA rule:

```
TRANSACTION
  in { resource {"file:registrations.xml"},
    registrations {{
      insert registration { var REGISTRANT }  }}  }  }
ON
  xchange:event {{
    newRegistrant { var REGISTRANT }  }}
  before "2006-08-15T12:00:00.00"
END
```

The actual functionality is only that some registrants name which is sent by an event from the management Web site is inserted into the list of registrants for the workshop. The keyword before following after the event in the event query part of the rule shows that this event query is time-dependent. It restricts the firing of the rule to the time before the date given in quotation marks after the keyword. The format for the time is the ISO 8601 standard format for dates and times – the same as is used by XChange to denote a rising time in events.[12]

If registrations should only be accepted in a finite time interval, the rule could instead be written as follows:

```
TRANSACTION
  in { resource {"file:registrations.xml"},
    registrations {{
      insert registration { var REGISTRANT }  }}  }  }
ON
  xchange:event {{
    newRegistrant { var REGISTRANT }  }}
  in ["2006-05-15T12:00:00.00" .. "2006-08-15T12:00:00.00"]
END
```

Using the keyword in and a notation with two points in time in square brackets and conjoined to each other by two dots, the event query will be restricted to registrations arriving in the given time period, here between 15th of May and the 15th of August 2006, at 12 o'clock.

If composite events are used, relative temporal restriction become important. As the organizer of a conference, if there are lots of proposals for panel moderators, one might want to receive a confirmation by the suggested persons if they will take over the task. If such a confirmation does not arrive within some time after the proposal, it is probably safe to say that the candidate does not consent. Such rules can be depicted in XChange by using composite events and relative temporal restrictions:

```
TRANSACTION
  in { resource {"file:moderators.xml"},
    moderators {{
      moderator {{
        name { var MODERATOR },
        confirmed { "no" replaceby "yes" }  }}  }}  }
ON
  andthen [
    xchange:event {{
        moderatorProposal { var MODERATOR }  }},
    xchange:event {{
        moderatorConfirmation { var MODERATOR }  }} ]
  within 7 day
```

[12]See the discussion of the structure of events in 2.1.4 Information of other nodes in the system: propagation of updates, and the discussion of time format issues of the prototype in section ??, Problems with time formats.

```
END
```

Note that in the event query part of the above rule, a conjunction of events is specified. The conjunction is followed by the keyword within denoting a temporal restriction to a relative time interval. Thereby, the consent of a moderator candidate is noted (see action part of the above rule) if he responds within 7 days of the proposal with a confirmation. To this end the event query looks for two temporally ordered events, first the moderatorProposal and then the moderatorConfirmation. The firing of the rule is restricted to the case that both events arrive within 7 days. Possible other units instead of day for use with within are second, minute, hour, month, and year.

The registration functionalities of the management node thus provide insights in how temporal restrictions can be employed in event queries. XChange makes a distinction between absolute and relative specifications of time. The absolute specifications work with finite time points and intervals given as fixed dates and times. The keywords before and in allow to restrict event queries to events arriving before a certain point in time or in a given time interval, respectively. Relative specifications of time make only sense with composite event queries. within takes time periods such as “seven days” to denote relative temporal restrictions.

2.3.2 Registration of papers: composite events without event consumption

XChange features composite events without event consumption, which means the matching of some event query and the subsequent firing of the corresponding rule does not “consume” the event. Instead it is still available in memory for consideration of future event queries. As an example for the use of this behavior, scheduling papers at a meeting can be done by XChange. At the ECSS meeting there are sessions in the morning and in the evening of each meeting day. At each session, two papers will be presented. Incoming paper proposals are therefore grouped by category and by working group, to find matching papers for a session. A possible combination of papers is one where both papers have the same category or working group. Possible combinations can be automatically determined by an XChange rule as follows:

```
TRANSACTION
  in { resource {"file:paperCombinations.xml"},
    combinations {{
      insert combination {
               paper1 {
                 lecturer { var LECTURER1 },
                 title { var TITLE1 }  },
               paper2 {
                 lecturer { var LECTURER2 },
```

```
                 title { var TITLE2 }  },
               workingGroup { optional var WORKINGGROUP },
               category { optional var CATEGORY }  }  }}  }
ON
  or {
    and {
      xchange:event {{
      xchange:id { ID1 },
        newPaper {
          lecturer { var LECTURER1 },
          title { var TITLE1 },
          workingGroup { var WORKINGGROUP },
          category { var CATEGORY1 }  }  }},
      xchange:event {{
      xchange:id { ID2 },
        newPaper {
          lecturer { var LECTURER2 },
          title { var TITLE2 },
          workingGroup { var WORKINGGROUP },
          category { var CATEGORY2 }  }  }}
    where ID1 != ID2  },
    and {
      xchange:event {{
      xchange:id { ID1 },
        newPaper {
          lecturer { var LECTURER1 },
          title { var TITLE1 },
          workingGroup { var WORKINGGROUP1 },
          category { var CATEGORY }  }  }},
      xchange:event {{
      xchange:id { ID2 },
        newPaper {
          lecturer { var LECTURER2 },
          title { var TITLE2 },
          workingGroup { var WORKINGGROUP2 },
          category { var CATEGORY }  }  }}
    where ID1 != ID2  }  }
END
```

The action part of this rule just saves a possible paper combination to a file. The event part is not only more substantial but also more interesting: It includes a larger composite event query, consisting of an all-embracing disjunction marked by the keyword or, and two alternate conjunctions. So this rule fires if either the first or the second conjunction of events is detected. Both conjunctions demand two events announcing new papers. In the first case, these papers must have the same working group, which is enforced by using the same variable WORKINGGROUP in the pattern for both events. The second case is analogous, only that there the same category is required. Two incoming events announcing papers of the same category or of the same working group therefore are necessary for a firing.

A further requirement is of course that not a single event triggers the rule. This might happen in this case as the conjuncted events have the same structure. (So both event terms in the conjunction would match the same event.) It is prevented by using the where construct of XChange, which allows to give additional conditions which must be met by events in the event query. In the above rule a simple comparison of the IDs of the events makes sure that the rule is not triggered by a single event.[13]

As a result of this rule, each two of all proposed papers belonging to the same category or having the same working group allocation would be grouped together in a list. Thereby, filling the time slots in the programme of a meeting with matching papers becomes an easy task for the management office.

Thus, the function explained in this section is an example for the advantage of event queries without event consumption. At times of course a behavior is desired which would imply event consumption. The example of monitions in a library as presented in section 2.2.2 Monitions: composite events could be implemented easier if there was event consumption in XChange. But a similar result can be achieved by auxiliary means like the library business transaction ID in the mentioned section.

2.4 Miscellaneous features

As listed in section 1.4 A portal for a distributed research community, the information system of the ECSS contains a mechanism to propagate news filtered from newsletters on the management node to the working group node. Additionally, this section covers the administration of publication data in the system. While the part about the propagation of content from e-mails takes up many of the aspects discussed already in previous sections, that on publication management describes a situation which at the moment cannot be addressed by the means of XChange, and thus hints at a desired language feature (see also section 4.4, Sorted insertions)

2.4.1 Propagation of content from e-mails: routing of events

As service of the management office, the working groups of the ECSS are provided with news that may be relevant for their research. This news is displayed on the home pages of the working groups (see “News” section in the lower part of the following screenshot):

[13] Due to problems of the prototype, this rule does not work properly at the moment (See **??**, Problems with where-clauses.

The news is gathered from newsletters which are subscribed by the management node and received regularly. Upon receipt they are filtered for relevant news which is then categorized as announcements about new books, new reviews, or upcoming conferences, as well as assigned to different working groups by its topics (For details on this, see section 3.3 Subsidiary PHP functions). The management node Web site then receives them as events of the following structure:

```
<xchange:event>
  <type>newsletter</type>
  <subtype>review</subtype>
  <workingGroup>nation of government</workingGroup>
  <message>some review</message>
</xchange:event>
```

Note that these events are very similar for all types of news. While the element type in these events distinguishes them from other events arriving at the management node, the element subtype characterizes what kind of information is contained in the news. This can be one of:

- book
- review

- talk

The actual content of the news is conveyed the element message, while workingGroup denotes the node the news has to be propagated to. For each working group there are event-raising rules registered with the management node, that route these events to the proper working group Web site:

```
RAISE
  xchange:event {{
    xchange:recipient { "http://www.nation-government.net/" },
    type { var SUBTYPE },
    var MESSAGE }}
ON
  xchange:event {{
    type { "newsletter" },
    subtype { var SUBTYPE },
    workingGroup { "working group: nation and government" },
    var MESSAGE -> message {{}} }}
END
```

In the instance of the above rule, news that has been detected to be of interest for the working group "nation and government" is passed on to the corresponding URL, http://www.nation-government.net/. On the working group nodes, a set of rules saves the incoming messages to XML files for display on the home page.

```
TRANSACTION
  in { resource { "file:news_reviews.xml" },
    news {{
      insert entry {
        message { var MESSAGE } } }} }
ON
  xchange:event {{
    type { "review" },
    message {{ var MESSAGE }} }}
END
```

The above rule for example receives new reviews and stores them in news_reviews.xml.

This functionality for routing events is another example for the propagation of updates besides the member management and library functions already presented in sections 2.1 and 2.2. The events detected by rules on the management node in this case are not triggered by Web forms, but by incoming e-mails. The XChange rules at the management node then decide which working groups are possibly interested in these events, and pass them on by means of event-raising rules. The working group nodes, finally, provide a set of rules to add news taken from these events to the data which is displayed in the news sections of their home pages.

2.4.2 Publication management: maintaining order

The participants' Web pages allow for the entry and display of publications of the employees. The corresponding Web form is linked from the pages showing the details of individual employees. The Web form looks as follows:

Collected data comprises of the title of the publication, the publishing company, and the year. The author of the publication is automatically set to the name of that employee from whose details page the Web form was called.[14]

The data is sent in an XChange event first to the local participant's server, and then passed on to the management node. (This is very similar to the propagation of changes in member details, see section 2.1.4 Information of other nodes in the system.) While the participants' Web sites display publications at the personal pages of the authors, the management node provides a central list of ECSS publications. The events notifying the management node of new publications are structured like this:

```
<xchange:event>
  <type>insert publication</type>
  <publication>
    <surname>Sandoz</surname>
    <firstName>Ellis</firstName>
    <title>
      A Government of Law. Political Theory, Religion, ...
    </title>
    <publishingCompany>
```

[14] As publications are fixed to one specific employee, it is not possible to assign more than one author to a publication. The rationale for this is not a technical one, and the implementation could easily be changed to allow several authors. As in the historical sciences – different from informatics – monographs by a single author are the rule, this seemed unnecessary, though.

```
      University of Missouri Press
    </publishingCompany>
    <year>2001</year>
  </publication>
</xchange:event>
```

So besides the information entered at the Web form – title, publishing company, and year – first name and surname of the author are specified, too.

The list of publications at the management node should maintain order in that entries are sorted by the authors' surnames, first names, and then by the titles of the books. This was intended to be accomplished by an XChange rule with an appropriate where-clause, as follows:

```
TRANSACTION
 in { resource { "file:publications.xml" },
     publications {{
       publication {{
         surname { var PREVIOUSSURNAME },
         firstName { var PREVIOUSFIRSTNAME },
         title { var PREVIOUSTITLE }  }},
       insert var PUBLICATION,
       publication {{
         surname { var FOLLOWINGSURNAME },
         firstName { var FOLLOWINGFIRSTNAME },
         title { var FOLLOWINGTITLE }  }}  }}  }

   where and { and { SURNAME >= PREVIOUSSURNAME,
                     FIRSTNAME >= PREVIOUSFIRSTNAME,
                     TITLE >= PREVIOUSTITLE },
               and { SURNAME <= FOLLOWINGSURNAME,
                     FIRSTNAME <= FOLLOWINGFIRSTNAME,
                     TITLE <= FOLLOWINGTITLE } }
ON
  xchange:event {{
    type { "insert publication" },
    var PUBLICATION -> publication {{
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE }  }}  }}
END
```

In the action part of this rule, the surname, first name, and title taken from the event are compared in a where-clause with currently existing data in the file publications.xml. The pattern in the action part assigns the new publication a place in between other publications by alphabetical order. This behavior is at the moment not available in XChange, as where-clauses in the action part are not yet supported. Therefore, corresponding suggestions have been made in the section or this report concerned with desired language features. (See sections 4.4, Non-structural conditions in update specifications and 4.4, Sorted insertions.)

As the last part of the section of this report concerned with the implementation of the use case, this part first presented another situation where event data has to be propagated. In this case the data is not entered by users, but taken from incoming e-mails. Plus, the administration of publications on the management node afforded the opportunity to hint at a feature not yet possible with XChange, sorted insertions.

