I5-D2

Use-cases on evolution

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Lisbon/I5-D2/D/PU/a1
Responsible editors:	José Júlio Alferes and Wolfgang May
Reviewers:	Nicola Henze and Michael Schroeder
Contributing participants:	Dresden, Goettingen, Hannover, Lisbon, Munich, Skoevde
Contributing workpackages:	I5
Contractual date of deliverable:	28 February 2005
Actual submission date:	28 February 2005

Abstract
This report presents a set of use cases for evolution and reactivity for data in the Web and Semantic Web. This set is organized around three different case study scenarios, each of them is related to one of the three different areas of application within REWERSE. Namely, the scenarios are: "The REWERSE Information System and Portal", closely related to the work of A3 – Personalised Information Systems; "Organizing Travels", that may be related to the work of A1 – Events, Time, and Locations; "Updates and evolution in bioinformatics data sources" related to the work of A2 – Towards a Bioinformatics Web.

Keyword List
Languages for updates, ECA rules, Event languages, Reactivity, Evolution of data

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

© *REWERSE 2005.*

Use-cases on evolution

José Júlio Alferes[1], **Mikael Berndtsson**[2], **François Bry**[3], **Michael Eckert**[3], **Nicola Henze**[4] **Wolfgang May**[5], **Paula Lavinia Pătrânjan**[3], **Michael Schroeder**[6]

[1] Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa

[2] School of Humanities and Informatics, University of Skövde

[3] Institut für Informatik, Ludwig-Maximilians-Universität München

[4] Institut für Informatiomssysteme, Universität Hannover

[5] Institut für Informatik, Universität Göttingen

[6] Biotec/Dept. of Computing, TU Dresden

28 February 2005

Abstract

This report presents a set of use cases for evolution and reactivity for data in the Web and Semantic Web. This set is organized around three different case study scenarios, each of them is related to one of the three different areas of application within REWERSE. Namely, the scenarios are: "The REWERSE Information System and Portal", closely related to the work of A3 – Personalised Information Systems; "Organizing Travels", that may be related to the work of A1 – Events, Time, and Locations; "Updates and evolution in bioinformatics data sources" related to the work of A2 – Towards a Bioinformatics Web.

Keyword List

Languages for updates, ECA rules, Event languages, Reactivity, Evolution of data

Contents

1 Introduction **1**

2 Concepts and Requirements **3**
2.1 Active Rules . . . 4
2.1.1 Trigger-Like Local ECA Rules . . . 4
2.1.1.1 XML . . . 5
2.1.1.2 RDF . . . 5
2.1.1.3 Condition and Action Part . . . 6
2.1.2 Local and Global ECA Rules . . . 6
2.2 Updates and Actions . . . 7
2.3 Communication . . . 7

3 Summary of the scenarios **9**
3.1 Rewerse Information System and Portal . . . 9
3.2 Organising Travels . . . 10
3.3 Updates in Bioinformatics Data Sources . . . 11

4 Rewerse Information System and Portal **13**
4.1 Overview of Use Cases . . . 14
4.1.1 Propagation of Updates: Upwards . . . 15
4.1.2 Information Propagation and Distribution: Downwards . . . 15
4.1.3 Personalization and Presentation . . . 16
4.1.4 Negotiations and Policies . . . 16
4.1.5 Composite Events and Composite Actions . . . 16
4.1.6 Further Extensions . . . 16
4.2 Use Cases: XML . . . 17
4.2.1 The Data . . . 17
4.2.2 Maintenance Use Cases . . . 20
4.2.3 Dissemination Use Cases . . . 24
4.2.4 Personalization Use Cases: Local Evolution . . . 28
4.2.5 Reasoning and Evolution Use Cases . . . 28
4.2.6 Composite Events Use Cases . . . 30
4.2.7 Composite Actions Use Cases . . . 35
4.3 Further issues . . . 37

5 Organising Travels Scenario **39**
5.1 Language issues . . . 39
5.2 Initial Planning . . . 41
5.2.1 Gather information and plan trip . . . 41
5.2.2 Arranging the trip according to plan . . . 41
5.2.3 Scenario . . . 42
5.3 Adapt Plan to Changes . . . 46
5.3.1 Recognize changes affecting the plan . . . 46
5.3.2 React to changes . . . 46
5.3.3 Scenario . . . 47

6 Updates and Evolution in Bioinformatics Data Sources **51**
6.1 Bioinformatics and the Semantic Web . . . 51
6.2 Sample data sources and application . . . 52
6.2.1 GoPubMed, ontology-based literature search . . . 52
6.2.1.1 GeneOntology . . . 53
6.2.1.2 PubMed . . . 53
6.2.2 PDB and SCOP . . . 53
6.3 Organisation of Bioinformatics Data . . . 54
6.4 Reactivity/evolution scenario: GoPubMed, GO, PubMed, SCOP, and PDB . . . 54

7 Conclusions **57**

Chapter 1

Introduction

The Web and the Semantic Web, as we see it, can be understood as a "living organism" combining autonomously evolving data sources, each of them possibly reacting to events it perceives. The dynamic character of such a Web requires declarative languages and mechanisms for specifying the evolution of the data.

This vision of the Web, as well as a state of the art overview of related areas, is described in our previous work ([ABB+04, MAB04]). Rather than a Web of data sources, we envisage of Web of Information Systems, where each such system, besides being capable of gathering information (querying, both on persistent data, as well as on volatile data such as occurring events), is capable of updating persistent data, communicating the changes, requesting changes of persistent data in other systems, and being able to react to requests from other systems.

In this report we present three case study scenarios for such an evolving Web, each with a set of use-cases. It is our goal that the scenarios and use-cases show the potential applicability of the concepts and systems we are developing, and intend to develop within REWERSE. Moreover, these use-cases should make it clear what are the goals and features a system for evolution in the Web should have, and also to delimit the scope of what we intend to develop. In other words, the set of use-cases should be such that it can serve as guidance, based on the example scenarios, for the development of a language for evolution and reactivity on the Web, containing a sufficiently rich set of cases for testing the various required features.

The choice of the case study scenarios had these goals in mind, but also the goal of fostering a greater integration within the project REWERSE. Namely, it was our goal to have case study scenarios close to the application areas being developed in the "Application Working Groups" of REWERSE. This lead us to the development of three scenarios:

Rewerse Portal. This scenario is a joint activity of the working groups PRA (Project administration and presentation), TTA (Technology Transfer), I4 (Querying), I5 (Evolution & Reactivity), and A3 (Personalization). Its targets are to develop a portal for:

- exchanging and collecting information about the project, its participants, and its results;
- and for presenting and providing information about the project on the (Semantic) Web.

Organising Travels. This scenario is concerned with planning travels based on information gathered in the web, and acting on the web for the organization of such travels (e.g. by

buying train tickets, booking flights online, etc). The issue of reacting to happenings that influence the initial plan, and re-planning accordingly, is also taken in consideration in this scenario. In the use-cases it is made clear that location reasoning and the capability to use different calendars and constraints over time notions is needed, and thus the potential relation to work developed in WG A1 (events, time, and locations).

Updates and evolution in bioinformatics data sources. In bioinformatics there are many publicly accessible data sources, which are often mirrored locally and integrated with other data. The bioinformatics use case discusses four specific data sources, PubMed, a database of 12.000.000 biomedical literature abstracts, GeneOntology, an ontology for moelecular biology, with 19.000 concepts, PDB, a database with some 25.000 protein structures, and SCOP, the Structure Classification of Proteins, which groups PDB structures according to their evolutionary relationships. The scenario is concerned with mirroring these data sources locally, keeping them consistent and integrating them, and its relation and usefulness to the work being developed in working group A2 is clear.

These scenarios are detailed in Chapters 4, 5 and 6. Each of them illustrates different features and aspects of Evolution and of Reactivity. Before presenting the scenarios, in Chapter 2 we expose and briefly analyze some of these aspects that we find important to illustrate for Evolution and Reactivity, and in Chapter 3, in order to better guide the reading, we summarizes which aspects are worked out by which scenarios.

It is worth noting that in the scenarios we do not distinguish between Evolution and Reactivity, though the deliverables in REWERSE were structured according to such a distinction. As we see it, Evolution in the Web is concerned with (consecutive) modification of data and propagation of modifications, whilst Reactivity is concerned with detection of changes, and possibility to take actions (most likely updates, which determine evolution) in reaction to those changes. In examples and, as such, in scenarios and use-cases, it is quite difficult, and perhaps even unnatural, to decouple these two aspects of changes of data on the Web. Having examples simply concerned with dealing with modifications, and without concerns about how these modifications where triggered, or having examples concerned with triggering of modifications without concerns about how these modification are realised would not show the full potential of Evolution and Reactivity, and would not make clear what the whole of the problems involved are. Thus, we opted to describe use-cases for both Evolution and Reactivity, without separating these two aspects.

Chapter 2

Concepts and Requirements

Web-applications dealing with evolution and reactivity usually have the following common characteristics:

- *They gather information,* that is they query (persistent and volatile) data and reason with them.
- *They update persistent data,* that is they modify data in Web resources (e.g. XML or RDF data). Such modifications often have to happen in an all-or-nothing manner, posing a requiring support for transactions.
- *They react to changes in data,* that is they communicate notifications about changes in data (events), detect situations of interest (i.e. time-related patterns of events), and execute reactions to such situations.

For realizing a system with these characteristics, we proposed in [ABB+04, MAB04] to use active *event-condition-action* like (ECA) rules. More precisely, we assume that communication in the Semantic Web takes place by *peer-to-peer* communication between resources, and that this communication is based on *messages* and *events* (that both are represented or wrapped in XML). Active rules, similar to event-condition-action rules, are used for communication and for specification of the local behavior of Semantic Web nodes. Such a rule is triggered by events, then optionally checks for conditions and finally takes appropriate actions (in most cases, updating local information accordingly).

Events are either atomic events (e.g., updates in the local databases, received messages, or happenings on the Web), or complex events formed as combinations of atomic ones (e.g. "when A is updated, and then B is updated") expressed in some event algebra. Moreover, events can be either *explicit* (in case they are explicitly communicated, through event messages, to the node or resource), or *implicit* (in case they are not explicitly communicated, and have to be somehow automatically detected, such as local updates of data or system events, e.g. system clock events). Conditions denote queries to one or several nodes and are to be expressed in the a query language. Atomic actions are e.g. update requests or sending a message, and actions can be grouped as transactions.

Gathering information is a matter of querying data that, though essential for Evolution and Reactivity, is not the main subject of our study (and is studied elsewhere in Rewerse). So, we assume that various query languages exist that make it possible for data to be queried,

and conditions to be tested in ECA-rules. The aspects of evolution and reactivity developed by us should, as much as possible, be parametric on the query language. In the use-cases we consider various possibilities such as XPath/XQuery (mostly in Chapter 4) and Xcerpt (mostly in Chapter 5).

In the remainder of this chapter we analyse several relevant aspects of Evolution and Reactivity via these rules for structuring the systematic choice and presentation of use-cases. Namely, we analyse:

- rules: database triggers or high-level active rules,
- actions: explicit updates or actions,
- communication issues.

2.1 Active Rules

There are several abstraction levels on which active rules can be defined:

- programming language level: triggers as built-in constructs of a given database model, like SQL triggers. Usually they are implemented inside the database. This level can e.g. directly be based on the *DOM Level 2/3 Events* [DOM98].
- logical level – XML. Here ECA rules consist of distinguished event-condition-action parts that are also marked up in XML/RuleML; one of the results of the research in I5 (jointly with I1) should be an ECA-ML language.

  This requires a definition of atomic update events on XML data; probably on the same level and granularity as updates in XUpdate located by XSL patterns or by using an update language like XChange[1]
- semantic level: RDF. Here, several aspects can (also independently) be lifted from XML:
  - use XML-ECA rules on underlying RDF/OWL data,
  - use RDF/OWL descriptions of events, conditions, and actions in the XML-ECA framework,
  - use an RDF/OWL ontology even on the rule level. (Conversely, rules in this ontology can themselves use event/condition/action parts in XML, and even data in XML).

2.1.1 Trigger-Like Local ECA Rules

Trigger-like local ECA Rules have to react directly on the changes of the database, which is assumed to be in XML or RDF format. While triggers in relational databases/SQL were only able to react on changes of a given tuple or an attribute of a tuple, the XML and RDF models call for more expressive event specifications according to the (tree or graph) structure.

[1] For more details on XChange, and its usage in this context, see Chapter 5.

2.1.1.1 XML

For modifications of an XML tree, the following atomic events could be considered:

- `ON DELETE OF` *xsl-pattern*: if a node matching the *xsl-pattern* is deleted,
- `ON INSERT OF` *xsl-pattern*: if a node matching the *xsl-pattern* is inserted,
- `ON MODIFICATION OF` *xsl-pattern*: if anything in the subtree is modified,
- `ON UPDATE OF` *xsl-pattern*: the value (text or attribute) of a node matching the *xsl-pattern* is modified,
- `ON INSERT INTO` *xsl-pattern*: if a node is inserted (directly) into a node matching the *xsl-pattern*,
- `ON INSERT [IMMEDIATELY] BEFORE|AFTER` *xsl-pattern*: if a node is inserted (immediately) before or after a node matching the *xsl-pattern*.

All triggers should make relevant values accessible, e.g., `OLD AS ...` and `NEW AS ...` (like in SQL), both referencing the complete node to which the event happened, additionally `INSERTED AS`, `DELETED AS` referencing the inserted or deleted node.

