A2-D5

Links to other WGs

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/Dresden/A2-D5/D/PU/b1
Responsible editors:	Loïc Royer and Michael Schroeder
Reviewers:	Gihan Dawelbait and Albert Burger
Contributing participants:	Dresden, Munich, Zurich
Contributing workpackages:	A2
Contractual date of deliverable:	31 August 2006
Actual submission date:	31 August 2006

Abstract
The deliverable gives an overview over A2's links to working groups I2, I4, and I5. It discusses how the language ACE can be used for extraction of protein interactions from literature (A2/I2), how Xcerpt can be used to query bioinformatics contents (A2/I4), and how Prova can be used to implement reactive bioinformatics workflows (A2/I5).

Keyword List
Bioinformatics and querying, bioinformatics and reactivity, bioinformatics and reasoning

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

Links to other WGs

François Bry[Mun], **Gihan Dawelbaith**[Dre], **Norbert Fuchs**[Zur] **Tim Furche**[Mun], **Tobias Kuhn**[Zur], **Benedikt Linse**[Mun], **Loïc Royer**[Dre], **Michael Schroeder**[Dre],

[Dre] Technische Universität Dresden, Germany, [Mun] LMU Munich [Zur] University of Zurich,

31 August 2006

Abstract
The deliverable gives an overview over A2's links to working groups I2, I4, and I5. It discusses how the language ACE can be used for extraction of protein interactions from literature (A2/I2), how Xcerpt can be used to query bioinformatics contents (A2/I4), and how Prova can be used to implement reactive bioinformatics workflows (A2/I5).

Keyword List
Bioinformatics and querying, bioinformatics and reactivity, bioinformatics and reasoning

Contents

Contents

1 Introduction

The A2 group works toward a semantic web for the life sciences. As documented in previous deliverables, A2 has developed various use cases and applications using rules and reasoning on the web. Here, we document how A2 links to the I-technology working groups. There are five technology working groups:

- I1, rule mark up: This working group is a prerequisite for other I-groups, as it defines syntax and markup. There is no direct link to A2.
- I2, policy languages, enforcement, composition: Among others the I2 group works on ACE, Atempto Controlled English. We show how this formal language can be used for text-mining of biomedical literature. The result of this collaboration between I2 and A2 has been published in:
  - Tobias Kuhn, Loïc Royer, Norbert E. Fuchs, and Michael Schroeder. Improving text mining with controlled natural language: A case study for protein interactions. In Ulf Leser, Barbara Eckman, and Felix Naumann, editors, Proc. of 3rd International Workshop on Data Integration in the Life Sciences 2006 (DILS'06), volume LNBI. Springer, 2006
- I3, composition and typing: Typing is an important mechanism to achieve consistency of a model. In systems biology, there are mark-up languages, which allow sharing of metabolic networks. Typing is useful for such networks to infer properties of proteins and interactions. The Paris group, who is a member of I3 and A2, has developed the Biocham system and has developed a type system for it, which is published in:
  - François Fages, Sylvain Soliman, Type Inference in Systems Biology, Proceedings of Computational Methods in Systems Biology, CMSB 2006.
- I4, querying: A central working group of REWERSE is dedicated to query languages for the web. As discussed in previous deliverables, bioinformatics poses specific requirements to querying, as data can be large, distributed, and in varying formats such as text, HTML, XML. A2 groups from Bucarest, Dresden, and Edinburgh have cooperated with I4 to discuss and address these problems. The results have been published in 4 papers. The two papers by Bry et al. discuss general design principles for the semantic web. Doms et al. and Royer et al. compare the concrete query languages Xcerpt and Prova.
  - François Bry, Tim Furche, Liviu Badea, Christoph Koch, Sebastian Schaffert, and Sacha Berger: Querying the Web Reconsidered: Design Principles for Versatile Web Query Languages. International Journal of Semantic Web and Information Systems (IJSWIS) 1 (2), April-June 2005
  - François Bry, Christoph Koch, Tim Furche, Sebastian Schaffert, Liviu Badea, Sacha Berger: Querying the Web Reconsidered: Design Principles for Versatile Web Query Languages. Int. J. Semantic Web Inf. Syst. 1(2): 1-21 (2005)
  - Andreas Doms, Tim Furche, Albert Burger, and Michael Schroeder. How to query the GeneOntology. In Catherine Bounsaythip, editor, Proc. of KRBIO'05: Symposium on Knowledge Representation in Bioinformatics, 2005