The main intent of section 2 Implementation in XChange was to give an overview over the applications included in this use case. Second, basic XChange constructs have been introduced in an informal way by means of code examples from the ECSS information system. The remainder of this report will first have a look at auxiliary scripts implemented mainly for demonstration purposes, and then turn to an extended conclusion which also lists experiences from working with the prototype and makes suggestions for additional language features.

3 Auxiliary tools for demonstration

Besides the files containing XML data and those containing the rules, the bulk of files of this use case are auxiliary scripts that provide graphical interfaces and views. If the use case is run on a single server, the files for each node are stored in separate directories:

- lmu - Participant LMU
- ceu - Participant CEU
- edinburgh - Participant University of Edinburgh
- management - ECSS management office
- wg2 - Working group “Nation and Government”
- wg3 - Working group “Church and Reformation”

For a discussion of how to adapt the use case to a distributed environment, see A Running the use case in a network.

On the participant nodes, PHP scripts are employed for displaying information about employees, publications, and books (= the local libraries, discussed in section 2.2 A virtual library.) To the management node, the employees of the participants are “members” (of the ECSS). This node basically works with the same type of information as the participants (members, publications, books), plus that of the internal tasks of the management office, namely papers, moderators, and registrations (for a meeting or workshop). For the working group nodes, scripts for the display of news are added to those for members and publications. Some of the PHP functions used for display and interfaces as well as those for the logging of events are discussed in greater detail below.

Furthermore, for demonstration purposes there are simple PHP and shell scripts resetting the data to its initial state. While the data the use case works with is contained in files with the suffix “.xml”, the “original” data (mostly empty XML structures) is saved in files ending in “.save.xml”. The scripts reset.sh and reset.php replace the “.xml”-files with the original versions.

Finally, there is the Perl script deploy.pl written by Michael Eckert, which adapts URLs and addresses in the rule files to the network environment actually used for demonstrations of the use case. Its functionality and configuration is subject of section A Running the use case in a network.

3.1 Logging of firings and visualization of events

For demonstration purposes, besides showing the Web pages of the use case, it is helpful to present overviews over events as they are sent and received by the different nodes, propagated to other nodes, and result in certain updates. To this end, rule firings are logged in this use case, and the log is displayed as a Web page.

On the firing of every XChange rule, a log entry is written. There are different log files on all implemented nodes, that is for the participants' Web sites of the LMU, CEU, and University of Edinburgh, for the management node, and for the nodes of working groups 2 and 3. Log files are of an XML format, and thus it is possible to maintain them by updates using XChange. For this purpose, all the update rules in the use case consist of a conjunction of updates, one for the original task of the rule – adding / changing / deleting member records for example – and one for adding log entries to the log file. Their structure therefore is always like this example of the rule inserting an employee on the LMU node:

```
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "LMU website" },
                 event { "insert employee" },
                 reaction { "local update of employee data" }
                 }  }}  },
    in { resource {"file:staff.xml"},
      ...  }  }
ON
  ...
END
```

As event raising rules cannot perform updates simultaneously, each of them is complemented with an update rule for writing the log.

The first update in the action part of the above rule writes the log entry to the file log.xml. The log file always has a root element labeled log and one child element entry for each log entry. Logged information consists of the sender of the incoming event that triggered the rule, a description of the event itself, and a description of the reaction the rule specifies. In the above example of the rule inserting an employee on the LMU node, as it concerns a local update, the sender is the LMU Web site. All events in this use case have a certain type, which also is announced in the events in their type element. The received event in this case is called "insert employee", and this is also the description of the event in the log. Finally, the reaction characterizes what the rule does if fired. An example log file thus looks like this:

```
<log>
  <entry>
    <sender>LMU website</sender>
    <event>insert employee</event>
    <reaction>
```

```
      raise insert employee on management node
    </reaction>
  </entry>
  <entry>
    ...
  </entry>
  ...
</log>
```

The log file is read and displayed as HTML in a browser by the script log.php. To allow continuous tracking of the events on screen, log.php periodically reloads itself using the construct

```
<meta http-equiv="refresh" content="5; URL=log.php">
```

Sender, event, and reaction are displayed as the three columns of a table. New entries are displayed on top of the list, so the chronology of fired rules is from bottom to top. The display of a typical log file at the LMU server thus looks as follows:

sender	event	reaction
LMU website	delete employee	local update of employee data
LMU website	delete employee	raise delete employee on management node
management node	initiate book delivery	local update of availability
management node	initiate book delivery	raise book location update on management node
management node	initiate book delivery	raise book availability update on Edinburgh University node
LMU website	insert employee	local update of employee data
LMU website	insert employee	raise insert employee on management node

3.2 Displaying rules

For demonstration, it is also often necessary to bring up some rule of the use case on screen. Although this could be done by opening the rule files in some editor, for convenience there is the script displayRules.php, which on all nodes displays the rule file as a Web page. The rule files are text files named rules.xchange, one for every node. At the beginning of each of them, a table of contents is given in comments (The character for commenting out lines in XChange is "#"). Calling displayRules.php first reads and displays this table of contents. All rules in a rules file are numbered with a consecutive number preceded by the character "R". Clicking on the link of one of these numbers will bring up the corresponding rule on screen. The script finds rules in the text file by looking for the comment stating the rule's number and description at the beginning of each rule. A typical table of contents of a rule file in this use case is the following:

```
################################################################################
# Table of Contents                                                            #
#                                                                              #
# R001 - insert employee                                                       #
# R002 - delete employee                                                       #
# R003 - change employee                                                       #
# R004 - insert publication                                                    #
# R005 - propagate insertion of a new employee to management node              #
# R006 - propagate deletion of employee to management node                     #
# R007 - propagate change of employee to management node                       #
# R008 - propagate insertion of a new publication to management node           #
# R009 - lend book                                                             #
# R010 - prebook book                                                          #
# R011 - raise request book on management node                                 #
# R012 - update local data: book sent                                          #
# R013 - update local data: book received                                      #
# R014 - transfer book: inform management node                                 #
# R015 - transfer book: inform requesting library (CEU)                        #
# R016 - transfer book: inform requesting library (Edinburgh)                  #
################################################################################
```

3.3 Subsidiary PHP functions

The use case contains a number of subsidiary PHP scripts to provide Web interfaces, send data from the Web interfaces to the XChange server, and to display data from XML files. The most important of these is sendXChangeEvent.inc.php, a short script for sending XChange events to a server. sendNewsletter.php and cronFetchMail.php are used to demonstrate the routing of newsletter content to the home pages of the working groups (See 2.4.1 Propagation of content from e-mails: routing of events). The choice of PHP for these tasks is arbitrary. Other languages could have been employed as well, for instance a combination of CSS and XSLT for the transformation of the XML data to Web pages. The following discussion is to some extent concerned with technical details, which are of lesser importance for the use case and only given to support future maintenance of the application.

sendXChangeEvent.inc.php

In many parts of this use case, data for local updates is gathered by Web forms. For every Web form, there is a script which creates event data in XML format from the input. Those events are subsequently sent to the local XChange server. The sending is the task of the function sendXChangeEvent. It accepts a string containing the XML data of the event as argument. The PHP socket functions are used to establish the connection to the server. Note that this has nothing to do with the sending of events done by XChange, for instance to propagate events to other servers. This script is only needed to initiate local updates by sending form input to the local XChange server.

readXmlFile.inc.php

This file contains the function readXmlFile, which accepts a filename as parameter and returns a two-dimensional array. It reads XML data from the file using the built-in XML parser of PHP. The returned structure consists of one index array and one values array, with the index array containing pointers to the location of the appropriate values of the values array.

xml.inc.php

This files contains several functions operating on the structure returned by readXmlFile. The first, getNextTagContent returns the content of the next tag following on a tag whose index is specified as an argument. Assume that some XML data consists of "records" like the employee records in the following example:

```
<staff>
  <employee>
    <surname>some surname</surname>
    <firstName>some first name</firstName>
    <office>some office</office>
  </employee>
  <employee>
     ...
  </employee>
   ...
</staff>
```

All tags in this example are indexed by the XML-parser of PHP, with <staff> having index 0, the first <employee> having index 1, <surname>...</surname> having index 2, index 5 for the first </employee> closing tag, and index 6 for the second <employee> opening tag. That is, "complete" tags – those having no sub-tags – get an index and the opening and closing tags of "incomplete" tag pairs get an index each.

To retrieve the first name of the first employee, one can use getNextTagContent if the index of the opening tag of the record is known (here: 1). The tag whose content shall be fetched must be a complete tag.

The other functions in this file all operate on similar principles. getNextTagIndex returns the index of one tag following on another tag given by its index. getIndexClosingTag returns the index of the corresponding closing tag of a given opening tag according to the indices array. getContentsSubtags returns an array containing the contents of all sub-tags of an incomplete tag given by its index. The sub-tags must all be complete. As an example consider the following XML data:

```
<workingGroups>
  <workingGroup>The Enlightenment</workingGroup>
  <workingGroup>Nation and Government</workingGroup>
  <workingGroup>Church and Reformation</workingGroup>
</workingGroups>
```

getContentsSubtags with index number 1 as argument in this case returns:

```
Array (
  [0] => "The Enlightenment"
  [1] => "Nation and Government"
  [2] => "Church and Reformation"
)
```

displayXml.inc.php

displayXml is a small function accepting XML code as an argument and masking XML specific characters with HTML entities in order to display them in a browser. (Otherwise they would be interpreted as HTML elements.) Additionally, line breaks and space characters are converted to their HTML counterparts, so that the format of displayed XML data is preserved (For example indented lines).

sendNewsletter.php and cronFetchMail.php

These two scripts are used to send and receive newsletters, information of which is then processed on the management node and passed on to the working group nodes for display on their home pages. sendNewsletter.php provides a Web form for sending e-mails. The latter is done by the PHP mail-function. A sample newsletter is already contained in the Web form, as an example of the required format. sendNewsletter.php is independent of any special node and can be used on every server. As an alternative, newsletters can also be sent from any mail client to the e-mail address reserved for the use case.

cronFetchEmails.php retrieves mails from a POP3-mailbox, analyzes their content, and constructs and sends XChange events from it. The IP address of the mail server and the user name and password of the POP3 account must be specified in the head of the file. This script is supposed to be installed as a cronjob on the server of the management node. Calling it every minute has been turned out to be sufficient for demonstration purposes. (For the setup of the use case and the crontab entry, see A Running the use case in a network.)

E-mails sent for processing by this script should be newsletters of a certain format. Currently three types of news are recognized: Information about newly published books, new reviews, and announcements of talks and workshops. The newsletter is considered to contain separate sections for this information, introduced by the headings:

- "NEW THIS MONTH" (referring to books),
- "REVIEWS THIS MONTH", and
- "WORKSHOPS AND TALKS THIS MONTH".

News items in each section should start out with the character "*". Each news item is scanned for the terms "church", "reformation", "nation", and "government" in order to decide which working groups might be interested in them (The two working groups implemented in the use case are "Church and Reformation" and "Nation and Government".) An example newsletter might look as follows:

```
Newsletter 03/2006 03/01/2006

:::::::::::::::::::::::

NEW THIS MONTH:

* Thomas M. Magstadt: Nation and Government : Comparative Politics
in Regional Perspective (with CD-ROM) (Paperback), Wadsworth
Publishing, ISBN 0534631231.

* John D. Hannah: Charts of Reformation and Enlightenment Church
History. Church and Reformation in the 18th century. Zondervan, ISBN
0310233178

:::::::::::::::::::::::

REVIEWS THIS MONTH:

* Rainer Mentel: Building Scotland's Church. Das sechste Jahrhundert
schottischer Kirchenbaukunst (1725-1800), Weimar: Verlag und
Datenbank f\"{u}r Geisteswissenschaften 2002, ISBN 3-89739-281-x.
Reviewed by: Ute Engel in sehepunkte 3 (2003), No. 7/8

* Nation and Nationality in Russia. The process of integration of
sibirian tribes, 1722-1825, Stuttgart: Franz Steiner Verlag 2001,
ISBN 3-515-077138. Reviewed by: Wim van Meurs in sehepunkte 2
(2002), No. 11

:::::::::::::::::::::::

WORKSHOPS AND TALKS THIS MONTH:

* The change of English government during the time of the French
```

```
Revolution. Talk by Jonathan I. Israel on the seventh international
colloquium on 18th century history at Princeton University.
(03/18/2006)
```

The XChange event for a single news item constructed from such a newsletter is:

```
<event>
  <xchange:recipient>http://www.ecss.org/</xchange:recipient>
  <type>newsletter</type>
  <subtype>book</subtype>
  <workingGroup>nation and government</workingGroup>
  <message>
    Thomas M. Magstadt: Nation and Government ...
  </message>
</event>
```

A discussion of those parts of the use case employing these functions can be found in 2.4.1 Propagation of content from e-mails: routing of events.