Similar to the SQL `STATEMENT` and `ROW` triggers, the granularity has to be specified for each trigger; the following granularities are proposed here:

- `FOR EACH STATEMENT` (as in SQL),
- `FOR EACH NODE`: for each node in the *xsl-pattern*, the rule is triggered only at most once (cumulative, if the node is actually concerned by several matching events) per transaction,
- `FOR EACH MODIFICATION`: each individual modification (possibly for some nodes in the *xsl-pattern* more than one) triggers the rule.

For data-dependent information propagation, mainly `FOR EACH NODE` and `FOR EACH MODIFICATION` are adequate.

The implementation of such triggers in XML repositories is probably to be based on the *DOM Level 2/3 Events* [DOM98].

2.1.1.2 RDF

RDF triples, describing properties/values of a resource are much more similar to SQL. In contrast to XML, there is no assignment of data with subtrees (which makes it impossible to express "deep" modifications in a simple event; such things have then to be expressed in the condition part).

- `ON DELETE OF` *property* `[OF` *class*`]`,
- `ON INSERT OF` *property* `[OF` *class*`]`,
- `ON UPDATE OF` *property* `[OF` *class*`]`.

If a property is removed from/added to/updated of a resource of a given class, then the event is raised.
Additionally,

- `ON CREATE OF` *class* is raised if a new resource of a given class is created.

Probably, also metadata changes have to be detected:

- `ON NEW CLASS` is raised if a new class is introduced,
- `ON NEW PROPERTY [OF CLASS` *class*`]` is raised, if a new property (optionally: to a specified class) is introduced.

All triggers should make relevant values accessible, e.g., `OLD AS ...` and `NEW AS ...` (like in SQL), both referencing the original/new value of the property, `RESOURCE AS ...` and `PROPERTY AS ...` refer to the modified resource and the property (as URIs), respective.
Trigger granularity is `FOR EACH STATEMENT` or `FOR EACH TRIPLE`.

2.1.1.3 Condition and Action Part

Since triggers are on the programming language level, the condition part is (as in SQL) either a simple comparison, or, a boolean query in the appropriate query language (XQuery, Xcerpt, or an RDF query language).

Similarly, the action part must specify an action that can be executed by the database, usually an update or a program. In case that sending a message to the outside is supported, this can also be done.

2.1.2 Local and Global ECA Rules

While "triggers" are restricted, programming-language concepts, general ECA rules provide an abstract concept using an own language. Especially in our setting, they are usually separated from the database. Thus, they do not react on "physical" events in the database, but on *logical* events (that nevertheless are actually raised by events in a database).

ECA rules are marked up in the language that we will probably call ECA-ML (XML), or even formulated more abstractly in RDF, using an OWL ECA ontology. In general, they use sublanguages for events (EventML), conditions (allowing to embed XQuery), and (trans)actions (embedding SOAP for service calls as atomic actions).

Local ECA Rules. Local ECA rules are more general than triggers. They still react on local events only, but they use an own event language that is based on a set of atomic events (that are not necessarily simple update operations) and that usually also allows for composite events. Their event detection mechanism is not necessarily located in the database. Detection of atomic logical events can be based on

- database triggers that generate events that are visible/detectable outside the database, or
- they have to poll the database regularly if such an event occurred.

Global ECA Rules. Global ECA rules have to be used if a composite event consists of subevents at different locations (or if the source of an event is not able to process local rules). When considering global rules in the Web and in the Semantic Web

- "distributed" variants of the above local ECA rules, with events that explicitly mention a database/node where the event is located (e.g., "change of *xpath-expr* at *url*"),
- rules that react on events in a set of known databases (e.g., "when a new researcher is added at one of the participants nodes" (which itself is a dynamic set)),
- high-level rules of an application, that are not based on schema knowledge of individual databases, often even not explicitly on a given database (e.g., "when a publication p becomes known that deals with ..."). Here, Semantic Web reasoning comes heavily into play even for detecting atomic events "somewhere in the Web".

The capability to detect and react to composite events is needed for many Web-based reactive applications, and is exemplified in the scenario of Chapter 5. However, to the best of our knowledge, existing languages for reactivity on the Web do *not* consider the issues of detecting and reacting to such composite events, and further developements are need in this issue [2].

2.2 Updates and Actions

There are different ways how to express the actions to be taken.

Explicit updates: In this case the action is an explicit update statement e.g. described in XUpdate, XQuery+Updates, XChange, or in an RDF Update language. This requires knowledge of the underlying schema.

Explicit actions: In this case by calling a procedure/method (SOAP),

Semantic: This requires the declarative specification of what has to be changed, using an RDF/OWL ontology of changes (to RDF data).

2.3 Communication

Peer-to-peer-communication in the Web and in the Semantic Web can be done in two distinct ways:

- Push: a node informs registered nodes. A directed, targeted propagation of changes by the *push* strategy is only possible along registered communication paths. It takes place by explicit *messages*.
- Pull: resources that obtain information from a fixed node can *pull* updates by either explicitly asking the node whether it executed some updates recently, or can regularly update themselves based on queries against their providers. Communication is based on queries and answers (that are in fact again sent as messages).

[2] [BKK04] considers "composite events". However, this notion refers in [BKK04] to updates of several elements of a single XML document.

We need this distinction also on two levels:

- application-level: how to communicate changes from one node to another? The basic choice of using "push" or "pull" communication influences the design of the rules that implement the communication.
- infrastructure level: how to communicate events and how to raise actions.
  - Push (events): sending a message that contains the description of an event (e.g., the phone number of person P has been changed to 1234), which can either be in an agreed XML format, or in RDF. The receiving node must then decide how to react on this event.
  - Pull (events): seeing events as events "on the Web", and other nodes must detect these events by themselves (or supported by third-party event detection mechanisms).
  - Push (actions): sending a message that contains explicitly the action/service call to be taken there. The action can be an explicit update, or a description of an action in XML/SOAP or RDF.
  - note that the contrary of "push actions" is not "pull actions", but that actions are not pushed, i.e., only local actions are allowed.

Chapter 3

Summary of the scenarios

Each of the three proposed scenarios is meant to illustrate different aspects of evolution and reactivity, and to focus on different possible underlying languages (e.g. the scenarios on "organizing travel" focuses on Xcerpt and XChange, while the "Rewerse portal" scenarios focuses more on XPath/XQuery and XUpdate). In this Chapter, to better guide the reading of the report, we summarize which aspects are better illustrated in each scenario.

3.1 Rewerse Information System and Portal

The Rewerse Personalized Portal (Rewerse-IS&PP) scenario is a joint activity of several working groups in the project, which all contribute with their specific expertise to build and explore innovative (personalized) semantic information portals. We can distinguish between technology providers for the Rewerse-IS&PP (currently working groups I5, A3), and users (TTA, PRA). As technology providers for the scenario, the working group I5 focuses on the dynamic and evolutionary aspects of the information system, while A3 is focussing on architectures and personalization functionality for the portal (see deliverable A3-D3). Scenarios and use-cases for the Rewerse-IS&PP have been developed in co-operation with working groups TTA and PRA. Its targets are (i) exchanging and collecting information about the project, its participants, and its results, and (ii) presenting and providing information on the (Semantic) Web. The scenario consists of the following nodes:

- Participant's nodes: The structure of the Web nodes of the participants is not dedicated to their role as Rewerse participants, but is an XML or RDF repository that contains data (e.g. names of participant, publications, interests, etc) about a research group. The only relationship to Rewerse is given in a block of project-specific information (e.g., funding), and in the enumeration of working groups where a person is involved.

  Thus, communicating data to the other nodes then relies on active rules that e.g. react if an entry "Rewerse" is added to a person's projects.

- Working group nodes: here, a common schema is used for all nodes (scenario: both in XML and in RDF), and nodes contain information relative to the various working groups of Rewerse, including information about participants, deliverables, coordinators, meetings, etc.

- Central project node (scenario: both in XML and in RDF), containing general information about the project. The externally visible functionality as a portal is associated with this central node. In it, e.g. personalization is implemented.

The repositories can be either in XML or in RDF – only the rules must be formulated appropriately.

This is one scenario where we can have control over what is being developed, and can serve as a good controlled environment, where both the ontology and the organization of the nodes is known, to illustrate (and, afterwards, to test) evolution and reactivity aspects. In it we illustrate in detail:

- some basic aspects of information propagation (both upwards from participants to working groups and to the central node, and downwards from the central node all the way down to participants);
- some communication issues, such as push and pull communication strategies;
- usage of trigger-like, local, and also global ECA rules;
- use of active rules (and reasoning) for negotiation, e.g. for agreeing on dates for meetings;

Though in this scenario we elaborate on some example involving complex (composite and cumulative) events and composite actions and transactions, most of the focus on the use-cases is on atomic events and simple actions.

Most of the use-cases are developed at the XML level, some considerations on lifting to the RDF/OWL level being simply sketched at the end. Moreover the use-cases do not assume any particular query or update language for XML data, and for illustrative purposes, XPath/XQuery and XUpdate are mostly used.

3.2 Organising Travels

This scenario is concerned with planning travels based on information gathered in the web, and acting on the web for the organization of such travels (e.g. by buying train tickets, booking flights online, etc). The issue of reacting to happenings that influence the initial plan, and re-planning accordingly, is also taken in consideration in this use-case.

This is a more general scenario, in a much less controlled environment. As such, the issue of gathering information, and of reasoning on such information, plays in this scenario a much more central role than in the previous one. Though the querying features are outside the specific scope of our working group (I5), and are studied and developed elsewhere in Rewerse (in working group I4), it is essential to consider them in the context of this scenario. As such, this is a good scenario to illustrate and, afterwards, experiment with the integration of querying and evolution aspects developed in Rewerse, this being a focus on the use-cases for this scenario. Being so closely related with the work on queries, rather than what happens in the previous scenario, in this one, all example are illustrated with query and update languages developed in Rewerse; namely, the query language Xcerpt and the update language XChange are used in examples.

Besides the greater integration with queries, this scenario is also more adequate to show some aspects of complex events and transactions, and some examples of such are given in the use-cases.

3.3 Updates in Bioinformatics Data Sources

The scenario describes four specific data sources, all with considerable volume of data, and applications that integrate them. The following aspects are specific to the bioinformatics scenario:

- The data is distributed and the original sites that publish data are typically outside the control of the application. The remote databases are often cached.
- There are primary and secondary databases. The latter are based on the former.
- Data sources are updated at different intervals ranging from weekly to yearly updates.
- Data sources are published in various formats such as free text, relational database, or XML.
- It is necessary to pose recursive queries over ontologies.

Chapter 4

Rewerse Information System and Portal

In this Chapter we describe the Rewerse Information System and Portal as a scenario of a (Semantic) Web application with the focus on *evolution and reactivity*. Using this scenario, we partition *evolution and reactivity* into several aspects that can be investigated and combined in a modular way. The scenario and the use-cases are described and analyzed first on the Web (XML) level. Moving towards the Semantic Web (RDF/OWL) level, most of the aspects of *evolution and reactivity* can be treated individually, often just by appropriate RDF/OWL representations and mapping them to the underlying mechanisms.

The Rewerse Personalized Portal (Rewerse-IS&PP) scenario is a joint activity of several working groups in the Project, which all contribute with their specific expertise to build and explore innovative (personalized) semantic information portals. We can distinguish between technology providers for the Rewerse-IS&PP (currently working groups I5, A3), and users (TTA, PRA). As technology providers for the scenario, the working group I5 focuses on the dynamic and evolutionary aspects of the the information system, while A3 is focussing on architectures and personalization functionality for the portal (see deliverable A3-D3). Scenarios and use-cases for the Rewerse-IS&PP have been developed in co-operation with working groups TTA and PRA. Its targets are (i) exchanging and collecting information about the project, its participants, and its results, and (ii) presenting and providing information on the (Semantic) Web. The scenario consists of the following nodes:

- Participant's nodes: The structure of the Web nodes of the participants is not dedicated to their role as Rewerse participants, but is an XML or RDF repository that contains data (e.g. names of participant, publications, interests, etc) about a research group. The only relationship to Rewerse is given in a block of project-specific information (e.g., funding), and in the enumeration of projects where a person is involved.

  Thus, communicating data to the other nodes then relies on active rules that e.g. react if an entry "Rewerse" is added to a person's projects.

- Working group nodes: here, a common schema is used for all nodes (scenario: both in XML and in RDF).

- the central project node (scenario: both in XML and in RDF).

Note that the repositories can be either in XML or in RDF – only the rules must be formulated appropriately.

The externally visible functionality as a portal is associated with the central node. There, e.g. personalization is implemented.

Ontology. For the REWERSE Portal, an ontology of the REWERSE project - its organization, working groups, persons, participants, etc., has been developed (see deliverable A3-D3).

Propagation of Updates/Materialization vs. View Definition. In the scenario, the data integration aspect is mainly implemented by propagation of updates (i.e., materializing and maintaining the integrated information) instead of a view-based integration (by mappings in the global-as-view or local-as-view style). We prefer this for the following reasons:

- updates are infrequent, so the maintenance overhead is low,
- queries that each time involve contacting all WG and participants nodes and all participants are ineffective, and would depend too much on network accessibility,
- local consistency conditions (especially at the central node), e.g. on publications of multiple coauthors, (note that this is also concerned with a notion of global consistency between different nodes in the Web) are better supported with materialized data,
- local active rules to be fired upon updates can then be based on local events (both rules on participant nodes upon e.g. new appointments, and rules on the central node upon local updates).