  - Loïc Royer, Benedikt Linse, Thomas Wchter, François Bry, and Michael Schroeder. Querying the semantic web: A case study. In Christopher Baker and Kei-Hoi Cheung, editors, Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences. Springer, 2006

- I5, reactivity and evolution: The I5 group develops a general framework for reaction rules including concrete implementations such as Prova and RuleCore. A2 and I5 collaborate on Prova, a language specifically designed for systems integration in bioinformatics. Kozlenkov et al. discusses a use case involving a client/server architecture with remote servers, which implement a workflow integrating detailed information for proteins from remote sites.

  - Alex Kozlenkov, Rafael Penaloza, Vivek, Nigam, Loïc Royer, Gihan Dawelbait, and Michael Schroeder. Prova: Rule-based Java Scripting for Distributed Web Applications: A Case Study in Bioinformatics. In Sebastian Schaffert, editor, Proceedings of Workshop on Reactivity on the Web at the International Conference on Extending Database Technology (EDBT 2006). Springer, 2006

To summarise, the collaboration between I-groups and A2 are documented in 7 papers spanning a wide range of technologies from I2, I3, I4, and I5. Here, we give details of the collaboration between A2 and I2 (chapter 2), A2 and I4 (chapter 3), A2 and I5 (chapter 4).

2 Improving Text Mining with Controlled Natural Language: A Case Study for Protein Interactions (A2/I2)

2.1 Abstract

Linking the biomedical literature to other data resources is notoriously difficult and requires text mining. Text mining aims to automatically extract facts from literature. Since authors write in natural language, text mining is a great natural language processing challenge, which is far from being solved. We propose an alternative: If authors and editors summarize the main facts in a controlled natural language, text mining will become easier and more powerful. To demonstrate this approach, we use the language Attempto Controlled English (ACE). We define a simple model to capture the main aspects of protein interactions. To evaluate our approach, we collected a dataset of 459 paragraph headings about protein interaction from literature. 56% of these headings can be represented exactly in ACE and another 23% partially. These results indicate that our approach is feasible.

2.2 Introduction

We introduce a new paradigm of how to make knowledge of scientific papers accessible by computers. We focus on the fields of life sciences – particular biology – but our approach could be used in other fields as well.

Our approach consists of letting authors express their scientific results in a formal summary that could be an integral part of the papers they publish. We argue that it is more reasonable to let the authors formalize their own results, instead of trying to extract these results from the articles.

This section explains our motivation, introduces the language Attempto Controlled English (ACE) and compares it with other knowledge representation languages. Section 2.4 shows how ACE is used to build an ontology for protein interactions. In Sect. 2.5 we use this ontology as foundation for the expression of scientific results and we show how 89 selected articles could have been summarized in ACE. Section 2.6 shows the benefits of our approach and Sect. 2.7, finally, gives a short outlook.

2.3 Motivation

Biomedical scientists are challenged by an ever-increasing amount of scientific papers. The indexing service *PubMed*[1] shows the huge quantity of literature that the scientists have to face. It contains at the moment 16 million articles and grows every year by over 600'000 articles. All these biomedical articles are written in natural language. That means that we cannot easily process them with computers. But, facing the quantity of literature, it is clear that we need computational support in order to manage the contained knowledge.