All the subsidiary PHP functions presented in this section are in no way necessary for the functioning of the XChange programs that form the main part of the use case. Their purpose in most cases is simply to allow for a more convenient display of data, or to provide Web forms as interfaces for user input. The scripts for keeping the log files and displaying the content of rule files exist only due to demonstration purposes.

4 Conclusion

The research project that led to the development of XChange is in progress. The same applies to work on the prototype, which is still continuing. As is natural in a state of ongoing research, there are some problems with the prototype that have been discovered during the implementation of this use case. Additionally, some suggestions for improvements of the prototype and new language features are part of this conclusion.

4.1 Not yet or not correctly implemented language features in the prototype

This section lists a few XChange language features that are not yet or not correctly implemented by the prototype.

Time formats

In some cases the format used to describe time points and durations does not comply to the XChange language specifications. In general, there are three different formats for time in XChange: for finite time intervals, for relative time intervals or durations, and for single points in time.

A single point in time is given in the ISO 8601 standard format for dates and times (See [4] for details), which is the following:

```
YYYY-MM-DDThh:mm:ss
```

This is the year in four digits, followed by the month and day with two digits each, and separated by hyphens. A capital T parts this date specification from the time specification, which is given as hour, minutes, and seconds – also two digits each, but separated by colons.

Finite time intervals are given as two points in time separated by two dots and enclosed in square brackets:

```
[YYYY-MM-DDThh:mm:ss..YYYY-MM-DDThh:mm:ss]
```

The first point in time denotes the beginning, the second the end of the interval. The second point in time must be later than or equal to the first one.

Relative time intervals are given by one of the keywords second, minute, hour, day, month, and year, preceded by a number. An interval of seven months for example would be noted as follows:

```
7 month
```

The prototype in some cases uses other formats than those. The relative time interval after the keyword within for example uses a Haskell-like syntax at the moment. A period of seven months in this syntax is given as follows:

```
"TimeDiff {
   tdYear = 0, tdMonth = 7, tdDay = 0, tdHour = 0,
   tdMin = 0, tdSec = 0, tdPicosec = 0 }"
```

Note that additional to the Haskell-style there are double quotes limiting this specification. The original XChange syntax does not require double quotes.

The point in time used with the before construct must be given at the moment with additional double quotes and an additional two digits number as last specification separated from the seconds by a dot (which probably denotes the hundredths of a second). It thus looks like this:

”YYYY−MM−DDThh:mm:ss.cc”

Other problems with this time specification, as inverted declaration of day and month, and missing interpretation of seconds have been fixed in the course of the project.[15]

The syntax for finite time intervals does not work at all at present. If it is used in a rule, for example together with the in construct, it throws a parse error (Adding double quotes does not help in this instance, so the problem obviously lies with parsing the time format itself).

Prefixed namespace declarations

Namespace declarations are not supported by the prototype. This applies to event queries as well as to update specifications (and probably also to condition queries, although that has not been tested). Events containing namespace declarations are not recognized by the prototype, and updates specifications containing them are not executed, but result in a error message that the given namespace is “not representable as an attribute”. The content of the file to be updated is lost in such cases.

where-clauses

According to the language specification, where-clauses are allowed in the event query as well as in the Web query.[16] The idea is that non-structural conditions are subsumed in a condition box introduced by the keyword where. If for example in an event query the content of some variable should be greater than a certain value, say 7, this could be written as follows:

```
ON
  event {{
    number { var VARIABLE }  }}
  where { var VARIABLE > 7 }
```

Unfortunately, such conditions are not evaluated by the prototype. So the event query in the above example matches even if the number contained in VARIABLE is lower than 7. Additionally, where currently is not supported in Web queries. Using it there results in a parse error.

[15] By Michael Eckert, one of the developers of the prototype

[16] See [5], p. 72 (event query), and p. 94 (Web query)

Unsupported Xcerpt constructs

XChange employs certain querying constructs which are provided by the Xcerpt querying language it is built upon. Among them is for example the except construct, which allows to leave out subterms from a variable binding.[17] This does not work at the moment, because XChange uses an older Xcerpt version, which does not support that construct.[18]

4.2 Discovered problems with the prototype implementation

This section explains which problems have been discovered while working with the prototype. Most of them are parser or update problems.

"Locked" files

In the following data, there is only one element beneath the root element:

```
<data>
  <content>abc</content>
</data>
```

The following rule should delete the element content:

```
TRANSACTION
 in { resource { "file:data.xml" },
    data {{
      delete content { "abc" }  }}  }
ON
  event {{}}
END
```

This does not work. Instead an error message that the file is locked is displayed, although this is not the case. Only if there are other elements on top of and below the element content, the action is executed as expected. The above rule works with the following contents in data.xml:

```
<data>
  <additionalContent>123</additionalContent>
  <content>abc</content>
  <additionalContent>123</additionalContent>
```

[17] See [5], p. 70, and example 2.23 on p. 23.
[18] Source: Personal communication with Tim Furche

```
</data>
```

A similar behavior occurs using replaceby, but only if variables are employed in the rule. The following data:

```
<data>
  <oldContent>abc</oldContent>
</data>
```

is replaced without problems by this rule:

```
TRANSACTION
  in { resource { "file:data.xml" },
    data {{
      oldContent { "abc" } replaceby newContent { "ABC" }
    }} }
ON
  event {{
    oldContent { "abc" } }}
END
```

If some element in the event query is bound to a variable, the rule refuses to work:

```
TRANSACTION
 in { resource { "file:data.xml" },
   data {{
     oldContent { "abc" } replaceby newContent { "ABC" }
   }} }
ON
  event {{
    var VARIABLE -> oldContent { "abc" } }}
END
```

This rule results in an error message as mentioned above, although the data file is not locked. Note that the variable need not be used in the update specification. It suffices to bind some content in the event query to a variable, as shown in the above rule, to produce this behavior.

Rule formatting

There is an error dependent on textual rule formatting. Different from other examples, space characters and line breaks do matter in the following rule. Note that there are no space characters or other "invisible" characters at the end of any row in the rule:

```
TRANSACTION
  in
  {
    resource { "file:data.xml" },
    rootElement
    {{
      insert var ELEMENT
    }}
  }
ON
  event
  {{
    var ELEMENT -> element {{}}
  }}
END
```

For testing, there should also be no other rule in the rules file, and no empty lines or other characters than those shown above. In this case the rule file containing the above rule consist of 171 bytes (including line breaks). Although this rule does not contain any errors, it is not registered on starting the XChange server.

Strangely enough, it suddenly works if any minor change is introduced to the code. It suffices to add a space character after some line, to rename rootElement to root, or to give a complete pattern for element in the event query to make this rule run without problems:

```
TRANSACTION
  in
  {
    resource { "file:data.xml" },
    rootElement
    {{
      insert var ELEMENT
    }}
  }
ON
  event
  {{
    var ELEMENT -> element { "content" }
  }}
```

```
END
```

The above code works. Compare the pattern for element in line 13.

Inserting into data with many identical elements

The following XML data stored in the file data.xml contains a root element called data and three identical item elements:

```
<data>
  <item>content</item>
  <item>content</item>
  <item>content</item>
</data>
```

Consider the following rule to work on that data:

```
TRANSACTION
 in { resource { "file:data.xml" },
   data {
       item { "content" },
       insert item { "even more content" },
       item { "content" }  }  }
ON
   ...
END
```

In the action part of this rule a complete pattern is specified using single curly braces. If the rule is triggered, expectations are that no insertion takes place as the pattern in the rule only matches a root element data with exactly two item elements as children – while the element data in data.xml has three children. Nonetheless the update is executed, and in doing so, one of the existing child elements in data.xml is lost. The outcome therefore is:

```
<data>
  <item>content</item>
  <item>even more content</item>
  <item>content</item>
</data>
```

Using the same rule on the same data, but with double curly braces also does not yield the expected result. Instead, the existing item elements are multiplied. The rule in this case is:

```
TRANSACTION
 in { resource { "file:data.xml" },
    data {{
      item { "content" },
      insert item { "even more content" },
      item { "content" }  }}  }
ON
  ...
END
```

The result of the operation is a lot more item elements than in the original data:

```
<data>
  <item>content</item>
  <item>content</item>
  <item>content</item>
  <item>even more content</item>
  <item>content</item>
  <item>content</item>
  <item>content</item>
</data>
```

Using double square brackets leads to a similar wrong result, with less new elements:

```
<data>
  <item>content</item>
  <item>content</item>
  <item>even more content</item>
  <item>content</item>
  <item>content</item>
</data>
```

Finally, with single square brackets, the data is not changed at all, but XChange issues an error message. If the persistent data is provided in such a way that it matches the pattern of the update specifications exactly (that is, two instead of three item child elements), similar problems occur, with fewer new child elements. If in this case single square brackets are used, the prototype complains about data.xml being locked, which is not the case.

Function showXcerpt

In the following rule, the argument on the left side of the replaceby construct is marked as optional:

```
TRANSACTION
  in { resource { "file:data.xml" },
    rootElement {{
      element {
        optional var OLD replaceby "replacement" }  }} }
ON
  event {{
    old { optional var OLD }  }}
END
```

This results in an error message informing about problems with the function showXcerpt:

> Xcerpt/Xcerpt/Show/XcerptV2.hs:(36,14)-(145,30): Non-exhaustive patterns in function showXcerpt

The presence of optional in the event query has no influence on this behavior. Also, using this keyword and a variable for the second argument – the replacement – works without problems. Using optional with the first argument of replaceby in this use case would be helpful for example for updating employee data. If some data field of an employee record – like the academic title of the employee – is left blank at first and completed later on, a rule for this update needs to replace the – previously empty – title information.

all construct

Consider the case that an event contains an unknown number of data items which have to be inserted into the XML data of a file. The data items in the event are subelements of the root element:

```
<event>
  <item>item1</item>
  <item>item2</item>
  <item>item3</item>
   ...
</event>
```

All item elements have to be copied to the file. The following rule is employed:

```
TRANSACTION
 in { resource { "file:data.xml" },
    data {{
       insert all var VARIABLE  }}  }
ON
  event {{ var VARIABLE }}
END
```

This rule causes a parse error.

4.3 Suggestions concerning the prototype

Working with the prototype brought up some suggestions for improvements, which are listed in this section.

Rule management tool

Useful for the development of XChange programs and later maintenance would be a rule management tool. While the XChange server is running, this tool should provide the facilities to display a list of all rules currently registered, and to add, delete, and change rules. Other features might comprise a view of the event history, rule firings, and similar runtime data. Some of this information can be retrieved at the moment by running the prototype with certain switches, but only in a less than convenient way, and with crude outpoint at times. [19]

Notation of comparison operators

A source of minor inconvenience are the comparison operators for use in where-clauses, whose notation given in the language specification is different from that implemented in the prototype. They are gt() instead of >, lt() instead of <, and eq() instead of =. For instance an event query requiring a certain variable to be a number greater than, say, 7 must be written as follows in order to avoid a "lexical error":

```
ON
  event {{
    number { var VARIABLE }  }}
  where { gt(var VARIABLE, 7) }
```

[19] [3] discusses similar tools for ADBM systems.

Format of updated XML data

At the moment, the format of the XML data which is manipulated by XChange rules is not very concise. A minor inconvenience is that every element is complemented by Xcerpt specific attributes such as ordered and total. As XChange writes data files anew for every update – even if its only a minor update like the deletion of a single element – those attributes are added to all elements as soon as the file containing the data is written by XChange. Therefore, especially large data sources get rather bulky and confusing when they are updated.

A second disadvantage is that the prototype does not keep the order of the elements in given XML data, except if it is explicitly enforced by using square brackets in the rules. As it is usually much more convenient to use curly braces and incomplete patterns, the order in data sources is messed up with the first XChange update. This not only makes it difficult to compare XML files before and after changes made by XChange during development, it also makes it impossible to process files manipulated by XChange with applications which anticipate a given order.

Confusing error messages

Error messages issued by the prototype are currently not very telling. In section 4.2 it has already been said that the prototype often misleadingly complains about "locked" files, although the problem is located somewhere else. The same error message appears if the XML in the data file is not valid, for example because elements are not closed or parts of tags are missing (for example missing closing brackets). If the prototype is run with switches for displaying output, such as -d (for debugging information), error messages often appear because the "show"-functions which should display the output on screen are incomplete.

Unreadable output

If the prototype is run with switches for displaying runtime information, such as -d for debugging information, the output often is unreadable. Obviously several threads are printing their output to the screen simultaneously, which results in a complete mess. This gets worse with a larger number of registered rules.

4.4 Desirable language constructs

During the development of the use case, some projected functionality could not be implemented because an appropriate language construct was missing. Additionally, some of the suggestions for new language features presented here may considerably simplify the development of applications with XChange.