Although the actual realization will be based in propagating and materializing information, we will also illustrate rules that react on remote events by this scenario.

In the next section, we give an overview of how the various use cases are structured and presented with respect to the relevant issues spelled out in Chapter 2. Section 4.2 then discusses the Portal Scenario data and use cases for the XML case. Here, we systematically list the features that an ECA language for evolution and reactivity on the Web should provide. Section 4.3 concludes this Chapter with a short summary of the requirements, related work, and the lifting to the RDF/OWL level.

4.1 Overview of Use Cases

The use cases are partitioned into several aspects:

- maintenance of base data: simple propagation of updates from the participants and WGs to the central node
- data-dependent flow of information and communication: appointments and to-do-lists from the central node to the WGs and the participants, and from the WGs to participants,
- personalization and presentation,

- application-specific tasks: e.g., negotiations of dates for meetings.

So far, the use-cases are partitioned wrt. their role in the application. Some more use-cases that use composite events are also discussed.
Below, we describe these situations, and the roles of *evolution* and *reactivity*.

4.1.1 Propagation of Updates: Upwards

Evolution in this aspect means user-driven updates to the scenario which are then propagated (upwards) through the nodes, e.g.,

- Inserting a new person. Persons are inserted by the participants into their nodes. The insertions are then propagated to the WGs where the person is associated and to the central node.

- Analogous for deleting persons or changing information about a person (e-mail, phone, fax etc.)

Here, the choice to which nodes a message has to be sent, is relatively simple. Communication is then by reactivity:

- the updated node reacts on user-driven updates by sending appropriate messages to other nodes, and then the other nodes react on the message (leading to local evolution again) , or

- other nodes react on the event (which is for them a remote event), again leading to evolution.

Obviously, global evolution (of the information system) consists of local evolution in all nodes which is coordinated by reactivity.

4.1.2 Information Propagation and Distribution: Downwards

Information from the project coordination office and from the WG coordinators is propagated downwards, e.g., dates for meetings which are then added to the WG and participants' nodes.

In case of a *push* communication, the subordinate node must determine to which nodes the message/action must be sent/directed. If the action is an explicit update of a node, this selection must be complete and correct, since the remote nodes are definitely updated. In case that an informative message is sent that contains some information, the selection has only to be complete, since the nodes can then decide by themselves what to do with the message; note that this information may be broadcasted to all participant nodes.

In case of a *pull* communication, each node for which the event is relevant, must detect it by its own and execute appropriate actions.

In the same way as above, global evolution (of the information system) consists of local evolution in all nodes which is coordinated by reactivity. Here, the disseminated information causes further reactions and evolution, e.g., when a new appointment or task becomes known to a participant.

4.1.3 Personalization and Presentation

The scenario serves as an internal information system, and includes the portal that serves to the external world for presentation of the project and its results.

Personalization takes place in at least two issues. First, for researchers in the project where every researcher probably wants to have an own view of the project that includes his primary WG (e.g., I5), some WGs where he is involved in (e.g., I4 and A3), and an overview of the results of other WGs (e.g., I1 for general work on rule markup languages). Here, large parts of the information that is required for personalization is already contained in the project data; additionally, users should be able to define their own rules, and furthermore the system can probably be adaptive. For external users, restricted rules could be definable, e.g., depending on their research area.

Here, local evolution of the personalizated node/service results from local and remote events; in our approach this takes place by reaction rules.

4.1.4 Negotiations and Policies

Fixing dates for meetings depends on a lot of constraints. Here, the use of a calendar together with the personal schedules of prospective participants (e.g., exams, travels etc) can help to select possible and preferable dates.

This task consists of (i) communication and (ii) reasoning. Communication e.g. comprises to ask the participants' nodes for the available dates of certain participants. Reasoning is then applied for combining the available information and proposing a set of dates based on policies. Policies are e.g. of the form "meetings must not include holidays in the country where the meeting takes place", or "if there is a time/place where several relevant persons meet for a conference, then preferably a meeting should be associated with this event". If an appointment is made, it is communicated according to the description in Section 4.1.2.

Here, reactivity (for reasoning) plays a much more important role as in the first two situations for propagating and disseminating information. Evolution then results from reasoning.

4.1.5 Composite Events and Composite Actions

The previous sections show different aspects of ECA rules and of communication and detection of mostly atomic events. A separate section is devoted to the investigation of composite events. This aspect is mainly concerned with providing powerful mechanisms for expressing and controlling reactivity.

In the same way, the action part of ECA rules can contain transaction specifications. A short section describes this well-known concept. This aspect is mainly concerned with providing powerful mechanisms for specifying evolution.

4.1.6 Further Extensions

There are several extensions that can be added to the scenario for demonstrating additional functionality. Some of them are mentioned below.

- a global bibliography/repository of relevant documents (including documents from outside the project), possibly with annotations (cf. the Personal Reader),

- functionality for organizing travels to meetings: when the systems knows that a certain person will attend the meeting, possible flight/train connections can already be proposed. This includes strong connections to the *Travel Planner* scenario and calendar and geographical reasoning (WG A1).

4.2 Use Cases: XML

4.2.1 The Data

Consider the XML instances as below.

Participants' nodes. Participants' nodes have a simple structure, describing ther "technical data" and one entry for each researcher, e.g. `goettingen.xml`:

```
<participant>
 <name>Georg-August-Universitaet Goettingen</name>
 <shortname>Goettingen</shortname>
 <country>Germany</country>
 <representative>Wolfgang May</representative>
 <group>
  <name>Databases and Information Systems</name>
  <funding months="1-18" currency="Euro">12345.00</funding>
  <expenses>
    <expense reason="Kickoff Meeting">1234</expense>
    <expense reason="I5 Meeting 12-2004">5678</expense>
  </expenses>
  <person type="researcher" id="WM">
   <name>
    <givenname>Wolfgang</givenname>
    <familyname>May</familyname>
   </name>
   <title>Prof.</title>
   <title>Dr.</title>
   <title>Dipl.-Inform.</title>
   <contact>
    <email>may@informatik.uni-goettingen.de</email>
    <phone>0551 39 14412</phone>
   </contact>
   <participation role="deputy coordinator">I5</participation>
   <participation>I4</participation>
   <participation>A3</participation>
   <crossreader>I4-D1</crossreader>
   <crossreader>I4-D2</crossreader>
   <crossreader>A3-D2</crossreader>
  </person>
  <person type="researcher" id="EB">
   <name>
    <givenname>Erik</givenname>
    <familyname>Behrends</familyname>
   </name>
   <title>Dipl.-Inf.</title>
   <contact>
    <email>behrends@informatik.uni-goettingen.de</email>
    <phone>0551 39 14412</phone>
   </contact>
   <participation>I5</participation>
```

```
  </person>
 </group>
</participant>
```

Working Group Nodes. The Working Group nodes combine data of all participants' nodes as far as they concern the WG, and additionally contains WG information and materials (including documents in pdf format).

```
<workinggroup>
  <name>Evolution and Reactivity</name>
  <ID>I5</ID>
  <coordinator>Jose Alferes</coordinator>
  <assistant>Alexandre Pinto</assistant>
  <deputycoordinator>Wolfgang May</deputycoordinator>
  ⋮
  people
  ⋮
  <deliverable status="delivered">
    <number>I5-D1</number>
    <title>State of the Art</title>
    <crossreader>Thomas Eiter</crossreader>
    <crossreader>Gerd Wagner</crossreader>
    <date>Sept-01-2004</date>
    <availableat format="pdf">http:...</availableat>
  </deliverable>
  <deliverable status="working">
    <number>I5-D2</number>
    <title>Use Cases</title>
    <crossreader>Nicola Henze</crossreader>
    <crossreader>Michael Schroeder</crossreader>
  </deliverable>
</workinggroup>
```

Note that names (i.e., cross-readers and coordinators) are stored as text. For global use, a function is required that matches the detailed format of names as given in the person entries with e.g. these entries, authors of publications etc. Since all entries concerning persons are communicated with the project office, *policies* can be defined by rules that check if a matching is possible, or a wrong name is used (e.g. "Alexander" instead of "Alexandre"). Here, also actions can be taken by the central node to propagate the standard writings (as given in the participant's node) of names (e.g., adapting cross-reader entries). From the WG data, also WG HTML pages can be generated.

Project Node. The Central Project Node combines data of all participants' and working group nodes, and additionally contains project information and materials (including documents in pdf format).

```
<project>
 <workinggroup>
  <name>Evolution and Reactivity</name>
  <ID>I5</ID>
  <coordinator>Jose Alferes</coordinator>
  <assistant>Alexandre Pinto</assistant>
  <deputycoordinator>Wolfgang May</deputycoordinator>
```

```
  <deliverable status="delivered">
    <number>I5-D1</number>
    <title>State of the Art</title>
    <crossreader>Thomas Eiter</crossreader>
    <crossreader>Gerd Wagner</crossreader>
    <date>Sept-01-2004</date>
    <availableat format="pdf">http:...</availableat>
  </deliverable>
  <deliverable status="working">
    <number>I5-D2</number>
    <title>Use Cases</title>
    <crossreader>Nicola Henze</crossreader>
    <crossreader>Michael Schroeder</crossreader>
  </deliverable>
 </workinggroup>
 :
 <participant>
  <name>Georg-August-Universitaet Goettingen</name>
  <shortname>Goettingen</shortname>
  <country>Germany</country>
  <representative>Wolfgang May</representative>
  <url>http://dbis.informatik.uni-goettingen.de/...</url>
  <group>
   <name>databases and Information Systems</name>
   <funding months="1-18" currency="Euro">12345.00</funding>
   <expenses>
    <expense reason="Kickoff Meeting">1234</expense>
    <expense reason="I5 Meeting 12-2004">5678</expense>
    :
   </expenses>
   <person type="researcher" id="WM">
    <name>
     <givenname>Wolfgang</givenname>
     <familyname>May</familyname>
    </name>
    <title>Prof.</title>
    <title>Dr.</title>
    <title>Dipl.-Inform.</title>
    <contact>
     <email>may@informatik.uni-goettingen.de</email>
     <phone>0551 39 14412</phone>
    </contact>
   </person>
   <person type="researcher" id="EB">
    <name>
     <givenname>Erik</givenname>
     <familyname>Behrends</familyname>
    </name>
    <title>Dipl.-Inf.</title>
    <contact>
     <email>behrends@informatik.uni-goettingen.de</email>
     <phone>0551 39 14412</phone>
    </contact>
   </person>
  </group>
 </participant>
 :
</project>
```

Globally, there is a lot of redundancy, e.g. contact data of persons are stored at the participant's node, at every WG node where the person is involved, and at the central node. Each node is able to provide all relevant information for its "owner" as a standalone node.

Note that all syntax in this section is not normative, but only intended as a base for discussion:

- the event language must contain a set of connectives that covers usual event algebra connectives (especially, free variables and cumulative events)
- variables can be bound anywhere in the rule (e.g., after inside the event part, or after evaluating the condition part),
- the action language must contain connectives e.g. similar to those of Concurrent Transaction Logic [BK94, BK95]

4.2.2 Maintenance Use Cases

Use Case 4.2.1 (Changing Phone Number) *Phone numbers (or any other contact details) are updated at the participants' nodes. The updates have to be propagated to the WG nodes and to the central node.*

- *by a trigger:*

  ```
  ON UPDATE OF phone AS $phone
  LET $person := $phone/ancestor::person
  ... do something ...
  ```

  (analogously for `INSERT OF phone`)
  where `do_something` can be of several forms:

  - *explicit remote updates (in case a language allows this), or*
  - *send XUpdate messages to the WG node(s) where the person participates and the central node.*

- *by a local ECA rule with an explicit remote update*

```
<event>
  <evt:atomic>
    <change-of select="phone">
      <variable name="phone" select="."/>
      <variable name="person" select="$phone/ancestor::person"/>
    </change-of>
  </evt:atomic>
</event>
<action language="XQuery+Updates">
  update http://...
  set //person[matches(name,$person/name)]/phone := $phone
</action>
```

Note that `<change-of select="phone">` *includes deletion, insertion and modification of a phone number.*

- *by a local ECA rule that sends an XUpdate message*

```
<event>
  // same as above
</event>
<action>
 <sendmessage receiver="http://...">
  <msg:contents language="XUpdate">
   <xu:modifications version="1.0"
                     xmlns:xu="http://www.xmldb.org/xupdate">
     <xu:update select='{//person[matches(name,$person/name)]/phone}'>
       $phone
     </xu:update>
   </xu:modifications>
  </msg:contents>
 </sendmessage>
</action>
```

- *by an ECA rule at the WG node that detects remote events and then executes a local update:*

```
<event>
  <evt:atomic>
    <change-of select="http://...//person/phone">
      <variable name="phone" select="."/>
      <variable name="person" select="$phone/ancestor::person"/>
    </change-of>
  </evt:atomic>
</event>
<action language="XQuery+Updates">
  update
  set //person[matches(name,$person/name)]/phone := $phone
</action>
```

Use Case 4.2.2 (Association with Working Group) *The associations of researchers with Working Groups can be changed by the participants' nodes, by the WG nodes, and by the central node. In all cases, suitable propagation is required. This can again be done by triggers that react directly on the update, or by event detection.*

Below, non-local rules are given as ECA rules (note that upon the event, it is first checked in the condition whether the new information is already stored in the other nodes – which would be the case if the person has first been inserted in the WG node and then propagated downwards to the participant's node).