In the last years, *text mining* and *information extraction* – which build both upon natural language processing (NLP) – gained an increasing interest in biomedical sciences. They aim to extract some kind of formal knowledge from natural language texts, which is generally considered a very demanding task. Even the basic problem of *named entity recognition*, that aims to identify named entities (e.g. protein names) in natural texts, is far from being solved. Other major aspects of text mining are the extraction of relationships (e.g. protein interactions), the automatic classification of texts, and the generation of new hypotheses on the basis of the available literature [3]. The *BioCreAtIvE* contest [21] nicely shows, that even sophisticated tools for text mining have a considerable lack of precision and recall: For a

[1] `http://www.pubmed.gov`

simple “named entity recognition”-task the precision ranged up to 86% and the recall was at most 84%. Another attempt is described in [4]: Information about protein-interactions was extracted from a data set of 1.2 million sentences that were taken from biomedical abstracts. They achieved a precision of 91%, but with a poor recall of only 21%. We recommend [3] and [12] for a more comprehensive overview of the “accomplishments and challenges” of text mining.

As a first step towards a better management of biomedical literature, controlled vocabularies like *MeSH*[2] and the *Gene Ontology*[3] have been created. They serve to classify biomedical publications and to link them to other resources. *GoPubMed*[4], for example, is a search engine that connects the abstracts from PubMed with the formal structure of the Gene Ontology. Thus a researcher can exploit the Gene Ontology for the search of relevant literature. Such tools are very valuable for scientists and there has been a notable progress in the last years, but it will never be possible to extract all the information correctly. There is inherent ambiguity and vagueness in natural language that prevents its perfect processing by computers.

For this reason we present an alternative approach: The authors of scientific articles formally summarize their own results. Such formal summaries are added to the articles which makes them processable by computers. This requires a formal language that on the one hand is easy to learn and understand, and on the other hand is expressive enough to represent even complicated scientific results. It is clear that this approach is not applicable for papers that have been written without the formal summaries, and that means that we still need NLP or manual extraction for such papers. Thus we propose rather a concept for the future than a solution for today’s problems. To explore our approach we use Attempto Controlled English as knowledge representation language.

2.3.1 Formalization of Scientific Results

Since we want to access scientific results by computers, we have to formalize this knowledge at some point. Today researchers write their results in natural language. To extract these results and to formalize them, manual or computer-supported text mining is necessary. Thus the formalization is accomplished by computer-programs or by humans, and in either case it is done without the help of the corresponding researchers. The article is the only source of information. Since such articles are highly domain-specific, they require a lot of background knowledge. Therefore the formalization is a very demanding task, even for humans. Altogether this causes a lot of knowledge to be lost in the vast amount of biomedical literature.

We claim that most of these problems can be solved, if we simply let the authors of scientific articles formalize their own results. The researchers themselves are the most qualified to understand their results, and thus they can give the most precise formal representation. This is not even a big extra-effort for a scientist, since he already has a – more or less – formal model of the domain in his mind, and must write an abstract anyway. He just needs to learn how to express his knowledge in a formal way. This means that we need to provide an intuitive, yet formal language in which a scientist can write his results.

2.3.2 Attempto Controlled English

Attempto Controlled English (ACE)[5] is a controlled natural language that has been developed by Norbert E. Fuchs and his group at the University of Zurich. ACE is a subset of natural English with a

[2] `http://www.nlm.nih.gov/mesh/meshhome.html`
[3] `http://www.geneontology.org`
[4] see [5] and `http://www.gopubmed.org`
[5] see [7], [8], and `http://www.ifi.unizh.ch/attempto/`

restricted grammar. There are no limitations on the vocabulary, apart from some function words with predefined meanings (e.g. 'every', 'of'). ACE looks like English, but it is in fact a formal language, which means that texts can be translated unambiguously into first-order logic. Some ACE sentences would be ambiguous in natural English, but ACE provides interpretation rules that allow in each case only one reading. The report [13] contains a comprehensive description of the syntax of ACE.

In order to be able to write ACE texts, one has to learn the restrictions on the grammar. Thus, like every formal language, ACE has to be learned. However, since it looks like natural English, everyone is able to understand ACE texts with almost no training. This is a big advantage over other formal languages.