XChange and current real Web applications

XChange has been developed as an academic proposition for a language allowing reactivity on the Web. The design of the language is inspired by certain assumptions, among them an aspiration for elegance and ease of use. Both of these assumptions have been fulfilled by the language in a sense. The focus on XML data to work with in all parts of the employed ECA rules makes XChange programs look very uniformly and being of a piece. XChange is easily learned and XChange rules are easily written, as for event queries, Web queries, and transaction specifications the same constructs are used, and with XML the same data format is employed for events and persistent data. Using XML for all tasks a reactive language has to accomplish, makes it possible to concentrate on a single query language, and with Xcerpt on a very convenient one.

But there are tasks which XChange at the moment cannot perform, at least not without help. The ECSS use case contains some of them. First, in many cases it is necessary to react to events in other ways than by simply updating text files containing XML data. The simplest case may be sending an e-mail. In this use case, e-mails could be sent as notifications of books being ready for pickup at a library. Such an event would trigger an update of a certain file, which is amended by a line denoting recipient, message, and other necessary data. Using utilities of the operating system, a cronjob has been set up which runs a PHP script every few minutes. This script checks the file for new entries made by XChange, sends the corresponding e-mails, and deletes processed entries from the file. Altogether this is a quite long winded workaround. (An alternative is the programming of a special server – not an XChange server – that receives corresponding XChange events and then sends the e-mails. This avoids the periodic polling, but involves system programming which is maybe slightly more complicated than the above solution.)

Second, use cases like the one described in this paper, but also many others that are often referred to in the context of XChange, such as the travel organizer or the car rental use case would typically rely on databases for the storage of persistent data. A likely choice for the ECSS use case would have been a MySQL database for example. So for all data manipulation in the transaction parts of XChange rules, and also for all Web queries, there should be some interface to operate with databases. That the use of databases such as MySQL is widespread in today's Web definitely is rather due to historical development than to technical feasibility. Nonetheless, as XChange aims at applications situated in this environment, it should be integrated into it as seamlessly as possible.

Another feature for transactions that might be more interesting than is obvious at first glance is the manipulation of XChange rules itself, and especially of those currently registered with the server. This requires a possibility to define meta-rules that change the behavior of an XChange server accordant to incoming events. A simple example is changing time points and intervals in temporal restrictions to rules.

Probably by simply introducing a way to call arbitrary programs (and pass over event and Web query data to them as arguments) in the action part of ECA rules would make XChange a lot more usable. This would allow for sending e-mails, updating databases, and many other actions, and is probably easy to introduce to XChange. A similar approach could be pursued for the Web queries

– here only an additional facility is needed to return data from programs to XChange.

All in all, while XChange code itself at the moment indeed is elegant and easy to use, developing applications with XChange makes it sometimes necessary to resort to not so elegant solutions outside of XChange. Providing interfaces for arbitrary programs (and maybe databases) may take away some of the clarity of the language XChange, but add greatly to the elegance and usability of XChange as a reactive language for real applications.

Order of evaluation of rules

The order in which rules registered with XChange are evaluated is not defined by the language. The prototype executes them in the same order as they appear in the rules file, but this is not a required behavior. For certain applications it might be useful to define this order. An example for such a case is given in section 2.1.4 Information of other nodes in the system: propagation of updates. In this example an event informing about changes in a member record comes in at the management node. There are two different rules that react to this event. One of these rules updates the member data stored on the management node. The other rule raises a new event, but for this relies on a Web query of exactly the same member data as is updated by the first rule. For maintaining data consistency, the event raising rule in this case has to be evaluated prior to the update rule. A definition of the order in which events are evaluated, or a language construct to influence this behavior would be helpful.

Alternative constructs for exclusions

Exclusions – or event negation – in XChange is handled by the keyword without. This can be used only in combination with the keyword during to specify exclusions during a certain time period, which is either given explicitly as a finite time interval, or as a composite event, whose atomic events denote the start and the end of the time period. The following example excludes a certain event during a finite time interval as a condition in an event query:

```
without {
  event {{ ... }}  }
    during [YYYY-MM-DDThh:mm:ss..YYYY-MM-DDThh:mm:ss]
```

A rule containing such an exclusion only fires if the specified event does not occur during the given time period.

Other ways to denote time periods for exclusions, which are not yet supported by XChange, may be useful. One is the exclusion of certain events until a certain point in time or until the arrival of a certain event message. As an example, consider statements of bank accounts. If a customer of a bank has not yet requested his statements himself, the bank might initiate that they are sent to him. For this purpose they send an event to the system, asking for all customers who have not yet requested their statements. The event query of the corresponding ECA rule could look like this:

```
ON
  without {
    event {{
      type { "statements request by customer" }  }}  }
      until {
       event {{
         type { "send statements" }  }}  }
```

The suggestion therefore is to complement the without construct with a keyword until which allows the exclusions of events until a certain event comes in or until a certain point in time.

The second suggestion is to allow the use of relative time intervals for exclusions. An example for the need of this is given in section 2.2.2 Monitions: composite events. There it is necessary to find books which are lent but not returned within a certain period of time. That a book has been lent can be easily detected with a simple event query looking for the lending event. Additionally, it must be ascertained that there is no corresponding return event for that book. So after the arrival of the lending event the server needs to wait a certain period of time, and then fire the rule if no return event came in in the meantime. This behavior is currently not supported by XChange.[20]

Non-structural conditions in update specifications

The where construct for non-structural conditions at the moment is only allowed in event queries and condition queries. It would be useful also in update specifications, for example to compare the data to be updated with variables bound in the other parts of the rule. An example for the need of this is the following suggestion about sorted insertions.

Sorted insertions

A nice additional language feature for XChange would be the possibility to insert elements into sorted content while maintaining the order. Consider a list of surnames as follows:

```
<surnames>
  <surname>Smith</surname>
  <surname>Vanderzee</surname>
  <surname>Whitfield</surname>
```

[20] The issue is, however, discussed in [2].

```
</surname>
```

Now another surname, for example Thompson, is to be inserted into this alphabetically ordered list. A prerequisite for this are of course comparison operators for strings. Assumed that the XChange comparison operators are capable of comparing strings, the following assertions would hold:

> Smith < Thompson
>
> and
>
> Thompson < Vanderzee

With this, the following rule would allow to insert Thompson into the list in proper order:

```
TRANSACTION
 in { resource { "file:surnames.xml" },
   surnames [[
     surname { var SURNAME1 },
     insert surname { var NEW_SURNAME },
     surname { var SURNAME2 }   ]]   }
   where and { var NEW_SURNAME > var SURNAME1,
               var NEW_SURNAME < var SURNAME2 }
ON
  event {{
    surname { var NEW_SURNAME }   }}
END
```

Note that this solution would not suffice to insert elements into ordered lists that are greater or lower than all other elements of the list, and therefore must be inserted at the beginning or end of the list. For this, additional constructs are necessary to determine the first or last child of a given element, or to compare the contents of all child elements with a given value.

Raising several events with one rule

Currently it is not possible in XChange to raise several events of different structure with the same rule.[21] The action part in an event-raising XChange rule at the moment consists of a single event term, that is, it is only possible to specify and send events of the same structure with one rule.[22] A conjunction of event terms would be useful in event-raising rules. In the ECSS use case for example,

[21] This problem is also brought up by [6], see p. 15.

[22] For the structure of event terms in XChange, see [5], pp. 112f. There is the possibility to populate one event term with data from several records using the keyword all, but this is only handy if an event of a certain structure has to be sent to several recipients. This does not cover the case that events of different structure have to be raised at once.

if the management node is notified of a change in the working group allocation of a member, it has to raise several events and send them to the working group nodes. In case a member leaves one working group and joins another, event term conjunction in raising rules would be useful and could be modeled like this:

```
RAISE
  and {
    xchange:event {{
      xchange:recipient
        { "http://www.church-reformation.net/" },
      type { "delete member" },
      memberId { "105" }  }},
    xchange:event {{
      xchange:recipient
        { "http://www.nation-government.net/" },
      type { "insert member" },
      memberId { "105" }  }}  }
ON
  ...
END
```

Distinction between TRANSACTION and RAISE rules

XChange makes a distinction between rules for the manipulation of data, which begin with the keyword TRANSACTION and event-raising rules, beginning with the keyword RAISE. To lift this restriction and allow the combination – especially conjunction – of event terms raising events and transaction specifications denoting updates seems expedient. Consider the case that the raising of events should be tracked in a log file. This could be accomplished by a conjunction of a term for raising events and one for updating the log file, which might look as follows:

```
TRANSACTION
  and {
    raise xchange:event {{
            xchange:recipient
              { "http://www.church-reformation.net/" },
            type { "delete member" },
            memberId { "105" }  }},
    in { resource { "file:log.xml" },
      entry
        { "sent delete member to church-reformation.net" }  }
ON
  ...
```

```
END
```

Concatenation, simple calculations

In many cases it would be useful if XChange, and Xcerpt, allowed the concatenation of strings and simple calculations with numbers. If for example an incoming event contains the first name and surname of a person in separate XML elements, and these have to be combined for insertion in some persistent data repository (and maybe even complemented by some predefined text), it would be handy if this could be done with XChange like this:

```
1  TRANSACTION
     in { resource { "file:data.xml" },
3      data {{
         insert message { "Dear " . var FIRSTNAME . " " .
5                          var SURNAME . ...  } }}  }
   ON
7    event {{
       surname { var SURNAME },
9      firstName { var FIRSTNAME },
       ...  }}
11 END
```

The dots on line 4 in this case denote a concatenation operator.

A simple example for the need of calculations is a counter:

```
TRANSACTION
  in { resource { "file:data.xml" },
    data {{
      qtyEvents {
        var NUMBER replaceby (var NUMBER + 1) } }}  }
ON
  event {{}}
END
```

This rule would count the number of events received by regular updating an XML counter.

4.5 Concluding remarks

This report has described a use case for the reactive language XChange. The scenario of the use case is the distributed information system of a fictitious scientific research community, the Eighteenth Century Studies Society. The main focus of this work has been a description of the implementation of the use case in XChange. The purpose of the implementation is the demonstration of the functionality of the prototype and the features of the language. Finally, the report lists problems that have been discovered while working with the prototype, and makes suggestions for additional features of future versions of XChange.

A Running the use case in a network

For demonstration purposes, the use case implementation has been made available on several bootable CDs by Michael Eckert. Additionally, he wrote a Perl script to adapt the implementation for running in a distributed environment. Both are described in this section.

CD version of the implementation

To underline the distributed character of the use case during demonstration, it is possible with the CD version to easily run different nodes on different computers. Thus each CD contains a bootable LAMPPIX installation and the data of some of the nodes. The distribution is as follows:

- CD 1: Participant nodes: LMU, CEU, University of Edinburgh
- CD 2: Management node
- CD 3: Working groups 2 and 3

To run the use case, use these CDs to boot three computers connected to each other in a network. The following steps then apply to each individual computer:

- When the LAMPPIX desktop comes up, start a terminal session and switch the login to super user by typing the command su −. The password is lamppix.
- The files of the use case are located in the directory /ramdisk/opt/lampp/htdocs. start.sh starts the XChange servers for all nodes present on the respective CD.
- Next, start a Web browser. (Mozilla Firefox is available at the icon bar of the desktop.) Enter the URL of the Web site of the node you wish to display. The Web sites are situated in the directories: lmu, ceu, edinburgh, management, wg2, and wg3, respectively. The URL for the LMU node for example reads: http://localhost/lmu/
- The log for each node can be displayed by calling log.php in a browser window, for example: http://localhost/lmu/log.php
- Similarly, displayRules.php can be used to view the rules of each node: http://localhost/lmu/displayRules.php

Adapting the use case to other distributed environments

The following adjustments are necessary in order to run the use case in a custom distributed environment:

- In the directory of each node there is a file rules.xchange which contains all the XChange rules employed on this node. The event-raising rules make use of fictitious domain names to specify the recipient of the events, such as history.lmu.de for the LMU participant web site. These have to be replaced by the IP addresses of the computer the respective nodes are supposed to run on. Additionally, these addresses must be complemented by the port number of the XChange server.

- The port number also has to be specified in the file port.inc.php in the subdirectory inc for each node.

The script deploy.pl takes care of these adjustments. In its configuration file, config.txt, for each node and its symbolic URI the real IP addresses and port numbers can be specified. deploy.pl then generates new rules files (named runrules.xchange) that reflect these changes. It also changes the port.inc.php files for all nodes. Finally, it copies the XChange binary to the directories of the nodes and creates a file start.sh, which can be used to start all the necessary XChange servers with the correct parameters (port number and runrules.xchange file to be used.) The individual directories can then be copied to their destination computers.