- *ECA rule that detects a new participation entry at a participant's page and propagates the change to the WG node, by means of explicit updates:*

```
<event>
  <evt:atomic>
    <change-of select="http://goe/rewersedata.xml//person/participation">
      <variable name="participation" select="."/>
      <variable name="person" select="$participation/ancestor::person"/>
      <variable name="WGUrl">
        // compute $WGUrl := url of the WG node
      </variable>
    </change-of>
  </evt:atomic>
</event>
<condition language="XPath">
   not $WGUrl//person[matches(name,$person/name)]
</condition>
<action>
 <act:atomic language="XQuery+Updates">
  update $WGUrl
  // possibly strip person to contain only WG-relevant information
  insert $person into $WGUrl//workinggroup
 </act:atomic>
</action>
```

Note that the rule must be given for each participant node, or a global rule must be created that reacts on the change at any of the participant nodes (see below).

- *analogous rule, with the action updating the project node,*
- *or, instead of the two above, one rule that updates both the WG node and the project node. In this case, the condition part is empty and the action part contains two* conditional *actions:*

```
<event> as above </event> <action>
 <act:conj>
  <act:cond test="not $WGUrl//person[matches(name,$person/name)]">
    <act:atomic language="XQuery+Updates">
     update $WGUrl
     // possibly strip person to contain only WG-relevant information
     insert $person into //workinggroup
    </act:atomic>
  </act:cond>
  <act:cond test="not $ProjUrl//person[matches(name,$person/name)]
                      //participation=$participation">
    <act:atomic language="XQuery+Updates">
     update $ProjUrl
```

```
    let $person2:=//person[matches(name,$person/name)]
    insert <participation>$participation<participation>
      into $person2
   </act:atomic>
  </act:cond>
 </action>
```

- *As stated above, such a local rule must be given for each participant node (push-propagation of changes). Alternatively, a global rule can be used that reacts on the change at any of the participant nodes. Here, the "event" is much more complicated to express since all participant nodes have to be considered. One possibility is a large disjunctive event "update of .. at url_1 or url_2 or ... or url_n". Since this set of urls can change over time, it is useful to define it by a query against the database.*

  Then, assuming an event language that allows for existential quantification, the following rule can be formulated:

```
<event>
  <evt:forany>
   <variable name="url" select="//participant/url"/>
   <!-- bound to the set of all urls of current participants -->
   <evt:atomic>
    <change-of select="$url//person/participation">
      <variable name="participation" select="."/>
      <variable name="person" select="$participation/ancestor::person"/>
    </change-of>
   </evt:atomic>
  </evt:forany>
</event>
<action>
 <act:atomic language="XQuery+Updates">
  insert $person into //workinggroup
 </act:atomic>
</action>
```

  Note that event detection must adapt to possible changes of participants urls (possibly by another reactive rule).

- *Another possibility is to constrain the urls in the condition part:*

```
<event>
  <evt:atomic>
    <change-of select="person/participation">
      <variable name="participation" select="."/>
      <variable name="person" select="$participation/ancestor::person"/>
      <variable name="url" select="... url where the change happens..."/>
    </change-of>
```

```
  </evt:atomic>
</event>
<condition>
  $url in //participant/url
</condition>
<action>
 <act:atomic language="XQuery+Updates">
  insert $person into //workinggroup
 </act:atomic>
</action>
```

Note that whereas the event part alone is hard to evaluate since all Web nodes have to be monitored, whereas the condition part actually specifies a finite (and small) set of nodes that are relevant. Thus, remote *event detection in general requires additional reasoning.*

Use Case 4.2.3 (Cross-Reader) *Cross-Readers for deliverables are inserted by the WG coordinator in the WG node. The information has to be propagated to the central node and to the cross-reader's participant's node. Here, again, triggers, local ECA rules or global ECA rules can be used.*

Use Case 4.2.4 (Deliverable) *If a deliverable is published, this is done either by a WG node or by the project node. In both cases, some communication has to be done, and possibly further actions at the project node should be taken automatically.*

The above use cases dealt with the maintenance of the basic data of the scenario. In the following sections, use cases that support specific tasks in this scenario are described.

4.2.3 Dissemination Use Cases

The knowledge about the project structure and the people is intended to be used for disseminating information in the project, e.g., to make appointments for meetings, and to keep to-do-lists and deadlines.
For these tasks, there are two different approaches:

Centralized Portal: The "Portal" provides information about everything in the project. Since the portal is personalized (see Section 4.2.4), each member of the project can have his or her personal view that e.g. lists all upcoming relevant events and deadlines.

On the other hand, if a person then wants to have an overview of what he has to do "today", he has to visit the project portal, the local university portal (e.g. for faculty meetings), possibly other project portals he is involved in, and his own schedule and duties (e.g., reviews).

Decentralized Information: Here, the information is sent (as XML or RDF packets) to the local nodes of the persons. Here, the participant's node is seen as the interface to the persons local environments. E.g., personal time planners can more easily be scripted to contact this node (and the `person` subtree or resource) to receive project-related information.

Thus, we again vote for a redundant approach which implements both a centralized portal and also local availability of all relevant information in the participants' nodes.

We consider both the "push" variant where nodes inform others about relevant issues, and the "pull" variant where nodes have active rules that react on *external* events.

Use Case 4.2.5 (Meeting – Push) *If a meeting is fixed, it is inserted either by the central node or by a WG node (e.g., a meeting of I4 and I5, initiated by I4). Then, it must be communicated (together with e.g. a Web page where the materials can be found) to appropriate other nodes.*

For this, an ontology of meeting information must be developed to characterize the meeting. An entry could e.g. look as follows.

```
<meeting>
  <title>I4/I5 meeting summer 04</title>
  <place>Munich</place>
  <date>7./8.6.2004</date>
   // note: calendar reasoning is done in A1
  <audience>
    a query/description that results in all names of person
    that are potentially interested, e.g., all members of I4/I5
    and people from other WGs who cooperate in some topics.
  </audience>
  <webpage href="www.rewerse.net/.../"/>
</meeting>
```

If a meeting is inserted into a node, appropriate rules must be there to evaluate and propagate this information. The insertion of meeting data in the project node e.g. triggers the following rule that disseminates the meeting information towards the concerned WG nodes and participant's nodes:

```
<event>
  <evt:atomic>
    <insert-of select="meeting">
      <variable name="meeting" select="."/>
    </insert-of>
  </evt:atomic>
</event>
<action language="XQuery+Updates">
  let $wgs := evaluate($meeting/audience)//wg
  let $persons := evaluate($meeting/audience)//person
  // send $meeting to $wgs nodes
  // send $meeting to $persons' participant nodes
</action>
```

The WG nodes must have a similar rule for handling incoming meeting information.

Note that a meeting can be propagated to a participant's node via several WG nodes if the participant belongs to several WGs. Similarly, it can be propagated to participants' nodes both

directly from the central node and from WG node(s). Rules have to check if information has already arrived by another way.

Use Case 4.2.6 (Meeting – Pull) *In the "pull" case, the WG nodes have a rule that reacts on insertions of meetings on the central node:*

```
<event>
 <evt:atomic>
  <insert-of select="http://centralnode//meeting">
   <variable name="meeting" select="."/>
  </insert-of>
 </evt:atomic>
</event>
<condition language="XQuery">
  evaluate($meeting/audience)//wg = $mywg
</condition>
<action language="XQuery+Updates">
  insert $meeting into //workinggroup
</action>
```

In the same way, deadlines are inserted by the Project Office into the central node and are propagated to the WGs and persons.

Use Case 4.2.7 (Progress Reports) *The deadline for the progress report is inserted into the central node, and then communicated to the WGs nodes. From there, the persons are called to send input (by mail, probably 10 days before the deadline), and the coordinator is called to produce the report. The coordinator then puts the report in the WG node. An active rule then publishes the report on the WGs Web page, and removes the* `deadline` *entry from the WG node and from the coordinator's* `person` *entry in the participant's node.*

Depending on "push" or "pull" strategy, (i) the WG node sends the report to the central node, or (ii) the central node reacts on the remote event on the WG node:

- *sending by WG node: similar to above.*
- *If the reports are collected by the central node, similar to the first "remote" rule given in Use-Case 4.2.2, this rule must react on remote events on any of the working group nodes. Here, we hardcode this as a disjunctive event:*

  ```
  <event>
   <evt:disj>
    <evt:atomic>
     <insert-of select="http://url-of-wg-i1//progressreport">
       <variable name="report" select="."/>
     </insert-of>
    </evt:atomic>
    :
    <evt:atomic>
  ```

```
    <insert-of select="http://url-of-wg-i5//progressreport">
      <variable name="report" select="."/>
    </insert-of>
   </evt:atomic>
   :
  </evt:disj>
 </event>
 <action language="XQuery+Updates">
   insert $report into //documents
 </action>
```

Another rule can e.g. notify the project office when all *progress reports (of a given period) have been received. Now, we have a* conjunction *of events.*

- *This can be expressed as a* conjunctive event*:*

```
<event>
 <evt:conj>
  <evt:atomic>
   <insert-of select="http://url-of-wg-i1//progressreport[@nr='1-2005']"/>
  </evt:atomic>
  :
  <evt:atomic>
   <insert-of select="http://url-of-wg-i5//progressreport[@nr='1-2005']"/>
  </evt:atomic>
  :
 </evt:conj>
</event>
<action>
  // send message to project office
</action>
```

- *... or by* universal quantification*:*

```
<event>
 <evt:forall>
  <variable name="url" select="//WG/url"/>
  <!-- bound to the set of all urls of WGs -->
  <evt:atomic>
   <insert-of select="$url//progressreport[nr='1-2005']">
   </insert-of>
  </evt:atomic>
 </evt:forall>
</event>
<action>
```

```
  // send message to project office
</action>
```

So far, still only data-driven communication about established facts takes place. In Section 4.2.5, we discuss use-cases that e.g. deal with making appointments and actually preparing meetings.

4.2.4 Personalization Use Cases: Local Evolution

Here, primarily two kinds of events are considered:

- personalization-specific events that are raised by interaction of the user,
- events in the database; e.g., a person becoming a coordinator, or adding a new research topic to a person's entry in the `person` entry in a participant's node.

Since the details of use cases depend heavily on the personalization ontology, examples cannot be very concrete here.

Use Case 4.2.8 (Coordinator) *When a person becomes a coordinator, his personal view of the project changes, e.g., deadlines of the WG, project meetings etc. become more relevant.*

In a latter stage, also external events on the Web that give hints to the personalization reasoner can be taken into consideration.

4.2.5 Reasoning and Evolution Use Cases

Use Case 4.2.9 (Planning a Meeting) *For planning a meeting, a request is inserted in the central node or in a WG node.*

```
<request-for>
 <meeting>
  <title>I4/I5 meeting summer 05</title>
  <date>between 15.5.2005 and 1.7.2005</date>
  <audience>
    a query/description that results in all names of persons
    that are potentially interested, e.g., all members of I4/I5
    and people from other WGs who cooperate in some topics.
  </audience>
  <webpage href="www.rewerse.net/.../"/>
 </meeting>
</request-for>
```

The request is propagated to the WGs and participants in the same way as in Section 4.2.3 above. All requests are also propagated to the central portal where an interface is prepared where the users indicate whether they plan to attend and enter preferable, impossible dates etc. (even the place can be negotiated?). The data is collected and processed by the central portal (using local active rules for collecting and reasoning; here also calendar reasoning has to be applied, e.g., concerning public holidays and fairs at meeting places). Additional rules are used

for sending reminders to people who do not answer. The organizer of the event then uses the portal for fixing date and place, and transforming the request into a fixed meeting entry. Then, travels are planned and funding is organized.

Use Case 4.2.10 (Travel Planning) *After fixing a meeting, the central node can also act as a travel agency and search for flights and train connections for people who will attend the meeting. This issue also provides a link to the "Travel Planning" scenario.*

A rule is needed that reacts on messages when people confirm their interest in participation in a meeting. With a suitable planning, it is e.g. possible to schedule people from neighboring places for the same flights/trains.

Use Case 4.2.11 (Funding) *If a member of a group should participate in a WG meeting, and the participant's funding is already spent for this period, organize additional funding by WG money.*
This can be done by a (global) ECA rule:

```
<event>
  <evt:atomic>
   <insert-of select="meeting">
    <variable name="meeting" select="."/>
    <variable name="persons" select="eval($meeting/persons)"/>
   </insert-of>
  </evt:atomic>
</event>
<condition>
  // for each person, check if funding is available
  // bind variable <nonfunded> to persons without funding
</condition>
<action>
  // specification of appropriate actions to be taken
</action>
```

Since the portal is also used to present the project and its participants on the Web, it should also inform about the dissemination of results of the project throughout the world.

Use Case 4.2.12 (External References to Rewerse Outcomes) *In case that some results of the project (courses, case-studies, samples) are used or referenced somewhere else, such information can be added to the presentation*

```
 <course>
   <title>Advanced Topics of Semantic Web Reasoning<title>
   <author>...</author>
   <cited-in href="http://..."/>
 </course>
```

An interesting issue here is that events like "University at XY uses course materials" have actually to be detected on the (Semantic) Web (e.g., via google). Thus, event detection in this general case means to know how to map the description of an event to queries against the (Semantic) Web.