B ECSS source code

Participant node: CEU

```
# R001 - insert employee
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "CEU website" },
                 event { "insert employee" },
                 reaction { "local update of employee data" }
}  }}  },
    in { resource {"file:staff.xml"},
      staff {{
        insert var EMPLOYEE  }}  }  }
ON
  event {{
    type { "insert employee" },
    var EMPLOYEE -> employee {{}}  }}
END
```

```
# R002 - delete employee
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "CEU website" },
                 event { "delete employee" },
                 reaction { "local update of employee data" }
}  }}  },
    in { resource {"file:staff.xml"},
      staff {{
        delete employee {
                 surname { var SURNAME },
                 firstName { var FIRSTNAME },
                 title { var TITLE },
                 email { var EMAIL },
                 phone { var PHONE },
                 office { var OFFICE },
                 var ECSSWORKINGGROUPS  }  }}  }  }
ON
  event {{
```

```
    type { "delete employee" },
    employee {
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}}  }  }}
END
```

```
# R003 - change employee
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "CEU website" },
                  event { "change employee" },
                  reaction { "local update of employee data" }
}  }}  },
    in { resource {"file:staff.xml"},
      staff {{
        employee {
          surname { var OLD_SURNAME replaceby optional var NEW_SURNAME },
          firstName { var OLD_FIRSTNAME replaceby optional var NEW_FIRSTNAME },
          title { var OLD_TITLE replaceby optional var NEW_TITLE },
          email { var OLD_EMAIL replaceby optional var NEW_EMAIL },
          phone { var OLD_PHONE replaceby optional var NEW_PHONE },
          office { var OLD_OFFICE replaceby optional var NEW_OFFICE },
          ecssWorkingGroups {{}} replaceby var NEW_ECSSWORKINGGROUPS
}  }}  }  }
ON event {{
    type { "change employee" },
    oldRecord {{
      surname { var OLD_SURNAME },
      firstName { var OLD_FIRSTNAME },
      title { var OLD_TITLE },
      email { var OLD_EMAIL },
      phone { var OLD_PHONE },
      office { var OLD_OFFICE }  }},
    newRecord {
      surname { optional var NEW_SURNAME },
      firstName { optional var NEW_FIRSTNAME },
      title { optional var NEW_TITLE },
      email { optional var NEW_EMAIL },
      phone { optional var NEW_PHONE },
```

```
      office { optional var NEW_OFFICE },
      var NEW_ECSSWORKINGGROUPS -> ecssWorkingGroups {{}}  }  }}
END
```

```
# R004 - insert publication
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "CEU website" },
                  event { "insert publication" },
                  reaction { "local update of employee data" }
}  }}  },
    in { resource {"file:publications.xml"},
      publications {{
        insert var PUBLICATION  }}  }  }
ON
  event {{
    type { "insert publication" },
    var PUBLICATION -> publication {{}}  }}
END
```

```
# R005 - propagate insertion of a new employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "insert member" },
    university { "CEU" },
    member {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS  }  }}
ON
  event {{
    type { "insert employee" },
    employee {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
```

```
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}}  }  }}
END
```

```
# The following rule writes a log entry for R005
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
               sender { "CEU website" },
               event { "insert employee" },
               reaction { "raise insert employee on management node" }
}  }}  }
ON
  event {{
    type { "insert employee" }  }}
END
```

```
# R006 - propagate deletion of employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "delete member" },
    university { "CEU" },
    member {
      surname { optional var SURNAME  },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS  }  }}
ON
  event {{
    type { "delete employee" },
    employee {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
```

```
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}} } }}
END
```

```
# The following rule writes a log entry for R006
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
              sender { "CEU website" },
              event { "delete employee" },
              reaction { "raise delete employee on management node" }
} }} }
ON
  event {{
    type { "delete employee" } }}
END
```

```
# R007 - propagate change of employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "change member" },
    university { "CEU" },
    var OLDRECORD,
    var NEWRECORD }}
ON
  event {{
    type { "change employee" },
    var OLDRECORD -> oldRecord {{}},
    var NEWRECORD -> newRecord {{}} }}
END
```

```
# The following rule writes a log entry for R007
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
              sender { "CEU website" },
              event { "change employee" },
```

```
                  reaction { "raise change employee on management node" }
} }} }
ON
  event {{
    type { "change employee" } }}
END
```

```
# R008 - propagate insertion of a new publication to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "insert publication" },
    university { "CEU" },
    var PUBLICATION }}
ON
  event {{
    type { "insert publication" },
    var PUBLICATION -> publication {{}} }}
END
```

```
# The following rule writes a log entry for R008
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                  sender { "CEU website" },
                  event { "insert publication" },
                  reaction { "raise insert publication on management node" }
} }} }
ON
  event {{
    type { "insert publication" } }}
END
```

```
# R009 - lend book
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
```

```
                 sender { "CEU website" },
                 event { "lend book" },
                 reaction { "local update of availability" } }
}} },
    in { resource {"file:books.xml"},
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "available" replaceby "conferred" },
          emailConferred { "none" replaceby var EMAIL } }} }}
} }
ON
  event {{
    type { "lend book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
END
```

```
# R010 - prebook book
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "CEU website" },
                 event { "prebook book" },
                 reaction { "local update of availability" } }
}} },
    in { resource {"file:books.xml"},
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "conferred" replaceby "prebooked" },
          emailPrebooked { "none" replaceby var EMAIL } }} }}
} }
ON
  event {{
    type { "prebook book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
END
```

```
# R011 - raise request book on management node
RAISE
```

```
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    requestingLibrary { "CEU" },
    type { "request2 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
ON
  event {{
    type { "request1 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
END
```

```
# The following rule writes a log entry for R011
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                 sender { "CEU website" },
                 event { "request book" },
                 reaction { "raise locate book on management node" }
} }} }
ON
  event {{
    type { "request1 book" } }}
END
```

```
# R012 - update local data: book sent
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                   sender { "management node" },
                   event { "initiate book delivery" },
                   reaction { "local update of availability" } }
}} },
    in { resource { "file:books.xml" },
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "available" replaceby "not available" } }}
}} } }
ON
```

```
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER }  }}
END
```

```
# R013 - update local data: book received
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { var DELIVERINGLIBRARY },
                  event { "book transferred" },
                  reaction { "local update of availability" }  }
}}  },
    in { resource { "file:books.xml" },
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "not available" replaceby "available" }  }}
}}  }  }
ON
  event {{
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { var DELIVERINGLIBRARY }  }}
END
```

```
# R014 - transfer book: inform management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY },
    deliveringLibrary { "CEU" }  }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
END
```

```
# The following rule writes a log entry for R014
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
              sender { "management node" },
              event { "initiate book delivery" },
              reaction { "raise book location update on management node" }
} }} }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY } }}
END
```

```
# R015 - transfer book: inform requesting library (LMU)
RAISE
  event {{
    xchange:recipient { "http://geschichte.lmu.de/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { "CEU" } }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "LMU" } }}
END
```

```
# The following rule writes a log entry for R015
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
              sender { "management node" },
              event { "initiate book delivery" },
              reaction { "raise book availability update on LMU node" }
} }} }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
```

```
    requestingLibrary { "LMU" }  }}
END
```

```
# R016 - transfer book: inform requesting library (Edinburgh)
RAISE
  event {{
    xchange:recipient { "http://history.ed.ac.uk/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { "CEU" }  }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "Edinburgh" }  }}
END
```

```
# The following rule writes a log entry for R016
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                 sender { "management node" },
                 event { "initiate book delivery" },
                 reaction { "raise book availability update
                               on Edinburgh University node" }  }
}}  }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "Edinburgh" }  }}
END
```

Participant node: Edinburgh

```
# R001 - insert employee
TRANSACTION
  and {
```

```
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "Edinburgh University website" },
                 event { "insert employee" },
                 reaction { "local update of employee data" }
} }} },
    in { resource {"file:staff.xml"},
      staff {{
        insert var EMPLOYEE }} } }
ON
  event {{
    type { "insert employee" },
    var EMPLOYEE -> employee {{}} }}
END
```

```
# R002 - delete employee
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "Edinburgh University website" },
                 event { "delete employee" },
                 reaction { "local update of employee data" }
} }} },
    in { resource {"file:staff.xml"},
      staff {{
        delete employee {
                 surname { var SURNAME },
                 firstName { var FIRSTNAME },
                 title { var TITLE },
                 email { var EMAIL },
                 phone { var PHONE },
                 office { var OFFICE },
                 var ECSSWORKINGGROUPS } }} } }
ON
  event {{
    type { "delete employee" },
    employee {
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}} } }}
```

```
END
```

```
# R003 - change employee
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "Edinburgh University website" },
                 event { "change employee" },
                 reaction { "local update of employee data" }
} }} },
    in { resource {"file:staff.xml"},
      staff {{
        employee {
          surname { var OLD_SURNAME replaceby optional var NEW_SURNAME },
          firstName { var OLD_FIRSTNAME replaceby optional var NEW_FIRSTNAME },
          title { var OLD_TITLE replaceby optional var NEW_TITLE },
          email { var OLD_EMAIL replaceby optional var NEW_EMAIL },
          phone { var OLD_PHONE replaceby optional var NEW_PHONE },
          office { var OLD_OFFICE replaceby optional var NEW_OFFICE },
          ecssWorkingGroups {{}} replaceby var NEW_ECSSWORKINGGROUPS
} }} } }
ON event {{
    type { "change employee" },
    oldRecord {{
      surname { var OLD_SURNAME },
      firstName { var OLD_FIRSTNAME },
      title { var OLD_TITLE },
      email { var OLD_EMAIL },
      phone { var OLD_PHONE },
      office { var OLD_OFFICE } }},
    newRecord {
      surname { optional var NEW_SURNAME },
      firstName { optional var NEW_FIRSTNAME },
      title { optional var NEW_TITLE },
      email { optional var NEW_EMAIL },
      phone { optional var NEW_PHONE },
      office { optional var NEW_OFFICE },
      var NEW_ECSSWORKINGGROUPS -> ecssWorkingGroups {{}} } }}
END
```

```
# R004 - insert publication
TRANSACTION
```

```
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "Edinburgh University website" },
                  event { "insert publication" },
                  reaction { "local update of employee data" }
} }} },
    in { resource {"file:publications.xml"},
      publications {{
        insert var PUBLICATION }} } }
ON
  event {{
    type { "insert publication" },
    var PUBLICATION -> publication {{}} }}
END
```

```
# R005 - propagate insertion of a new employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "insert member" },
    university { "Edinburgh" },
    member {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS } }}
ON
  event {{
    type { "insert employee" },
    employee {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}} } }}
END
```

```
# The following rule writes a log entry for R005
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
               sender { "Edinburgh University website" },
               event { "insert employee" },
               reaction { "raise insert employee on management node" }
} }} }
ON
  event {{
    type { "insert employee" } }}
END
```

```
# R006 - propagate deletion of employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "delete member" },
    university { "Edinburgh" },
    member {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS } }}
ON
  event {{
    type { "delete employee" },
    employee {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}} } }}
END
```

```
# The following rule writes a log entry for R006
TRANSACTION
```

```
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                sender { "Edinburgh University website" },
                event { "delete employee" },
                reaction { "raise delete employee on management node" }
} }} }
ON
  event {{
    type { "delete employee" } }}
END
```

```
# R007 - propagate change of employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "change member" },
    university { "Edinburgh" },
    var OLDRECORD,
    var NEWRECORD }}
ON
  event {{
    type { "change employee" },
    var OLDRECORD -> oldRecord {{}},
    var NEWRECORD -> newRecord {{}} }}
END
```

```
# The following rule writes a log entry for R007
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                sender { "Edinburgh University website" },
                event { "change employee" },
                reaction { "raise change employee on management node" }
} }} }
ON
  event {{
    type { "change employee" } }}
END
```

```
# R008 - propagate insertion of a new publication to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "insert publication" },
    university { "Edinburgh" },
    var PUBLICATION  }}
ON
  event {{
    type { "insert publication" },
    var PUBLICATION -> publication {{}}  }}
END
```

```
# The following rule writes a log entry for R008
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                sender { "Edinburgh University website" },
                event { "insert publication" },
                reaction { "raise insert publication on management node" }
}  }}  }
ON
  event {{
    type { "insert publication" }  }}
END
```

```
# R009 - lend book
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "Edinburgh University website" },
                  event { "lend book" },
                  reaction { "local update of availability" }  }
}}  },
    in { resource {"file:books.xml"},
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "available" replaceby "conferred" },
```

```
          emailConferred { "none" replaceby var EMAIL }  }}  }}
}  }
ON event {{
    type { "lend book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL }  }}
END
```

```
# R010 - prebook book
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "Edinburgh University website" },
                 event { "prebook book" },
                 reaction { "local update of availability" }  }
}}  },
    in { resource {"file:books.xml"},
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "conferred" replaceby "prebooked" },
          emailPrebooked { "none" replaceby var EMAIL }  }}  }}
}  }
ON event {{
    type { "prebook book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL }  }}
END
```

```
# R011 - raise request book on management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    requestingLibrary { "Edinburgh" },
    type { "request2 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL }  }}
ON
  event {{
    type { "request1 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL }  }}
```

```
END
```

```
# The following rule writes a log entry for R011
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                sender { "Edinburgh University website" },
                event { "request book" },
                reaction { "raise locate book on management node" }
}  }}  }
ON event {{
    type { "request1 book" }  }}
END
```

```
# R012 - update local data: book sent
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management node" },
                  event { "initiate book delivery" },
                  reaction { "local update of availability" }  }
}}  },
    in { resource { "file:books.xml" },
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "available" replaceby "not available" }  }}
}}  }  }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER }  }}
END
```

```
# R013 - update local data: book received
TRANSACTION
  and {
```

```
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { var DELIVERINGLIBRARY },
                  event { "book transferred" },
                  reaction { "local update of availability" } }
}}  },
    in { resource { "file:books.xml" },
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "not available" replaceby "available" }  }}
}}  }  }
ON
  event {{
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { var DELIVERINGLIBRARY }  }}
END
```

```
# R014 - transfer book: inform management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY },
    deliveringLibrary { "Edinburgh" }  }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
END
```

```
# The following rule writes a log entry for R014
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
              sender { "management node" },
              event { "initiate book delivery" },
              reaction { "raise book location update on management node" }
}  }}  }
```

```
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
END
```

```
# R015 - transfer book: inform requesting library (CEU)
RAISE
  event {{
    xchange:recipient { "http://history.ceu.org/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { "Edinburgh University" }  }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "CEU" }  }}
END
```

```
# The following rule writes a log entry for R015
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                  sender { "management node" },
                  event { "initiate book delivery" },
                  reaction { "raise book availability update on CEU node" }
}  }}  }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "CEU" }  }}
END
```

```
# R016 - transfer book: inform requesting library (LMU)
RAISE
  event {{
```

```
    xchange:recipient { "http://geschichte.lmu.de/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { "Edinburgh University" } }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "LMU" } }}
END
```

```
# The following rule writes a log entry for R016
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                sender { "management node" },
                event { "initiate book delivery" },
                reaction { "raise book availability update on LMU node" }
} }} }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "LMU" } }}
END
```

Participant node: LMU

```
# R001 - insert employee
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "LMU website" },
                  event { "insert employee" },
                  reaction { "local update of employee data" }
} }} },
    in { resource {"file:staff.xml"},
      staff {{
        insert var EMPLOYEE }} } }
```

```
ON
  event {{
    type { "insert employee" },
    var EMPLOYEE -> employee {{}}  }}
END
```

```
# R002 - delete employee
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "LMU website" },
                  event { "delete employee" },
                  reaction { "local update of employee data" }
}  }}  },
    in { resource {"file:staff.xml"},
      staff {{
        delete employee {
                  surname { var SURNAME },
                  firstName { var FIRSTNAME },
                  title { var TITLE },
                  email { var EMAIL },
                  phone { var PHONE },
                  office { var OFFICE },
                  var ECSSWORKINGGROUPS  }  }}  }  }
ON
  event {{
    type { "delete employee" },
    employee {
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}}  }  }}
END
```

```
# R003 - change employee
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
```

```
        insert entry {
                sender { "LMU website" },
                event { "change employee" },
                reaction { "local update of employee data" }
} }} },
    in { resource {"file:staff.xml"},
      staff {{
        employee {
          surname { var OLD_SURNAME replaceby optional var NEW_SURNAME },
          firstName { var OLD_FIRSTNAME replaceby optional var NEW_FIRSTNAME },
          title { var OLD_TITLE replaceby optional var NEW_TITLE },
          email { var OLD_EMAIL replaceby optional var NEW_EMAIL },
          phone { var OLD_PHONE replaceby optional var NEW_PHONE },
          office { var OLD_OFFICE replaceby optional var NEW_OFFICE },
          ecssWorkingGroups {{}} replaceby var NEW_ECSSWORKINGGROUPS
} }} } }
ON event {{
    type { "change employee" },
    oldRecord {{
      surname { var OLD_SURNAME },
      firstName { var OLD_FIRSTNAME },
      title { var OLD_TITLE },
      email { var OLD_EMAIL },
      phone { var OLD_PHONE },
      office { var OLD_OFFICE } }},
    newRecord {
      surname { optional var NEW_SURNAME },
      firstName { optional var NEW_FIRSTNAME },
      title { optional var NEW_TITLE },
      email { optional var NEW_EMAIL },
      phone { optional var NEW_PHONE },
      office { optional var NEW_OFFICE },
      var NEW_ECSSWORKINGGROUPS -> ecssWorkingGroups {{}} } }}
END
```

```
# R004 - insert publication
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                sender { "LMU website" },
                event { "insert publication" },
                reaction { "local update of employee data" }
} }} },
    in { resource {"file:publications.xml"},
      publications {{
```

```
        insert var PUBLICATION }} } }
ON
  event {{
    type { "insert publication" },
    var PUBLICATION -> publication {{}} }}
END
```

```
# R005 - propagate insertion of a new employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "insert member" },
    university { "LMU" },
    member {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS } }}
ON
  event {{
    type { "insert employee" },
    employee {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}} } }}
END
```

```
# The following rule writes a log entry for R005
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                sender { "LMU website" },
                event { "insert employee" },
                reaction { "raise insert employee on management node" }
} }} }
ON
```

```
  event {{
    type { "insert employee" }  }}
END
```

```
# R006 - propagate deletion of employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "delete member" },
    university { "LMU" },
    member {
      surname { optional var SURNAME  },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS  }  }}
ON
  event {{
    type { "delete employee" },
    employee {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}}  }  }}
END
```

```
# The following rule writes a log entry for R006
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                 sender { "LMU website" },
                 event { "delete employee" },
                 reaction { "raise delete employee on management node" }
}  }}  }
ON
  event {{
    type { "delete employee" }  }}
END
```

```
# R007 − propagate change of employee to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "change member" },
    university { "LMU" },
    var OLDRECORD,
    var NEWRECORD }}
ON
  event {{
    type { "change employee" },
    var OLDRECORD −> oldRecord {{}},
    var NEWRECORD −> newRecord {{}} }}
END
```

```
# The following rule writes a log entry for R007
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                 sender { "LMU website" },
                 event { "change employee" },
                 reaction { "raise change employee on management node" }
} }} }
ON
  event {{
    type { "change employee" } }}
END
```

```
# R008 − propagate insertion of a new publication to management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "insert publication" },
    university { "LMU" },
    var PUBLICATION }}
ON
  event {{
    type { "insert publication" },
```

```
    var PUBLICATION -> publication {{}} }}
END
```

```
# The following rule writes a log entry for R008
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
              sender { "LMU website" },
              event { "insert publication" },
              reaction { "raise insert publication on management node" }
} }} }
ON
  event {{
    type { "insert publication" } }}
END
```

```
# R009 - lend book
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                sender { "LMU website" },
                event { "lend book" },
                reaction { "local update of availability" } }
}} },
    in { resource {"file:books.xml"},
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "available" replaceby "conferred" },
          emailConferred { "none" replaceby var EMAIL } }} }}
} }
ON event {{
    type { "lend book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
END
```

```
# R010 - prebook book
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "LMU website" },
                 event { "prebook book" },
                 reaction { "local update of availability" } }
}} },
    in { resource {"file:books.xml"},
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "conferred" replaceby "prebooked" },
          emailPrebooked { "none" replaceby var EMAIL } }} }}
} }
ON event {{
    type { "prebook book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
END
```

```
# R011 - raise request book on management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    requestingLibrary { "LMU" },
    type { "request2 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
ON event {{
    type { "request1 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL } }}
END
```

```
# The following rule writes a log entry for R011
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
               sender { "LMU website" },
               event { "request book" },
```

```
                reaction { "raise locate book on management node" }
} }} }
ON
  event {{
    type { "request1 book" }  }}
END
```

```
# R012 - update local data: book sent
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "management node" },
                 event { "initiate book delivery" },
                 reaction { "local update of availability" }  }
}}  },
    in { resource { "file:books.xml" },
      books {{
        book {{
          callNumber { var CALLNUMBER },
          state { "available" replaceby "not available" }  }}
}}  }  }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER }  }}
END
```

```
# R013 - update local data: book received
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { var DELIVERINGLIBRARY },
                 event { "book transferred" },
                 reaction { "local update of availability" }  }
}}  },
    in { resource { "file:books.xml" },
      books {{
        book {{
          callNumber { var CALLNUMBER },
```

```
          state { "not available" replaceby "available" }  }}
}}  }  }
ON
  event {{
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { var DELIVERINGLIBRARY }  }}
END
```

```
# R014 - transfer book: inform management node
RAISE
  event {{
    xchange:recipient { "http://www.ecss.org/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY },
    deliveringLibrary { "LMU" }  }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
END
```

```
# The following rule writes a log entry for R014
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                sender { "management node" },
                event { "initiate book delivery" },
                reaction { "raise book location update on management node" }
}  }}  }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
END
```

```
# R015 - transfer book: inform requesting library (CEU)
RAISE
  event {{
    xchange:recipient { "http://history.ceu.org/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { "LMU" } }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "CEU" } }}
END
```

```
# The following rule writes a log entry for R015
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                 sender { "management node" },
                 event { "initiate book delivery" },
                 reaction { "raise book availability update on CEU node" }
} }} }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "CEU" } }}
END
```

```
# R016 - transfer book: inform requesting library (Edinburgh)
RAISE
  event {{
    xchange:recipient { "http://history.ed.ac.uk/" },
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    deliveringLibrary { "LMU" } }}
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "Edinburgh" } }}
END
```

```
# The following rule writes a log entry for R016
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                sender { "management node" },
                event { "initiate book delivery" },
                reaction { "raise book availability update on Edinburgh University node'
} }} }
ON
  event {{
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    requestingLibrary { "Edinburgh" }  }}
END
```