4.2.6 Composite Events Use Cases

Basic Event Combinators. Composite events are usually defined by Event Algebras (see *Deliverable I5-D1* [ABB+04]). Common operations are sequential composition, alternatives/disjunctive events, iteration (so far, similar to regular expressions over ground atomic events).
In the above use-cases, we already identified disjunctive and conjunctive events (in Use Case 4.2.7), and that quantification is often needed as an extension of disjunction and conjunction:

- existential quantification in Use Case 4.2.2,
- universal quantification in Use Case 4.2.7,

So far, the constructs are similar to those from predicate logics.

Negated Events. Following the connectives from predicate logics, what about negation? When considering event algebras for reactive rules on the Web, "negative events" have to be related to a time interval where an event does not occur. Often this is used in the form of "until", and several ternary connectives in the style of "when E_1 happened, and then E_3 but not E_2 in between, then do something" have been proposed. E.g. the event algebras in [CKAK94, Sin95]. support such notions.

Use Case 4.2.13 (Progress Report Late) *For each progress report, a deadline is specified when it must arrive, e.g.,*

```
<todo type="progressreport" number="1-2005" deadline="28.2.2005"/>
```

Then, there is a rule that states that if for any WG, the progress report has not been checked in until noon at the day of the deadline, a message is sent to the WG's coordinator:

```
<event>
 <evt:forany>
  <variable name="url" select="//WG/url"/>
  <evt:seq>
   <evt:atomic>
    <insert-of select="//todo[@type='progressreport']/">
     <variable name="number" select="./@number"/>
     <variable name="deadline" select="./@deadline"/>
    </insert-of>
   </evt:atomic>
   <evt:not>
    <evt:atomic>
     <insert-of select="$url//progressreport[@number=$number]"/>
    </evt:atomic>
```

```
    </evt:not>
    <evt:temporal test="time=$deadline,12:00"/>
   </evt:seq>
  <evt:forany>
 </event>
 <action>...</action>
```

Note that the use of negation raises safety requirements on variables much in the same way as it happens in Logic Programming. Negation has then to be further investigated for composite events (possibly only allowed for some of them?)

Cumulative Events. Another important kind of events are "cumulative events" where series of (similar) events are accumulated into one, collecting data. Also, "aperiodic" and "cumulative aperiodic" events as in [CKAK94] should be considered.

We consider the following task: the project office makes a poll where an answer is needed from each WG coordinator. Then, there are different ways how to deal with it. Simple rules apply for making the poll known to the relevant persons (push to the participants' nodes or pull by them, or a rule at the central node that simply sends a mail to the people). Incoming responses are stored in the database as contents of the poll element; they are evaluated after the deadline.

```
<poll identifier="...">
  <persons>
     a query/description that results in all names of WG coordinators.
  </persons>
  <question>
     some text
  </question>
  <deadline>28.2.2005</deadline>
  <answer>
   <person>John Doe</person><text>yes</text>
  </answer>
  :
</poll>
```

Use Case 4.2.14 (Polls: Basic Rules) *Polls are evaluated as follows:*

1. *when a poll entry is inserted, evaluate the set of recipients and send them a mail,*

```
<event>
 <evt:atomic>
  <insert-of select="poll">
   <variable name="poll" select="."/>
  </insert-of>
 </evt:atomic>
</event>
```

```
<action>
   // send mail containing poll/@id to all persons in $poll/persons
</action>
```

2. *when a response comes in, store it in the database,*

```
<event>
 <evt:atomic>
  <incoming_message>
    <variable name="message" select="."/>
  </incoming_message>
 <evt:atomic>
</event>
<condition language=XQuery>
  // check if incoming message contains an answer to an open poll
  // with id $poll_id
</condition>
<variable name="sender" select="$message/@sender"/>
<variable name="poll_id" select="$message/contents/@reference_to"/>
<variable name="answer" select="$message/contents/text"/>
<action language="XUpdate">
 <xu:modifications version="1.0"
                  xmlns:xu="http://www.xmldb.org/xupdate">
  <xu:append select='{//poll[id=$poll_id]}'>
   <answer>
    <person>{$sender}</person><text>{$answer}</text>
   </answer>
  </xu:append>
 </xu:modifications>
</action>
```

3. *when the deadline is over (note that this is actually a sequential event: inserting a poll and the temporal event of the deadline), query the database and inform the project office about the result; in case that answers are missing, send another mail to these persons.*

```
<event>
 <evt:seq>
  <evt:atomic>
   <insert-of select="poll">
    <variable name="poll" select="."/>
   </insert-of>
  </evt:atomic>
  <evt:temporal test="time=$poll/deadline"/>
 </evt:seq>
</event>
<action>
```

```
  <variable name="receivers" select="evaluate($poll/persons)"/>
  <variable name="answered"
    select="//poll[id={$poll/id}]//answer/person"/>
  <act:conj>
   <act:atomic> send message with answers to project office</act:atomic>
   <act:forall>
    <variable name="missing"
      select="receivers - answered"/>
    <act:atomic> send reminder to $missing</act:atomic>
   </act:forall>
  </act:conj>
 </action>
```

Note that this also involves a complex action, namely the one that sends reminders to all those whose answer is missing.

Alternative:

2' replace (2) by "when a response comes in, store it in the database, check if all persons answered, if yes, inform the project office about the result immediately".

A cumulative event is one that collects data from a sequence of events. Its specification must at least contain the following information:

- what events to collect,
- what data to collect from them,
- when the cumulative event is detected (i.e., when it "reports" and the event sequence can proceed).

A more detailed look shows that we need two "until"s: one that ends the cumulative part with the first occurrence of the reporting event, and one that ends with *each* subsequent occurrence of the subsequent event, until another condition occurs that finally stops the cumulative event. E.g.,

cumulative(detect=$e_1(x)$,report=e_2,refresh=$true/false$,until=e_3)

does the following until e_3 is detected: whenever e_2 is detected, report all x that have occurred (i) if $bool = false$: then since the beginning of the detection of the cumulative event, (ii) if $bool = true$: since the last report. In case that until is not set, until=report.

Use Case 4.2.15 (Polls: Cumulative Events) *The same activity can be carried out without actually storing answers, but handling them in a* cumulative *event (that collects the answers):*

- *when a poll entry is inserted, evaluate the set of recipients and send them a mail,*
- *if first, a poll is issued, and then a set of answers* answer(person,answer) *comes in, collect the set of persons and the set of answers until the deadline (atomic events: poll, answer, deadline). The overall detection of this composite event results in a small XML instance that contains the collected information.*

```
<event>
 <evt:seq>
  <evt:atomic>
   <insert-of select="poll">
    <variable name="poll" select="."/>
    <variable name="poll_id" select="create_id()"/>
    <variable name="deadline" select="$poll/deadline"/>
    <variable name="persons" select="eval($poll/persons)"/>
   </insert-of>
  </evt:atomic>
  <evt:cumulative>
   <variable name="answers"/> // variable that holds collected data
   <evt:detect>
    <evt:atomic>
     <incoming_message>
       // with reference to $poll_id
       <append variable="answers"> //possibly similar to XUpdate?
         <answer person="$message/sender" text="$message/text"/>
       </append>
     </incoming_message>
    <evt:atomic>
   </evt:detect>
   <evt:report>
    <evt:temporal test="time=$deadline"/>
   </evt:report>
   // there is no <until> part, thus <until> defaults to <report>
  </evt:cumulative>
 </evt:seq>
<event>
```

Note that detecting the above cumulative event "consumes" the deadline event and ends with it. This rule can again be refined such that it informs the project office already in case that all answers are complete before the deadline:

```
<event>
 <evt:seq>
  :
  <evt:cumulative>
   :
   <evt:report consume="false">
    <evt:disj>
     <evt:temporal test="time=$deadline"/>
     <evt:cond>all persons have answered</evt:cond>
    </evt:disj>
   </evt:report>
  </evt:cumulative>
```

```
  </evt:seq>
 <event>
```

It seems to be reasonable to be able to specify if report *and* until *should consume the event, or not.*

Conditions in Intermediate States When considering rules and information, also the use of conditions upon the states during detection of composite events have to be considered. Especially, pre- and postconditions are relevant (e.g., in a banking context "a money transfer that lead to a negative balance"). As long as only simple rules like triggers are considered, such conditions can always be put in the `WHEN` part, evaluating `OLD` and `NEW` values. When composite events or conditions that are also concerned with information from other nodes are considered, there are two possibilities:

- bind variables to values in the atomic events, and state conditions on these variables in the condition part, or
- allow conditions in the event algebra. For this, atomic events can be extended with `precondition` and `postcondition` elements.

We do not present here a use-case for conditions in intermediate states, which we find better suited for scenarios with transactions like banking or travel planning. However, to make it clear, we propose the syntax:

```
<event>
 <evt:seq>
  <evt:atomic>
   <evt:precondition language="XQuery">
      // a boolean XQuery query;
      // here, also variables can be bound
   </evt:precondition>
   <insert-of ... />
   <evt:postcondition language="XQuery">
      // a boolean XQuery query;
      // here, also variables can be bound
   </evt:postcondition>
  </evt:atomic>
  :
 </evt:seq>
</event>
```

4.2.7 Composite Actions Use Cases

Here the usual connectives for *defining* transactions have to be provided:

- sequential composition,
- alternative composition, based on branching,

- parallel composition,
- iteration until a given condition is satisfied (including cumulative variables that are e.g. used for final actions).

For a more high-level specification of evolution and reactivity, the following constructs should be expressible:

- implicit existential alternatives: "exists an x of a given set for which ... must be executed". (as a rule that specifies a constraint – the actual activity planning must then use additional reasoning which x should do the task)
- universal quantification over a set of items; here it should be possible to specify "ordered" or "unordered",

Use Case 4.2.16 (Meeting: Dinner/Existential Quantification) *If a WG meeting is scheduled and participants have registered, some restaurant (with enough space and opened in this evening etc.) must be booked for the dinner (the set of restaurants can be specified as a query against the Semantic Web).*

Use Case 4.2.17 (Collecting Registrations and Booking Hotels) *The following use case illustrates cumulative events and cumulative actions:*
Event: the deadline of the summer school, here as "if the registrations for the summer school until a deadline are collected".
Action: "Starting with the largest hotel in the city book as much rooms as possible, until enough rooms for all participants are reserved". At the end return a table of hotels and number of booked rooms.

Furthermore, when defining transactions, these probably do also not simply consist of actions, but also of waiting, evaluating conditions etc. (which is implicitly already present in alternative composition). Here, the following features will turn out to be useful:

- updates of actions rules, possibly with negation of action (names) allowing for e.g. forbidding an action for some time as a reaction on some event (e.g., when defining and applying policies), or delaying the addition of an action rule
- evaluating queries and tests,
- temporal constraints: "... a must be executed before the deadline t".

Use Case 4.2.18 (Forbidding Actions until Event) *If once a restaurant has been booked for a dinner, and people complained about the quality, this restaurant must not be booked again until its owner changes.*
Note that this introduces events in the action part "not book it again until the event 'change of owner' *(which is an event somewhere on the (Semantic) Web) occurs". This ECA rule can be seen as one that as consequence has the negation of the action of booking dinner for that particular restaurant.*

Similarly, the "until" condition can be a query – which has then internally to be translated into an event, or a query that is evaluated in case the questionable action should be evaluated

Use Case 4.2.19 (Forbidding Actions until Condition) *If once a restaurant has been booked for a dinner, and people complained about the price, this restaurant must not be booked again until the price of a pizza is less than 10 Euro.*

Here, the constraint can be checked by waiting for the event "pizza price changed", or by querying the pizza price when a restaurant is to be chosen.

Use Case 4.2.20 (Delaying addition of Rules) *After inviting a participant for cross-reading a given deliverable of the project, a rule should be added stating that after (and if) that participant accepts the invitation, any incoming information regarding the deliverable should be communicated to the that participant.*

Here, the addition of an ECA rule, triggered by incoming information about a particular deliverable and communicating that information to the participant, is to be delayed until the participant accepts the invitation

4.3 Further issues

Requirements. As usual, detection of composite events will be based on internal algorithms (graph-based, residuation, automata – which is essentailly the same). Steps of these internal algorithms will be based on atomic events. Thus, for all atomic events, it must be possible to define a trigger that detects the atomic event and supplies values of variables and for checking pre- and postconditions. Possibly, the event algebra processor has then to state other remote queries to evaluate distributed pre- and postconditions.

Since the constructs discussed above contain variables, quantifiers, cumulative events etc., the event detection mechanism must be able to hold actual data (prospectively in XML). On the other hand, with this, there is no need for a history database.

Related Work. In addition to a considerable amount of work on event algebras [CKAK94, Sin95] and process algebras (CCS [Mil83], CSP [Hoa85]), which have especially to be considered when defining the action parts), especially the Transaction Logic approach [BK94] has to be mentioned. It provides a comprehensive clear and concise syntax and model-theoretic semantics for temporal relationships covering states, events and actions. Its logic-programming style rule semantics can be interpreted bottom-up, where it corresponds to ECA rules, or top-down where it can be used for planning.

Use case 4.2.15 can be defined in a Transaction Logic style not as an ECA rule but as a constraint. Using quantifiers, this could e.g. look like

$$
\begin{array}{l}
\mathsf{poll}(\mathit{text},\mathit{persons})\ \text{:-}\\
\quad \forall X \in \$\mathit{persons}\text{:}\\
\quad\quad ((t{=}\mathit{deadline} \otimes \mathsf{receive\_answer}(X,\mathit{poll\_id},\mathit{answer})) \vee\\
\quad\quad\quad ((\mathsf{not}\ (t{=}\mathit{deadline} \otimes \mathsf{receive\_answer}(X,\mathit{poll\_id},\mathit{answer})))\\
\quad\quad\quad\quad \wedge\ (\ \mathsf{receive\_answer}(X,\mathit{poll\_id},\mathit{answer}) \otimes\\
\quad\quad\quad\quad\quad\quad \mathsf{send\_question}(X,\ \mathit{poll\_id},\ \mathit{text}) \otimes t{=}\mathit{deadline})))\\
\quad\quad \otimes\ \forall X \in \$\mathsf{persons}\text{:}\ \mathsf{send\_question}(X,\ \mathit{poll\_id},\ \mathit{text})\\
\quad\quad \otimes\ (\mathit{poll\_id} := \mathsf{insert\_poll}(\mathit{text},\mathit{persons}) \wedge \$\mathsf{persons} := \mathsf{evaluate}(\mathit{persons}))
\end{array}
$$

meaning, that $\mathsf{poll}(\mathit{text},\mathit{persons})$ is specified to be executed by the above rule body. Note that either a temporal implication is needed, or a possibility to bind variables (safety requirement: variable must be bound in an earlier state). Additionally, "negated" events are needed.

For negated action, and updates of ECA rule, the work on update languages in logic programming [APPP02, EFST01, Lei03] is also relevant.

RDF/OWL For the RDF level, the scenario and use-cases are the same, mapped to RDF/OWL.

1. rules over RDF data,
2. atomic events described in RDF (i.e., mainly in form of changes of tripels),
3. composite events in RDF,
4. ECA rules in RDF.

All the above cases can be seen as just "lifting" the XML case to RDF/OWL. For RDF/OWL, a specification of atomic events and updates must be based on an RDF query language. Thus, concrete samples can only be given when there is data and an RDF query language. From the ECA point of view, expressing an event algebra as terms, in XML, or in RDF is just syntactic sugar plus RDF/OWL-inherent reasoning.

As lots of previous work shows (cf. [ABB$^+$04]), unlike e.g. *the* Boolean Algebra for logics, there is not a standard event algebra. Indeed, a bunch of different event algebras with different event combinators have been defined.

When expressing composite events in an OWL event ontology, this should be as much general as possible, trying to cover *all* event algebras. As such, it should contain a set of basic combinators, and a way to express derived combinators. Event detection then requires to map given events to appropriate actual event algebras, and to use their event detection mechanisms for actually detecting events.

Chapter 5

Organising Travels Scenario

Planning and booking travels, staying up to date with changes in the plan (e.g. due cancellations of flights or overbookings of hotels), and (if necessary) adapting to such changes is a time consuming task. E-business on the Web has made planning and booking of travels, e.g. a researcher's trip to a conference, a lot easier. A researcher can now compare offers on the Web and book simply with his or her credit card. But still, this requires a lot of human interaction. Booking requires updates, posing a requirement for evolution of data on the Web.

However, reasoning-capable applications do not have the capability to detect situations requiring an adaption of the initial plan. The first step towards filling this gap consists in recognising the features that a system supporting reactive planning should have. It is our conviction that the best way of doing this is by looking at real life applications, i.e. by developing use cases for this application domain.

In this Chapter we develop a use case for evolution and reactivity on the Web, called *Organising Travels*. It is an application of Web-based reactive travel planning and support. In realising such an application two subtasks need to be considered : 1. initial planning and 2. reacting to happenings that influence the plan. These subtask are explained in Section 5.2 and 5.3, respectively. Note that the task of travel planning and support does not involve a single system. Instead, in order to save time and perfectly manage trips, several systems must cooperate to realise 1. and 2., e.g. flight and train schedules, passenger notification system, hotel reservation service.

5.1 Language issues

In order to exemplify arrangements that are needed for travelling and the adaption of the initial plan to changes, in this scenario we use the reactive language XChange [BFPS04, BP05], as the language is currently under development by members of the REWERSE's participant Munich. XChange is a high-level language for programming reactive behaviour on the Web, that builds upon the Web query language Xcerpt [SB04, BBSW03], developed in the REWERSE Working Group I4, "Reasoning-Aware Querying", and provides advanced, Web-specific capabilities, such as propagation of changes on the Web (*change*) and event-based communication between Web sites (*exchange*). Xcerpt is a pattern and rule-based language for querying Web contents (i.e. persistent data).

The language XChange supports the concept of transactions, which are needed when executing updates like reservations on the Web. An XChange transaction specification is a group of update specifications and/or explicit event specifications (expressing events that are constructed, raised, and sent as event messages) that are to be executed in an *all-or-nothing manner*. An XChange *update specification* is a (possibly incomplete) *pattern* for the data to be updated, augmented with the desired update operations. Besides the events mentioned in Chapter 2 considers *Transactional events* (transaction commit, transaction abort, transaction request), which are local events needed in order to support the concept of transactions in XChange.

An XChange program is located at one Web site and consists of one or more (re)active ECA rules of the form *Event query – Standard query – Transaction/Raised events*. Every occurrence of an event is queried using the *event query* (introduced by keyword `ON`). If an answer is found and the *standard query* (introduced by keyword `FROM`) has also an answer, then the action is executed (i.e. a transaction is executed – keyword `TRANSACTION`, or explicit events are raised and sent to one or more Web sites – keyword `RAISE`). There are two kinds of XChange rules: *event-raising rules* (i.e. the head of the rules specifies explicit events) and *transaction rules* (i.e. the head of the rules specifies transactions).

To this end, it is important to distinguish between event and standard web queries. By standard queries, we mean queries to *persistent data* (data of Web resources) and can be expressed using a query language like XQuery or Xcerpt. To query volatile data, *event queries* are used. Standard and event queries can be very similar. However, event queries are more likely to refer to time or event sequences. In fact, event queries need to be evaluated in an *incremental* manner, as data (events) that are queried are received in a stream-like manner and are not persistent. For every incoming event that might be relevant to a Web site and could contribute as a component to an event query instance specified in the rules of the Web site's reactive program(s), a partial *instantiation* of the involved event queries is realised. An instance of a specified composite event query is detected when instances for all specified component event queries have been detected.

Regarding which data can be updated, we follow the metaphor of *speech vs. written text*: *speech* for volatile (events) data, and *written text* for persistent data. Speech cannot be modified. If one has communicated some information in this way one can correct, complete, or invalidate what one has told – through further speech. In contrast, written text can be updated in the usual sense. Likewise, volatile data (events) is *not* updatable but persistent data (Web content) is updatable. To inform about, correct, complete, or invalidate former volatile data, new *event messages* (data containing information about events that have occurred) are communicated between Web sites. Since events are volatile, like speech, a Web site cannot pose event queries against events that have been received by another Web site. Otherwise, events would have to be made persistent – confusing the clear picture volatile vs. persistent data and thus potentially making programming more complicated.

Communication of events between the same or different Web sites is done by *Event messages*. An event message is a representation of the information related to an event that occurred. E.g. an event message contains at least the following information: `raising-time` (the time of the event manager of the Web site raising the event), `reception-time` (the time at which a site receives the event), `sender` (the URI of the site where the event has been raised), and `recipient` (the URI of the site where the event has been received). Thus, an event message

is an XML document having a root labelled e.g. `event` and the above described parameters as child elements.

5.2 Initial Planning

The *initial planning* subtask of organising trips consists of two phases:

1. gathering the information about transportation, overnight stays etc, required for the trip, and developing itineraries (a travel plan) based on this information, and
2. arranging (booking) the trip according to this plan.

5.2.1 Gather information and plan trip

Generally, when planning a trip one knows the location(s) he/she wants to visit and a time period for doing this. In most cases the time period is not fixed at time points level, i.e. one might return late in the evening or next day in the morning, but it is likely to be constrained by working hours or appointments. Thus, the gathering of information for planning a trip involves finding flights and train connections, finding suitable hotels, considering weather forecast, and/or planning entertainment.

Another issue that sometimes plays a role in planning a trip is the amount of money that one is willing to pay for all the arrangements. This presupposes searching for cheapest accommodation and transportation means that conforms one's expectation. Moreover, limited time discounts can also be taken into consideration. (Note that such discounts might not be persistent Web data. Instead they can be sent as notifications by a Web-based information service. In a sense, this is already common today: via e-mail newsletters many low-cost airlines offer short-term discounts.)

For realising systems that are able to accomplish this part of initial planning, the following are required:

- Web query languages with advanced reasoning capabilities, e.g. for searching for the cheapest suitable hotel;
- support for receiving volatile data, e.g. discount information;
- query and reasoning with volatile data.

5.2.2 Arranging the trip according to plan

Gathering all the needed information on e.g. flights, train connections, hotels, prices represents the premises for planning a trip. After settling for a plan by choosing transportation means, accommodation, and (perhaps) entertainment, one needs to arrange the trip according to the plan, i.e. book flights, reserve seats in trains, make hotel reservations, buy tickets.

Booking a flight on the Web, for example, does not necessarily entail a successful execution of the desired reservation. Possible reasons for failures include system down-times, network communication problems, or problems caused by concurrency and time-delays, e.g. if the last seat on a flight was sold while we were still planning. As a consequence, reservations that are "related" should be executed in an *all-or-nothing* manner. E.g. booking a hotel reservation

without a flight reservation is useless. Note that the order in which such reservations (that represent actually updates to data of some Web resources) are executed needs also to be taken into consideration. Usually one might want to realise the flight reservation before the hotel reservation. Under some circumstances, e.g. if there is a big trade fair in a city, so hotel beds sparse, other execution orders are also sensible.

Systems that could arrange a trip on the Web according to a given plan require:

- communication of data between systems that (in some sense) cooperate to arrange a trip;
- updates to persistent data (i.e. local and remote persistent data items);
- some support for transactions (i.e. for executing updates in an all-or-nothing manner).

5.2.3 Scenario

Mrs. Smith uses a travel organiser that plans her trips and reacts to happenings that could influence her schedule. One of the travel organiser's tasks is to plan Mrs. Smith's vacation in Italy. Mrs. Smith wants to visit Milan, Venice, Florence, and Rome. The trip should begin on 5th of March 2005 and end on 20th of March 2005.

1. Gathering of and reasoning with information

- Searching for suitable flights from Munich to Rome and back. Schedules and price tables of several airlines have to be queried and compared. Only flights departing on 5th of March 2005 and arriving on 20th of March 2005 before 21:30h are of interest. Moreover, the search is constrained by a price limit of EUR 400.
- Querying schedules and prices for train or flight connections for Rome – Florence – Milan – Venice – Rome. Like searching for flights between Munich and Rome, queries have time and price constraints.
- Querying hotel reservation services' data to find suitable hotels in the cities Mrs. Smith wants to visit. Suitable here refers to the quality of services (e.g. at least 2 stars), the prices (e.g. price per night should be cheaper than EUR 70), the time period for which single rooms are available for booking (e.g. Mrs. Smith wants to book a hotel in Rome from 5th of March 2005 until 8th of March 2005), and the location of the hotels (e.g. a quite area near to a metro station).
- Receiving notifications from different kind of information systems, like weather forecast services, or services announcing exhibitions. Reasoning with notifications' data and data of Web resources is needed for planning departures and arrivals but also for planning entertainment. For example,
  - if a notification is received informing that between 11th and 14th of March 2005 in Venice is going to rain, the vacation could be planned so as to leave for Venice on 15th of March 2005.
  - weather forecast information can be used together with exhibition notifications in order to plan visiting an exhibition on a rainy day.

Some use-case examples are given that intend to give flavours of how such a scenario can be implemented.

Example 5.1 *The following two XML documents are fragments of two larger documents representing a flight timetable and a hotel reservation offer. At* ***http://airline.com****:*

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<flights>
  <last-changes>2005-03-02</last-changes>
  <currency>EUR</currency>
  <flight>
    <number>AI2011</number>
    <from>Munich</from>
    <to>Rome</to>
    <date>2005-03-05</date>
    <departure-time>06:40</departure-time>
    <arrival-time>08:20</arrival-time>
    <class>economy</class>
    <price>93</price>
  </flight>
  <flight>
    <number>AI2021</number>
    <from>Munich</from>
    <to>Rome</to>
    <date>2005-03-05</date>
    <departure-time>09:10</departure-time>
    <arrival-time>10:45</arrival-time>
    <class>economy</class>
    <price>138</price>
  </flight>
</flights>
```

At ***http://hotels.net****:*

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<accomodation>
  <currency>EUR</currency>
  <hotels>
    <city>Rome</city>
    <country>Italy</country>
    <hotel>
      <name>Altavilla</name>
      <category>2 stars</category>
      <price-per-room>69</price-per-room>
      <phone>+39 1 88 8219 213</phone>
      <no-pets/>
    </hotel>
    <hotel>
      <name>Giardino d Europa</name>
      <category>3 stars</category>
      <price-per-room>70</price-per-room>
      <phone>+39 1 82 8156 135</phone>
    </hotel>
    <hotel>
      <name>Villa Florence</name>
      <category>3 stars</category>
      <price-per-room>95</price-per-room>
      <phone>+39 1 77 8123 414</phone>
```

```
    </hotel>
  </hotels>
</accomodation>
```

The gathering of information for planning travels can be realised e.g. by using the Web query language Xcerpt. As already mencioned, Xcerpt is a pattern and rule-based language for querying Web contents (i.e. persistent data). Xcerpt uses query *patterns* for querying Web contents, and construction *patterns* for constructing new data items. *Terms* are used for denoting query patterns (i.e. *query terms*), construction patterns (i.e. *construct terms*) and also for denoting data items of Web contents (i.e. *data terms*).