Management node

```
# R001 - insert member
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { var UNIVERSITY },
                  event { "insert member" },
                  reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
      members {{
        insert var MEMBER  }}  }  }
ON
  event {{
    type { "insert member" },
    university { var UNIVERSITY },
    var MEMBER -> member {{}}  }}
END
```

```
# R002 - delete member
TRANSACTION
  and {
    in { resource {"file:log.xml"},
```

```
      log {{
        insert entry {
                 sender { var UNIVERSITY },
                 event { "delete member" },
                 reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
      members {{
        delete member {
                 surname { var SURNAME },
                 firstName { var FIRSTNAME },
                 title { var TITLE },
                 email { var EMAIL },
                 phone { var PHONE },
                 office { var OFFICE },
                 var ECSSWORKINGGROUPS  }  }}  }  }
ON
  event {{
    type { "delete member" },
    university { var UNIVERSITY },
    member {
      surname { var SURNAME  },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE },
      var ECSSWORKINGGROUPS -> ecssWorkingGroups {{}}  }  }}
END
```

```
# R003 - change member
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { var UNIVERSITY },
                 event { "change member" },
                 reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
      members {{
        member {
          surname { var OLD_SURNAME replaceby optional var NEW_SURNAME },
          firstName { var OLD_FIRSTNAME replaceby optional var NEW_FIRSTNAME },
          title { var OLD_TITLE replaceby optional var NEW_TITLE },
          email { var OLD_EMAIL replaceby optional var NEW_EMAIL },
```

```
          phone { var OLD_PHONE replaceby optional var NEW_PHONE },
          office { var OLD_OFFICE replaceby optional var NEW_OFFICE },
          ecssWorkingGroups {{}} replaceby var NEW_ECSSWORKINGGROUPS
} }} } }
ON event {{
    type { "change member" },
    university { var UNIVERSITY },
    oldRecord {{
      surname { var OLD_SURNAME },
      firstName { var OLD_FIRSTNAME },
      title { var OLD_TITLE },
      email { var OLD_EMAIL },
      phone { var OLD_PHONE },
      office { var OLD_OFFICE } }},
    newRecord {
      surname { optional var NEW_SURNAME },
      firstName { optional var NEW_FIRSTNAME },
      title { optional var NEW_TITLE },
      email { optional var NEW_EMAIL },
      phone { optional var NEW_PHONE },
      office { optional var NEW_OFFICE },
      var NEW_ECSSWORKINGGROUPS -> ecssWorkingGroups {{}} } }}
END
```

```
# R004 - insert publication
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { var UNIVERSITY },
                  event { "insert publication" },
                  reaction { "local update of member data" } }
}} },
    in { resource {"file:publications.xml"},
      publications {{
        insert var PUBLICATION }} } }
ON
  event {{
    type { "insert publication" },
    university { var UNIVERSITY },
    var PUBLICATION -> publication {{}} }}
END
```

```
# R005 − propagate insertion of new member to working group node
(working group 3)
RAISE
  event {{
    xchange:recipient { "http://www.church−reformation.net/" },
    type { "insert member" },
    var MEMBER }}
ON
  event {{
    type { "insert member" },
    var MEMBER −> member {{
                      ecssWorkingGroups {{
                        ecssWorkingGroup { "Church and Reformation" }
}} }} }}
END
```

```
# The following rule writes a log entry for R005
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
               sender { var UNIVERSITY },
               event { "insert member" },
               reaction { "raise insert member on working
                           group node: church and reformation" }
} }} }
ON
  event {{
    type { "insert member" },
    university { var UNIVERSITY },
    member {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Church and Reformation" } }} }}
}}
END
```

```
# R006 − propagate deletion of member to working group node (working
group 3)
RAISE
  event {{
    xchange:recipient { "http://www.church−reformation.net/" },
    type { "delete member" },
    var MEMBER }}
```

```
ON
  event {{
    type { ”delete member” },
    var MEMBER −> member {{
                     ecssWorkingGroups {{
                       ecssWorkingGroup { ”Church and Reformation” }
}}  }}  }}
END
```

```
# The following rule writes a log entry for R006
TRANSACTION
  in { resource { ”file:log.xml” },
    log {{
      insert entry {
               sender { var UNIVERSITY },
               event { ”delete member” },
               reaction { ”raise delete member on working
                           group node: church and reformation” }
}  }}  }
ON
  event {{
    type { ”delete member” },
    university { var UNIVERSITY },
    member {{
      ecssWorkingGroups {{
        ecssWorkingGroup { ”Church and Reformation” }  }}  }}
}}
END
```

```
# R007 − propagate change of member data to working group node
(working group 3)
RAISE
  event {{
      xchange:recipient { ”http://www.church−reformation.net/” },
      type { ”change member” },
      var OLDRECORD,
      var NEWRECORD  }}
ON event {{
    type { ”change member” },
    var OLDRECORD −> oldRecord {{
                       ecssWorkingGroups {{
                         ecssWorkingGroup { ”Church and Reformation” }
}}  }},
    var NEWRECORD −> newRecord {{
```

```
                          ecssWorkingGroups {{
                            ecssWorkingGroup { "Church and Reformation" }
}}  }}  }}
END
```

```
# The following rule writes a log entry for R007
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
               sender { var UNIVERSITY },
               event { "change member" },
               reaction { "raise change member on working
                           group node: church and reformation" }
}  }}  }
ON
  event {{
    type { "change member" },
    university { var UNIVERSITY },
    oldRecord {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Church and Reformation" }  }}  }},
    newRecord {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Church and Reformation" }  }}  }}
}}
END
```

```
# R008 - propagate insertion of new member due to change of working
group allocation to working group node (wg3)
RAISE event {{
    xchange:recipient { "http://www.church-reformation.net/" },
    type { "insert member" },
    member {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS  }  }}
ON
  event {{
      type { "change member" },
```

```
      oldRecord {{
        ecssWorkingGroups {{
          without ecssWorkingGroup { "Church and Reformation" }
}}  }},
      newRecord {
        surname { optional var SURNAME },
        firstName { optional var FIRSTNAME },
        title { optional var TITLE },
        email { optional var EMAIL },
        phone { optional var PHONE },
        office { optional var OFFICE },
        var ECSSWORKINGGROUPS -> ecssWorkingGroups {{
                                         ecssWorkingGroup { "Church and Reformation" }
}}  }  }}
END
```

```
# The following rule writes a log entry for R008
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                sender { var UNIVERSITY },
                event { "change member" },
                reaction { "raise insert member on working
                              group node: church and reformation" }
}  }}  }
ON
  event {{
      type { "change member" },
      university { var UNIVERSITY },
      oldRecord {{
        ecssWorkingGroups {{
          without ecssWorkingGroup { "Church and Reformation" }
}}  }},
      newRecord {{
        ecssWorkingGroups {{
          ecssWorkingGroup { "Church and Reformation" }  }}  }}
}}
END
```

```
# R009 - propagate deletion of new member due to change of working
group allocation to working group node (wg3)
RAISE
  event {{
```

```
    xchange:recipient { "http://www.church-reformation.net/" },
    type { "delete member" },
    member {
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE }  }  }}
ON
  event {{
    type { "change member" },
    oldRecord {
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE },
      ecssWorkingGroups {{
        ecssWorkingGroup { "Church and Reformation" }  }}  },
    newRecord {{
      ecssWorkingGroups {{
        without ecssWorkingGroup { "Church and Reformation" }
}}  }}  }}
END
```

```
# The following rule writes a log entry for R009
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                 sender { var UNIVERSITY },
                 event { "change member" },
                 reaction { "raise delete member on working
                        group node: church and reformation" }
}  }}  }
ON
  event {{
    type { "change member" },
    university { var UNIVERSITY },
    oldRecord {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Church and Reformation" }  }}  }},
    newRecord {{
      ecssWorkingGroups {{
```

```
        without ecssWorkingGroup { "Church and Reformation" }
}}  }}  }}
END
```

```
# R010 - register for workshop
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "management website" },
                 event { "register for workshop" },
                 reaction { "local update of workshop list" }
}  }}  },
    in { resource {"file:registrations.xml"},
      registrations {{
        insert registration { var REGISTRANT }  }}  }  }
ON
  event {{
    newRegistrant { var REGISTRANT }  }} before "2006-08-15T12:00:00.00"
END
```

```
# R011 - propose panel moderator
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "management website" },
                 event { "propose panel moderator" },
                 reaction { "local update of moderator list" }
}  }}  },
    in { resource {"file:moderators.xml"},
      moderators {{
        insert moderator {
                 name { var MODERATOR },
                 confirmed { "no" }  }  }}  }  }
ON
  event {{
    moderatorProposal { var MODERATOR }  }}
END
```

```
# R012 − confirm panel moderator
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management website" },
                  event { "confirm panel moderator" },
                  reaction { "local update of moderator list" }
} }} },
    in { resource {"file:moderators.xml"},
      moderators {{
        moderator {{
          name { var MODERATOR },
          confirmed { "no" replaceby "yes" } }} }} } }
ON
  andthen [
    event {{
        moderatorProposal { var MODERATOR } }},
    event {{
        moderatorConfirmation { var MODERATOR } }} ]
    within "TimeDiff {tdYear = 0, tdMonth = 0, tdDay = 1, tdHour = 0,
                        tdMin = 0, tdSec = 0, tdPicosec = 0}"
END
```

```
# R013 − note possible paper combinations on receipt of new paper
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management website" },
                  event { "new paper" },
                  reaction { "local update of paper list" } }
}} },
    in { resource {"file:paperCombinations.xml"},
      combinations {{
        insert combination {
                  paper1 {
                    lecturer { var LECTURER1 },
                    title { var TITLE1 } },
                  paper2 {
                    lecturer { var LECTURER2 },
                    title { var TITLE2 } },
                  workingGroup { optional var WORKINGGROUP },
```

```
                  category { optional var CATEGORY }  }  }}  }
}
ON
  or {
    and {
      event {{
        newPaper {
          lecturer { var LECTURER1 },
          title { var TITLE1 },
          workingGroup { var WORKINGGROUP },
          category { var CATEGORY1 }  }  }},
      event {{
        newPaper {
          lecturer { var LECTURER2 },
          title { var TITLE2 },
          workingGroup { var WORKINGGROUP },
          category { var CATEGORY2 }  }  }}  },
    and {
      event {{
        newPaper {
          lecturer { var LECTURER1 },
          title { var TITLE1 },
          workingGroup { var WORKINGGROUP1 },
          category { var CATEGORY }  }  }},
      event {{
        newPaper {
          lecturer { var LECTURER2 },
          title { var TITLE2 },
          workingGroup { var WORKINGGROUP2 },
          category { var CATEGORY }  }  }}  }  }
END
```

```
# R014 - propagate insertion of new member to working group node
(working group 2)
RAISE
  event {{
    xchange:recipient { "http://www.nation-government.net/" },
    type { "insert member" },
    var MEMBER  }}
ON
  event {{
    type { "insert member" },
    var MEMBER -> member {{
                   ecssWorkingGroups {{
                     ecssWorkingGroup { "Nation and Government" }
}}  }}  }}
END
```

```
# The following rule writes a log entry for R014
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                 sender { var UNIVERSITY },
                 event { "insert member" },
                 reaction { "raise insert member on working
                              group node: nation and government" },
} }} }
ON
  event {{
    type { "insert member" },
    university { var UNIVERSITY },
    member {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Nation and Government" } }} }} }}
END
```

```
# R015 - propagate deletion of member to working group node (working
group 2)
RAISE
  event {{
    xchange:recipient { "http://www.nation-government.net/" },
    type { "delete member" },
    var MEMBER }}
ON
  event {{
    type { "delete member" },
    var MEMBER -> member {{
                    ecssWorkingGroups {{
                      ecssWorkingGroup { "Nation and Government" }
}} }} }}
END
```

```
# The following rule writes a log entry for R015
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
```

```
    insert entry {
            sender { var UNIVERSITY },
            event { "delete member" },
            reaction { "raise delete member on working
                    group node: nation and government" }
} }} }
ON
  event {{
    type { "delete member" },
    university { var UNIVERSITY },
    member {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Nation and Government" } }} }} }}
END
```

```
# R016 - propagate change of member data to working group node
(working group 2)
RAISE
  event {{
      xchange:recipient { "http://www.nation-government.net/" },
      type { "change member" },
      var OLDRECORD,
      var NEWRECORD }}
ON event {{
    type { "change member" },
    var OLDRECORD -> oldRecord {{
                    ecssWorkingGroups {{
                      ecssWorkingGroup { "Nation and Government" }
}} }},
    var NEWRECORD -> newRecord {{
                    ecssWorkingGroups {{
                      ecssWorkingGroup { "Nation and Government" }
}} }} }}
END
```

```
# The following rule writes a log entry for R016
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
            sender { var UNIVERSITY },
            event { "change member" },
            reaction { "raise change member on working
```

```
                            group node: nation and government" }
} }} }
ON
  event {{
    type { "change member" },
    university { var UNIVERSITY },
    oldRecord {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Nation and Government" } }} }},
    newRecord {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Nation and Government" } }} }} }}
END
```

```
# R017 - propagate insertion of new member due to change of working
group allocation to working group node (wg2)
RAISE event {{
    xchange:recipient { "http://www.nation-government.net/" },
    type { "insert member" },
    member {
      surname { optional var SURNAME },
      firstName { optional var FIRSTNAME },
      title { optional var TITLE },
      email { optional var EMAIL },
      phone { optional var PHONE },
      office { optional var OFFICE },
      var ECSSWORKINGGROUPS } }}
ON
  event {{
      type { "change member" },
      oldRecord {{
        ecssWorkingGroups {{
          without ecssWorkingGroup { "Nation and Government" }
}} }},
      newRecord {
        surname { optional var SURNAME },
        firstName { optional var FIRSTNAME },
        title { optional var TITLE },
        email { optional var EMAIL },
        phone { optional var PHONE },
        office { optional var OFFICE },
        var ECSSWORKINGGROUPS -> ecssWorkingGroups {{
                                  ecssWorkingGroup { "Nation and Government" }
}} } }}
END
```

```
# The following rule writes a log entry for R017
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
               sender { var UNIVERSITY },
               event { "change member" },
               reaction { "raise insert member on working
                           group node: nation and government" }
}  }}  }
ON
  event {{
      type { "change member" },
      university { var UNIVERSITY },
      oldRecord {{
        ecssWorkingGroups {{
          without ecssWorkingGroup { "Nation and Government" }
}}  }},
      newRecord {{
        ecssWorkingGroups {{
          ecssWorkingGroup { "Nation and Government" }  }}  }}
}}
END
```

```
# R018 - propagate deletion of new member due to change of working
group allocation to working group node (wg2)
RAISE
  event {{
    xchange:recipient { "http://www.nation-government.net/" },
    type { "delete member" },
    member {
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE }  }  }}
ON
  event {{
    type { "change member" },
    oldRecord {
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
```

```
      office { var OFFICE },
      ecssWorkingGroups {{
        ecssWorkingGroup { "Nation and Government" }  }}  },
    newRecord {{
      ecssWorkingGroups {{
        without ecssWorkingGroup { "Nation and Government" }  }}
}}  }}
END
```

```
# The following rule writes a log entry for R018
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                sender { var UNIVERSITY },
                event { "change member" },
                reaction { "raise delete member on working
                            group node: nation and government" }
}  }}  }
ON
  event {{
    type { "change member" },
    university { var UNIVERSITY },
    oldRecord {{
      ecssWorkingGroups {{
        ecssWorkingGroup { "Nation and Government" }  }}  }},
    newRecord {{
      ecssWorkingGroups {{
        without ecssWorkingGroup { "Nation and Government" }  }}
}}  }}
END
```

```
# R019 - propagate newsletter info to working group node (wg2)
RAISE
  event {{
    xchange:recipient { "http://www.nation-government.net/" },
    type { var SUBTYPE },
    var MESSAGE  }}
ON
  event {{
    type { "newsletter" },
    subtype { var SUBTYPE },
    workingGroup { "working group: nation and government" },
    var MESSAGE -> message {{}}  }}
```

```
END
```

```
# The following rule writes a log entry for R019
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                 sender { "mail client" },
                 event { "incoming newsletter" },
                 reaction { "raise insert news on wg2 node" }  }
}}  }
ON
  event {{
    type { "newsletter" },
    subtype { var SUBTYPE },
    workingGroup { "working group: nation and government" }  }}
END
```

```
# R020 - propagate newsletter info to working group node (wg3)
RAISE
  event {{
    xchange:recipient { "http://www.church-reformation.net/" },
    type { var SUBTYPE },
    var MESSAGE  }}
ON
  event {{
    type { "newsletter" },
    subtype { var SUBTYPE },
    workingGroup { "working group: church and reformation" },
    var MESSAGE -> message {{}}  }}
END
```

```
# The following rule writes a log entry for R020
TRANSACTION
  in { resource { "file:log.xml" },
    log {{
      insert entry {
                 sender { "mail client" },
                 event { "incoming newsletter" },
```

```
                reaction { "raise insert news on wg3 node" }  }
}}  }
ON
  event {{
    type { "newsletter" },
    subtype { var SUBTYPE },
    workingGroup { "working group: church and reformation" },
}}
END
```

```
# R021 − send request for book to participant (LMU)
RAISE
  event {{
    xchange:recipient { "http://geschichte.lmu.de/" },
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
ON
  event {{
    requestingLibrary { var REQUESTINGLIBRARY },
    type { "request2 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL }  }}
FROM
  in { resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        library { "LMU" }  }}    }}  }
END
```

```
# The following rule writes a log entry for R021
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
                sender { var REQUESTINGLIBRARY },
                event { "locate book" },
                reaction { "raise request book on participant's node (LMU)" }
}  }}  }
ON
  event {{
    requestingLibrary { var REQUESTINGLIBRARY },
```

```
    type { "request2 book" },
    callNumber { var CALLNUMBER }  }}
FROM
  in { resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        library { "LMU" }  }}    }}  }
END
```

```
# R022 - send request for book to participant (CEU)
RAISE
  event {{
    xchange:recipient { "http://history.ceu.org/" },
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
ON
  event {{
    requestingLibrary { var REQUESTINGLIBRARY },
    type { "request2 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL }  }}
FROM
  in { resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        library { "CEU" }  }}    }}  }
END
```

```
# The following rule writes a log entry for R022
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
              sender { var REQUESTINGLIBRARY },
              event { "locate book" },
              reaction { "raise request book on participant's node (CEU)" }
}  }}  }
ON
  event {{
    requestingLibrary { var REQUESTINGLIBRARY },
```

```
    type { "request2 book" },
    callNumber { var CALLNUMBER }  }}
FROM
  in { resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        library { "CEU" }  }}    }}  }
END
```

```
# R023 - send request for book to participant (Edinburgh)
RAISE
  event {{
    xchange:recipient { "http://history.ed.ac.uk/" },
    type { "request3 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL },
    requestingLibrary { var REQUESTINGLIBRARY }  }}
ON
  event {{
    requestingLibrary { var REQUESTINGLIBRARY },
    type { "request2 book" },
    callNumber { var CALLNUMBER },
    email { var EMAIL }  }}
FROM
  in { resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        library { "Edinburgh" }  }}    }}  }
END
```

```
# The following rule writes a log entry for R023
TRANSACTION
  in { resource {"file:log.xml"},
    log {{
      insert entry {
              sender { var REQUESTINGLIBRARY },
              event { "locate book" },
              reaction { "raise request book on participant's node (Edinburgh" }
}  }}  }
ON
  event {{
    requestingLibrary { var REQUESTINGLIBRARY },
```

```
    type { "request2 book" },
    callNumber { var CALLNUMBER }  }}
FROM
  in { resource { "file:books.xml" },
    books {{
      book {{
        callNumber { var CALLNUMBER },
        library { "Edinburgh" }  }}    }}  }
END
```

```
# R024 - update books.xml
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { var DELIVERINGLIBRARY },
                  event { "book transferred" },
                  reaction { "local update of book table" }  }
}}  },
    in { resource { "file:books.xml" },
      books {{
        book {{
          callNumber { var CALLNUMBER },
          library { var DELIVERINGLIBRARY replaceby var REQUESTINGLIBRARY }
}}  }}  }  }
ON
  event {{
    type { "book transferred" },
    callNumber { var CALLNUMBER },
    requestingLibrary { var REQUESTINGLIBRARY },
    deliveringLibrary { var DELIVERINGLIBRARY }  }}
END
```