Example 5.2 *The following Xcerpt query term is used to query the data at* `http://airline.com` *about flights from Munich to Rome.*

```
in { resource { "http://airline.com" },
  flights {{ var F ⇝ flight {{
                       from {"Munich"}, to {"Rome"}  }}
       }}
  }
```

In the term syntax (used here as it is more readable as the Xcerpt's XML syntax), an ordered term specification *is denoted by square brackets [ ], an* unordered term specification *by curly brackets* {}.

Construct-query rules relate a construct term (introduced by the keyword `CONSTRUCT`*) to a query (introduced by the keyword* `FROM`*) consisting of* `AND` *and/or* `OR` *connected query terms. Queries or parts of a query may be further restricted by arithmetic constraints in a so-called condition box (introduced by the keyword* `where`*).*

Example 5.3 *The following Xcerpt rule gathers information about hotels in Rome that have a price limit.*

```
CONSTRUCT
  answer [ all var H ordered by var P ascending ]
FROM
  in { resource { "http://hotels.net" },
    accomodation {{
      hotels {{ city {"Rome"},
                var H ⇝ hotel {{
                          price-per-room { var P } }}
           }}
    }}
  } where var P < 80
END
```

An Xcerpt program consists of one or more rules. Xcerpt rules may be chained *to form complex query programs, i.e. rules may query the results of other rules. More on Xcerpt can be found in [Sch04] and at* `http://xcerpt.org`*.*

2. Arranging the trip

- Booking a flight from Munich to Rome and back, and a suitable hotel in Rome. (These reservations are updates to some Web resources that need to be executed in all-or-nothing manner, i.e. as a transaction.)

- Making train reservations and corresponding hotel reservations in order to visit also Florence, Milan, and Venice.

In order to exemplify arrangements that are needed for travelling and the adaption of the initial plan to changes (see Section 5.3) we use the reactive language XChange

Use Case 5.2.1 *The following XChange transaction rule is used to book a flight from Munich to Rome and back and a hotel in Rome:*

```
TRANSACTION
 and [
  update {
   in { resource {"http://airline.com/reservations"},
     reservations {{
       insert reservation { var F1, var F2, name {"Christina Smith"} }
      }}
     }
    },
  update {
   in { resource {"http://hotels.net/reservations"},
     reservations {{
       insert accomodation { var H,
                             name {"Christina Smith"},
                             from {"2005-03-05"}, until {"2005-03-07"}
                 }
      }}
     }
   }
 ]
FROM
 and {
  in { resource {"http://airline.com"},
   flights {{
     var F1 ⇝ flight {{
             from {"Munich"}, to {"Rome"},
             date {"2005-03-05"}
          }},
      var F2 ⇝ flight {{
             from {"Rome"}, to {"Munich"},
             date {"2005-03-20"}
          }}
    }}
  },
  answer [[ position 1 var H ⇝ hotel {{ }} ]]
 }
END
```

Note that the updates, i.e. the flight and hotel reservations, are to be both executed (specified by the keyword ***and****) and in the specified order (given by the usage of square brackets). Similar to the flight and hotel reservation in Example 4., the desired train reservations can be specified using the language XChange.*

5.3 Adapt Plan to Changes

5.3.1 Recognize changes affecting the plan

Events such as changing weather conditions, delays of flights or trains, and cancellations of flights can affect the arrangements made in the initial planning phase of organising a trip. Recognising such changes in very short time is the premise for adapting the initial plan, e.g. by booking another flight. Recognising changes involves communication of change notifications and detection of situations of interest.

Consider a personal travel organiser that plans trips for its owner and has also the capability to react to changes that might influence arranged plans. (A personal travel organiser is an example of a system for travel planning and support.)

There are two possible strategies for making the travel organiser aware of changes (events):

- pull strategy, i.e. the travel organiser periodically queries the data from remote Web sites (e.g. flights schedule) in order to determine whether simple changes of interest have occurred;
- push strategy, i.e. the Web-based information systems inform the travel organizer about changes that have taken place either locally or remote (case in which the systems have been notified about these changes).

Both strategies are useful. For propagating changes, a push strategy has several advantages over a strategy of periodically pulling: It allows faster reaction to events, as an event is communicated as soon as possible as opposed to a detection at the next periodical pull. It saves resources, both locally and on the network. Locally, a client interested in some change of Web data does not have to store the old Web page to detect differences (changes) from the new version. On the network, a push strategy can reduce network traffic, since communication only takes place when a change has happened, and only the changes in information have to be communicated.

The push strategy is used throughout the *Organising Trips* use case in order to notify information systems about events that have occurred. After receiving notifications about changes that might affect already made arrangements, the personal travel organiser needs to detect situations of interest, i.e. possibly time related combinations of simple happenings that might have an impact on already made plans.

Systems having the capability to recognize changes that might affect trip plans require:

- means for communicating notifications;
- event queries for detecting situations of interest;
- reasoning on events and persistent data.

5.3.2 React to changes

Of great importance in developing travel planning and support systems is the capability to automatically react to changes (to situations of interest). Based on the events that have occurred, on the detected situations of interest, actions need to be executed in order to adapt plans to these changes. Example of actions are reordering travel destinations, notifying affected friends and colleagues, finding and booking other flights (if necessary: booking overnight stays).

Systems having the capability to react to changes that affect already made plans require: reaction capabilities.

5.3.3 Scenario

Consider the scenario introduced in Section 5.2.3. After carrying out the tasks introduced in Section 5.2.3, the necessary arrangements for a vacation's initial plan have been made.

Mrs. Smith's travel organiser has the capability to detect changes that might affect the initial plan. Examples of such changes are:

1. The flight booked for Mrs. Smith from Rome to Munich on 20th of March 2005 has a delay. In such a case, the new arrival time and Mrs. Smith appointments need to be taken into consideration, before deciding how to react to such an event.
2. The flight booked for Mrs. Smith from Rome to Munich on 20th of March 2005 has been cancelled. In such a case, there are two possibilities:
   (a) the airline provides an accommodation for the night of 20th to 21st of March 2005;
   (b) the airline does not provide such an accommodation.

   The possibilities require different reactions. In the case of flight cancellation, the travel organiser could wait for a notification regarding accommodation only a fixed amount of time (e.g. it waits 2 hours from the reception time of the notification regarding the flight cancellation).
3. A train, for which Mrs. Smith has a reservation, has a delay.

For detecting changes like the ones presented above, the travel organiser needs to have the capability to:

- detect notifications (i.e. primitive events);
- detect time-related combinations of notifications (i.e. composite events). For example, event 2 followed by event 2b represents a time dependent conjunction of events.
- detect composite events that have occurred in a specified time interval (note that such a time interval could also be given as a duration, e.e. 2 hours);
- compare time notions (e.g. in case of event 1., the arrival time should be *before* 5:00h on 21st of March 2005).

Example 5.4 *Assume that a flight has been cancelled. The control point that has observed this event raises it and sends to* `http://airline.com` *the following event message (i.e. notification):*

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<xchange:event>
  <xchange:sender>
     control://controlpoint-A20
  </xchange:sender>
  <xchange:recipient>
     http://airline.com
```

```
  </xchange:recipient>
  <xchange:raising-time>
     2005-03-20T10:15:00
  </xchange:raising-time>
  <cancellation>
    <flight>
       <number>AI2021</number>
       <date>2005-03-20</date>
    </flight>
  </cancellation>
</xchange:event>
```

Note the use of the **xchange** *namespace for the keyword* **event** *and for the parameters of an XChange event message.*

Use Case 5.3.1 *The site* **http://airline.com** *has been told to notify Mrs. Smith's travel organiser of delays or cancellations of flights she travels with.*

```
RAISE
 xchange:event {
   xchange:recipient {"organiser://travelorganiser/Smith"},
   cancellation-notification { var F }
  }
ON
 xchange:event {{
  xchange:sender {"http://airline.com"},
  cancellation {{
    var F ⇝flight {{ number {"AI2021"},
                     date {"2005-03-20"} }}  }}
 }}
END
```

The use case examples are intended to give flavours of the XChange constructs and thus abstract away from a particular communication protocol. In the previous example `organiser` denotes a communication protocol (e.g. the protocol used by a mobile personalised organiser).

Use Case 5.3.2 *The travel organiser of Mrs. Smith uses the following rule: if the return flight of Mrs. Smith is cancelled then look for and book another suitable flight. The rule is specified in XChange as:*

```
TRANSACTION
 in { resource {"http://airline.com/reservations"},
   reservations {{
     insert reservation { var F, name {"Christina Smith"} }
     }}
 }
ON
 xchange:event {{
   xchange:sender {"http://airline.com"},
   cancellation-notification {{
     flight {{ number {"AI2021"}, var D ⇝date {"2005-03-20"} }}
    }}
 }}
FROM
 in { resource {"http://airline.com"},
   flights {{
     var F ⇝flight {{
```

```
            from {"Rome"}, to {"Munich"},
            var D, departure-time { var T }
          }}
    }}
  } where var T after "17:30"
END
```

One of the novelties introduced by the language XChange is the processing of *composite events*. To this aim, XChange offers *composite event queries*. An XChange *event query* may be *atomic*, i.e. one event query term that refers to a single event, or *composite*. XChange composite event queries are used for detecting e.g. conjunctions, disjunctions, temporal ordered conjunctions, negation of events in a finite time interval.

Use Case 5.3.3 *If no other suitable return flight is found and the airline does not provide an accommodation, then book for Mrs. Smith a cheap hotel and inform her secretary about the changes in her schedule:*

```
TRANSACTION
 and [
   in {
     resource {"http://hotels.net/reservations"},
     reservations {{
       insert reservation { var H, name {"Christina Smith"},
                from { var S }, until { var M ⇝"2005-03-20"} }
      }}
    },
   in {
     resource {"diary://diary/secretary"},
     diary {{ var M,
       news {{ insert my-hotel { phone { var Tel },
                remark {"My flight has been cancelled!},
                request{"Please cancel my appointments for" + var M}  }
        }}  }}
    }
  ]
ON
 and {
   xchange:event {{
      xchange:sender {"http://airline.com"},
      cancellation-notification {{
                flight {{
                   number {"AI2021"},
                   date { var S ⇝"2005-03-20"} }}
               }}
    }},
    without xchange:event {{
              xchange:sender {"http://airline.com"},
              accomodation-granted {{ hotel {{ }} }}
       }} before "18:00"
  }
FROM
 in { resource {"http://hotels.net"},
   accomodation {{
     hotels {{ city {"Rome"},
               var H ⇝hotel {{
                             phone { var Tel } }}
       }}  }}
  }
```

```
END
```

Note the XChange event query (specified in the `ON` *part of the rule) that is used to detect the* conjunction *of the occurrence of an atomic event (i.e. the flight cancellation) and the non-occurrence of another atomic event (i.e. the accommodation notification) that is limited by a time point (i.e. before 18:00).*

Chapter 6

Updates and Evolution in Bioinformatics Data Sources

Bioinformatics data is growing at tremedous rates and there are over 500 online databases and tools available. These databases can be classified as primary and secondary, which are derived from one or more primary databases. Users of such data sources keep local copies of these primary and secondary databases and often derive tertiary data sources. Keeping local and remote databases in sync and consistent is an important problem, which requires techniques to deal with evolution and reactivity.

6.1 Bioinformatics and the Semantic Web

Biology is changing rapidly as technological breakthroughs generate masses of data (see Fig. 6.1). Public databases contain over 1 000 000 protein sequences, over 25 000 protein structures, and over 12 000 000 scientific articles. These data are made accessible over the Web in numerous formats ranging from flat files to XML and RDF. At the same time, biologists are developing ontologies to standardise data and their annotation. Currently, the most prominent ontology is the GeneOntology (GO, www.geneontology.org), which has been designed for the annotation of genes. Over the past years GO has developed into the main ontology in molecular biology and it comprises over 19.000 terms organised in three subontologies for cellular location, molecular function and biological process.

Based on this data explosion in biology and on the pressing need to integrate data, an NSF and EUs strategic research workshop found that bioinformatics could play the role for the semantic web, which physics played for the web. The W3C also focuses on semantic web and the life sciences with a workshop held in October 2004 and working groups put in place. The reason that bioinformatics data can play an important role for the semantic web is that

- there are masses of information,
- there are masses of publicly accessible online information,
- data is more and more often published in XML,
- data standards are accepted and actively developed,

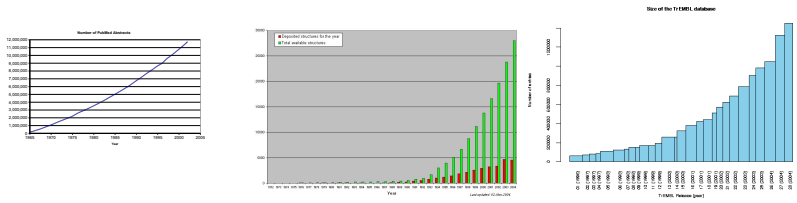

Figure 6.1: Superlinear growth of publicly accessible bioinformatics data. The graphs show the growth in the number of scientific papers being published, the growth of the number of protein sequences in the main sequence database UniProt and the growth of protein structures depoisted in the Protein DataBank.

- much valuable information is scattered (as production cheap and hence not centralised),
- systems integration and interoperation are prime concerns
- there are ontologies, which are referenced by many data sources.

In the light of these developments technology handling evolution of data and reactive agents, which integrate data, is important to develop the core of a Bioinformatics Semantic Web populated by a number of sample data sources. Evolution and reactivity play a role in demonstrating novel, reasoning-based solutions dealing with the following problems:

- Rules for mediation and to formulate complex queries
- Consistent integration of Bioinformatics data
- Adaptive portals for molecular biologists

6.2 Sample data sources and application

6.2.1 GoPubMed, ontology-based literature search

The main application under development uses ontologies to explore literature search results. Current approaches to literature search are based on keyword searches and results are presented as lists. Such approaches have two shortcomings: (a) in order to choose the keywords leading to good results users need to understand the domain very well; (b) list presentations are linear and thus do not capture multiple views of the results. Furthermore, list-based presentation encourages users to pursue only a few top hits, while more relevant papers may remain unexplored.

GoPubMed, a tool developed by TU Dresden in the A2 group, uses the GeneOntology (GO), a vocabulary for molecular biology, to structure large amounts of relevant literature. GoPubMed submits keywords to PubMed, extracts GO-terms from the retrieved abstracts, and presents the relevant sub-ontology for browsing. GoPubMed has a number of advantages. First, instead of pursuing only top hits, users get a high-level overview of the whole search result. Second, users are not forced to view multi-dimensional and thus often incomparable articles in a one-dimensional list. Instead they explore different search result "categories", which are

hierarchically ordered and therefore allow for fast navigation from general to specific concepts. Third, the use of GO enables one to derive general keywords relevant to the search although they are not mentioned in the article at all, as they are derived indirectly from the GO. Forth, our approach and system are general and can be applied to other ontologies and text bodies.

Consider the following example: A researcher wants to know which enzymes are inhibited by levamisole. A keyword search for "levamisole inhibitor" in PubMed produces well over 100 hits in PubMed. To find out about specific functions, we have to go through all these papers. We are interested in the relevant enzymatic functions. From the first titles it immediately is evident that levamisole inhibits alkaline phosphatase. A less well-known fact is however still buried in the abstracts. An abstract with low rank states that levamisole also inhibits phosphofructokinases. Without knowing specific activities, which a user is interested in, refining a keyword is also difficult. E.g. a search for "levamisole inhibitor enzymatic activity" produces only 5 hits: enough to learn about alkaline phosphatase, not enough to learn about the phosphofructokinase.

The two main data sources underlying GoPubMed are GeneOntology and PubMed, which are summarised below and described in more details in deliverable A2-D2.

6.2.1.1 GeneOntology

GeneOntology, www.geneontology.org, is a controlled, hierarchical vocabulary. GO has been designed for the annotation of genes. It comprises over 19.000 terms organized in three sub-ontologies for cellular location, molecular function and biological process. GO was initially created to reflect gene function of fruitflies, but has expanded to encompass many other genomes as well as sequence and structure databases. The hierarchical nature of GO allows one to quickly navigate from an overview to very detailed terms. As an example, there are maximally 16 terms from the root of the ontology to the deepest and most refined leave concept in GO.

GO is available in free text, XML and as database. The XML version is updated on a monthly basis. The deliverable A2-D2 contains further details on GO.

6.2.1.2 PubMed

PubMed, the main biomedical literature database references over 12.000.000 abstracts. It has grown by some 500.000 in 2003 alone. Besides biology it covers fields such as medicine, nursing, dentistry, veterinary medicine, the health care system, and the preclinical sciences. PubMed contains bibliographic citations and author abstracts from more than 4,600 biomedical journals published in 70 countries. Abstracts date back to the mid-1960's. Coverage is worldwide, but most records are from English-language sources or have English abstracts. PubMed is available in XML.

6.2.2 PDB and SCOP

Two other sources, which are widely used in the A2 group are PDB, the protein databank, and SCOP, the structure classification of proteins.

PDB is a repository of the atomic coordinates of proteins and nucleic acids. Entries contain among others, besides the coordinates, the resolution at which the coordinates have been obtained, the authors, who submitted the data, literature references, the species the data is coming from. PDB structures are classified according to their evolution by SCOP. SCOP contains four main structural classes, which are refined into some 1000 structural families of proteins. PDB

is updated every week, SCOP every 6 months. PDB is available as XML and flat file, SCOP as flat file.

6.3 Organisation of Bioinformatics Data

GeneOntology, PubMed, and PDB are examples of primary data sources and SCOP and GoPubMed are secondary. Consider Fig. 6.2. The primary databases are hosted at remote sites, which are updated at some interval with additions, deletions, and modifications of entries. Other sites, such as GoPubMed, then derive secondary databases from the primary ones. Whenever the primary database A is updated then the secondary databases should be updated, too. However, sometimes this is not possible as a secondary database, which a user wishes to use, is also remote, such as SCOP. I.e. there is a local tertiary database F depending on the local copy E of the remote secondary database B, which is derived from A. While F and E can be managed locally, A and B are outside the power of the local site. This can lead to problems, when A removes data, but B does not include the update instantly.

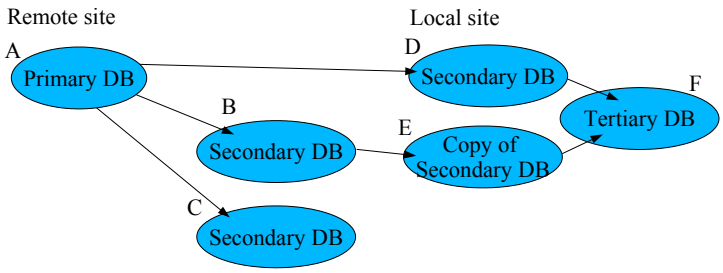

Figure 6.2: Typical organisation of bioinformatics data.

6.4 Reactivity/evolution scenario: GoPubMed, GO, PubMed, SCOP, and PDB

As the data sources are distributed, as there are dependencies between them, as they are updated at different intervals, there are requirements for dealing with evolution and reactivity.

Use Case 6.4.1 (Caching and Actuality of data in GoPubMed, PubMed, and GO) *GoPubMed is a distributed application: A query entered in GoPubMed is submitted on-the-fly*

to the remote PubMed site, which returns relevant articles. These are then annotated by GoPubMed with relevant terms from the GO, a local copy of which is residing at the GoPubMed site. To integrate the three sources, the GoPubMed application needs to exhibit reactive behaviour. On the event of a user query, a request is sent to PubMed. On the event of an answer from PubMed, a local cache is consulted. If the abstracts are not cached, then GoPubMed sends another message to PubMed requesting the abstracts. On receipt of them, they are annotated. Finally, the results are compiled an presented to the user. Overall, there are different distributed data sources, which are communicating with each other using event-condition-action rules.

Use Case 6.4.2 (Mirroring, Actuality, and Consistency of data in SCOP and PDB) *The original SCOP data is published on a website hosted in Cambridge. A researcher may have a copy of SCOP on his laptop besides a copy hosted at his university. The copy on the laptop is not up-to-date, so that the researcher usually uses the remote database, but when offline he is forced to use the local laptop copy. The researcher wants to transparently access SCOP and this access needs to handle the preference of the remote SCOP copy over the local SCOP copy.*

A reactive agent acts as a wrapper of the original SCOP site and data and upon the event of a new release it informs a local agent, who updates the local SCOP copy.

Updates of PDB can lead to inconsistencies as SCOP is derived from PDB and as PDB is updated weekly, while SCOP is only updated every six months. If a PDB entry gets withdrawn between two SCOP releases, then the constraint is violated that every SCOP entry should have a PDB entry it is derived from. The constraint can be satisfied if we know which PDB entry replaces a withdrawn PDB entry. Then the local SCOP copy can be updated accordingly.

To summarise, the above applications require infrastructure, which supports the development of distributed applications, whose behaviour is programmed by event-condition-action rules. These rules need to be tightly integrated with the underlying databases and access to XML documents. Access to web services is also desirable. As hierarchies such as GO and SCOP are used, rules are needed to reason over them.

Chapter 7

Conclusions

In this report we have presented a set of use cases for evolution and reactivity on the Web. These use cases will now serve as guidance for the definition of models, languages and architectures for dealing with reactivity, and for testing forthcoming implementations.

The choice of the use cases was made with several goals in mind. First, it was our intention to provide a set capable of illustrating all of the features and concepts that we find important for a reactive and evolving Web. Second, it should make clear that the aspects of evolution and reactivity that we intend to develop are, as much as possible, parametric on the underlying languages for querying and for making the local changes in data. Third, the choice of the use cases should be such that collaboration with other working groups of Rewerse is potentiated.

With these goals in mind, we presented three different case study scenarios, in three different application areas, each more related to each of the Application Work Groups of Rewerse.

In the "Rewerse Portal" scenario we illustrate many of the details and basic features of evolution and of a reactive language for the Web. In this scenario, since it is mostly internal to Rewerse, we may have control over what is the information in place, and so it may serve as a very good testbed for the basic features of our developments. Moreover, it is a scenario which is of interest to other working groups (namely A3, TTA and PRA, as explained above), and this way it will be an opportunity for collaboration between these working groups.

The "Organising Travels" scenario reveals some extra features and constructs that a language for evolution and reactivity on the Web should have. Namely, it better shows some aspects of complex events and transactions, and highlights the need for a good integration with query languages. Also, the need for location reasoning and the capability to use different calendars and constraints over time notions is made obvious. Thus, a strong motivation exists for cooperation and integration of research results between the Rewerse Working Groups regarding evolution and reactivity (I5), reasoning-aware querying (I4), and Web-based decision support for temporal, event, and geographical data (A1) [BBL+04].

Finally, the "Bioinformatics" scenario illustrates the practical need of some evolution and reactivity features for the work in working group A2, for maintaining actuality and consistency of data in large data sources in the area of bioinformatics. The need for collaboration with A2 in this case is clear.

Bibliography

[ABB+04] J. J. Alferes, J. Bailey, M. Berndtsson, F. Bry, J. Dietrich, A. Kozlenkov, W. May, P. L. Pătrânjan, A. Pinto, M. Schroeder, and G Wagner. State-of-the-art on evolution and reactivity. Technical Report IST506779/Lisbon/I5-D1/D/PU/a1, REWERSE, September 2004.

[APPP02] J. J. Alferes, L. M. Pereira, H. Przymusinska, and T. Przymusinski. LUPS: A language for updating logic programs. *Artificial Intelligence*, 132(1 & 2), 2002.

[BBL+04] S. Berger, F. Bry, B. Lorenz, H. J. Ohlbach, P.-L. Pătrânjan, S. Schaffert, U. Schwertel, and S. Spranger. Reasoning on the Web: Language Prototypes and Perspectives. In *Proc. of European Workshop on the Integration of Knowledge, Semantics and Digital Media Technology, London, U.K*, pages 157–164. IEE, 2004.

[BBSW03] S. Berger, F. Bry, S. Schaffert, and C. Wieser. Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data. In *Int. Conf. on Very Large Databases (VLDB)*, 2003.

[BFPS04] F. Bry, T. Furche, P.-L. Pătrânjan, and S. Schaffert. Data Retrieval and Evolution on the (Semantic) Web: A Deductive Approach. In *Workshop on Principles and Practice of Semantic Web Reasoning*. Springer, 2004.

[BK94] A. J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133(2):205–265, 1994.

[BK95] A. J. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *ICDT'95: Advances in Logic-Based Languages*, 1995.

[BKK04] M. Bernauer, G. Kappel, and G. Kramler. Composite Events for XML. In *13th Int. Conf. on World Wide Web*. ACM, 2004.

[BP05] F. Bry and P.-L. Pătrânjan. Reactivity on the Web: Paradigms and Applications of the Language XChange. In *20th Annual ACM Symposium on Applied Computing (SAC'2005)*. ACM Press, 2005.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proceedings of the 20th VLDB*, pages 606–617, 1994.

[DOM98] Document object model (DOM). `http://www.w3.org/DOM/`, 1998.

[EFST01] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. A framework for declarative update specifications in logic programs. In Bernhard Nebel, editor, *Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01)*, pages 649–654, San Francisco, CA, 2001. Morgan Kaufmann Publishers, Inc.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Lei03] J. A. Leite. *Evolving Knowledge Bases*, volume 81 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2003.

[MAB04] Wolfgang May, José Júlio Alferes, and François Bry. Towards generic query, update, and event languages for the Semantic Web. In *Principles and Practice of Semantic Web Reasoning (PPSWR)*, number 3208 in LNCS, pages 19–33. Springer, 2004.

[Mil83] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[SB04] S. Schaffert and F. Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Int. Conf. Extreme Markup Languages*, 2004.

[Sch04] S. Schaffert. *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. Dissertation, 2004.

[Sin95] Munindar P. Singh. Semantical considerations on workflows: An algebra for intertask dependencies. In *Intl. Workshop on Database Programming Languages*, electronic Workshops in Computing, Gubbio, Italy, 1995. Springer.

[xqu01] World Wide Web Consortium, http://www.w3.org/TR/xquery/. *XQuery: A Query Language for XML*, Feb 2001.

Acknowledgement

This research has been co-funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).