Working group node 2

```
# R001 - insert member
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
```

```
                 sender { "management node" },
                 event { "insert member" },
                 reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
      members {{
        insert var MEMBER  }}  }  }
ON
  event {{
    type { "insert member" },
    var MEMBER -> member {{}}  }}
END
```

```
# R002 - delete member
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "management node" },
                 event { "delete member" },
                 reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
      members {{
        delete member {{
                 surname { var SURNAME },
                 firstName { var FIRSTNAME },
                 title { var TITLE },
                 email { var EMAIL },
                 phone { var PHONE },
                 office { var OFFICE }  }}  }}  }  }
ON
  event {{
    type { "delete member" },
    member {{
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE }  }}  }}
END
```

```
# R003 - change member data
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "change node" },
                 event { "insert member" },
                 reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
      members {{
        member {{
          surname { var OLD_SURNAME replaceby optional var NEW_SURNAME },
          firstName { var OLD_FIRSTNAME replaceby optional var NEW_FIRSTNAME },
          title { var OLD_TITLE replaceby optional var NEW_TITLE },
          email { var OLD_EMAIL replaceby optional var NEW_EMAIL },
          phone { var OLD_PHONE replaceby optional var NEW_PHONE },
          office { var OLD_OFFICE replaceby optional var NEW_OFFICE },
          ecssWorkingGroups {{}} replaceby var NEW_ECSSWORKINGGROUPS
}}  }}  }  }
ON event {{
    type { "change member" },
    oldRecord {{
      surname { var OLD_SURNAME },
      firstName { var OLD_FIRSTNAME },
      title { var OLD_TITLE },
      email { var OLD_EMAIL },
      phone { var OLD_PHONE },
      office { var OLD_OFFICE }  }},
    newRecord {
      surname { optional var NEW_SURNAME },
      firstName { optional var NEW_FIRSTNAME },
      title { optional var NEW_TITLE },
      email { optional var NEW_EMAIL },
      phone { optional var NEW_PHONE },
      office { optional var NEW_OFFICE },
      var NEW_ECSSWORKINGGROUPS -> ecssWorkingGroups {{}}  }  }}
END
```

```
# R004 - insert news (book)
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
```

```
                  sender { "management node" },
                  event { "news: new book published" },
                  reaction { "local update of news" }  }  }}  },
    in { resource {"file:news.xml"},
      news {{
        insert entry {
                  type { "book" },
                  message { var MESSAGE }  }  }}  }  }
ON
  event {{
    type { "book" },
    message {{ var MESSAGE }}  }}
END
```

```
# R005 - insert news (review)
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management node" },
                  event { "news: new review available" },
                  reaction { "local update of news" }  }  }}  },
    in { resource {"file:news.xml"},
      news {{
        insert entry {
                  type { "review" },
                  message { var MESSAGE }  }  }}  }  }
ON
  event {{
    type { "review" },
    message {{ var MESSAGE }}  }}
END
```

```
# R006 - insert news (talk)
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management node" },
                  event { "news: new talk or workshop announced" },
                  reaction { "local update of news" }  }  }}  },
    in { resource {"file:news.xml"},
```

```
        news {{
          insert entry {
                    type { "talk" },
                    message { var MESSAGE }  }  }}  }  }
ON
  event {{
    type { "talk" },
    message {{ var MESSAGE }}  }}
END
```

Working group node 3

```
# R001 - insert member
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management node" },
                  event { "insert member" },
                  reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
      members {{
        insert var MEMBER  }}  }  }
ON
  event {{
    type { "insert member" },
    var MEMBER -> member {{}}  }}
END
```

```
# R002 - delete member
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management node" },
                  event { "delete member" },
                  reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
```

```
        members {{
          delete member {{
                    surname { var SURNAME },
                    firstName { var FIRSTNAME },
                    title { var TITLE },
                    email { var EMAIL },
                    phone { var PHONE },
                    office { var OFFICE }  }}  }}  }  }
ON
  event {{
    type { "delete member" },
    member {{
      surname { var SURNAME },
      firstName { var FIRSTNAME },
      title { var TITLE },
      email { var EMAIL },
      phone { var PHONE },
      office { var OFFICE }  }}  }}
END
```

```
# R003 - change member data
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                    sender { "management node" },
                    event { "change member" },
                    reaction { "local update of member data" }  }
}}  },
    in { resource {"file:members.xml"},
      members {{
        member {{
          surname { var OLD_SURNAME replaceby optional var NEW_SURNAME },
          firstName { var OLD_FIRSTNAME replaceby optional var NEW_FIRSTNAME },
          title { var OLD_TITLE replaceby optional var NEW_TITLE },
          email { var OLD_EMAIL replaceby optional var NEW_EMAIL },
          phone { var OLD_PHONE replaceby optional var NEW_PHONE },
          office { var OLD_OFFICE replaceby optional var NEW_OFFICE },
          ecssWorkingGroups {{}} replaceby var NEW_ECSSWORKINGGROUPS
}}  }}  }  }
ON event {{
    type { "change member" },
    oldRecord {{
      surname { var OLD_SURNAME },
      firstName { var OLD_FIRSTNAME },
      title { var OLD_TITLE },
```

```
      email { var OLD_EMAIL },
      phone { var OLD_PHONE },
      office { var OLD_OFFICE }  }},
    newRecord {
      surname { optional var NEW_SURNAME },
      firstName { optional var NEW_FIRSTNAME },
      title { optional var NEW_TITLE },
      email { optional var NEW_EMAIL },
      phone { optional var NEW_PHONE },
      office { optional var NEW_OFFICE },
      var NEW_ECSSWORKINGGROUPS -> ecssWorkingGroups {{}}  }  }}
END
```

```
# R004 - insert news (book)
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management node" },
                  event { "news: new book published" },
                  reaction { "local update of news" }  }  }}  },
    in { resource {"file:news.xml"},
      news {{
        insert entry {
                  type { "book" },
                  message { var MESSAGE }  }  }}  }  }
ON
  event {{
    type { "book" },
    message {{ var MESSAGE }}  }}
END
```

```
# R005 - insert news (review)
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                  sender { "management node" },
                  event { "news: new review available" },
                  reaction { "local update of news" }  }  }}  },
    in { resource {"file:news.xml"},
      news {{
```

```
        insert entry {
                 type { "review" },
                 message { var MESSAGE }  }  }}  }  }
ON
  event {{
    type { "review" },
    message {{ var MESSAGE }}  }}
END
```

```
# R006 - insert news (talk)
TRANSACTION
  and {
    in { resource {"file:log.xml"},
      log {{
        insert entry {
                 sender { "management node" },
                 event { "news: new talk or workshop announced" },
                 reaction { "local update of news" }  }  }}  },
    in { resource {"file:news.xml"},
      news {{
        insert entry {
                 type { "talk" },
                 message { var MESSAGE }  }  }}  }  }
ON
  event {{
    type { "talk" },
    message {{ var MESSAGE }}  }}
END
```

References

[1] José Júlio Alferes and Wolfgang May. Deliverable I5D2. Use-cases on evolution. Technical report, REWERSE, 2005.

[2] François Bry and Michael Eckert. A High-Level Query Language for Events. In *Int. Workshop on Event-driven Architecture, Processing and Systems*. IEEE, 2006.

[3] Oscar Díaz. Tool Support. In Norman W. Paton, editor, *Active rules in database systems*, pages 127–176. Springer-Verlag New York Inc., 1999.

[4] International Organization for Standardization. Numeric representation of Dates and Time. `http://www.iso.org/iso/en/prods-services/popstds/datesandtime.html`.

[5] Paula-Lavinia Pătrânjan. *The Language XChange: A Declarative Approach to Reactivity on the Web*. PhD thesis, Ludwig-Maximilians-Universität München, 2005.

[6] Inna Romanenko. Use Cases for Reactivity on the Web: Using ECA Rules for Business Process Modeling. Master's thesis, Ludwig-Maximilians-Universität München, 2006.