The Attempto parser APE[6] translates ACE texts into Discourse Representation Structures [6]. Such structures are equivalent to expressions in first-order logic, and thus every ACE sentence has a logical representation. Furthermore, APE creates a paraphrase that shows the interpretation of an ACE text. If a writer is not familiar with the ACE interpretation rules, then he can check the paraphrase for the validation of his ACE text.

2.3.3 Comparison of Knowledge Representation Languages

In order to show the benefits of ACE, we compare it with four other knowledge representation languages: first-order logic [9], Description Logics (DL) [15], Web Ontology Language (OWL) with its RDF/XML-syntax [14], and Unified Modeling Language (UML) [2].

We have to state that these four languages are not independent. DL and ACE build upon first-order logic, and DL are the basis for OWL. While first-order logic and DL focus on the theoretical concepts of knowledge, OWL, UML, and ACE concentrate on the implementation and application of knowledge representation. Nevertheless we dare to give a direct comparison between these five languages.

Figure 1 shows how the fact 'everything that is a protein has a terminus' is expressed in the five different languages. The OWL representation (using the RDF/XML syntax) is the most verbose and – from the human perspective – the least readable one. The representations in first-order logic and DL are more concise, but they are still not understandable for people who are not familiar with formal notations. The graphical notation of UML looks nice, but for a non-specialist it is hard to guess the meaning of all the shapes and arrows. The ACE representation, in contrast, should be immediately understandable for any English speaking person. It looks perfectly like natural English and thus the reader might not even recognize that it is a formal language.

We can state that controlled natural languages like ACE minimize the gap between machines and humans. A reader is able to understand such languages with almost no training. Furthermore, writing sentences in a controlled natural language is possible with only little effort, especially if the writer is supported by an authoring tool (see Sect. 2.5.3).

2.4 Ontology for Protein Interactions in ACE

In order to have a clear basis for the formal representation of scientific knowledge, we defined an ontology for proteins and their interactions. This section shows how ACE can be used as an ontology language, and introduces our ontology for protein interactions.

[6] http://www.ifi.unizh.ch/attempto/tools/

first-order logic	$\forall X(protein(X) \rightarrow \exists Y(terminus(Y) \wedge has(X,Y)))$
DL	$Protein \sqsubseteq \exists has.Terminus$

OWL (RDF/XML)	`<owl:Class rdf:ID="Protein">`
	`  <rdfs:subClassOf>`
	`  <owl:Restriction>`
	`  <owl:onProperty rdf:resource="#has"/>`
	`  <owl:someValuesFrom rdf:resource="#Terminus"/>`
	`  </owl:Restriction>`
	`  </rdfs:subClassOf>`
	`</owl:Class>`

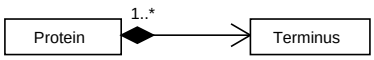
UML	
ACE	Every protein has a terminus.

Figure 1: Comparison of first-order logic, DL, OWL, UML, and ACE

2.4.1 Ontologies

The main goal of an ontology is to provide a *shared understanding* of a certain domain. This shared understanding can serve as basis for the communication between people, for the interoperability between systems, for the improvement of reusability and reliability of software systems, and for the specification of software [20]. Furthermore ontologies are an excellent basis for the formal representation of knowledge [11].

Ontologies are not yet broadly established in science, but they are expected to gain a very important role in the future, especially in life sciences. The *Gene Ontology* is the most famous example, although it is actually more a controlled vocabulary than a real ontology.

2.4.2 Ontology Elements

In order to provide basic structures for ontologies in ACE, we adopt the elements of DL – individuals, concepts, and roles – and we call them *ontology elements*. Furthermore we introduce an additional structure: context information.

Individuals. Individuals stand for single objects of the domain. They are represented in ACE as *proper names* like 'Bub1' (that stands for a protein) or 'Alzheimer' (that stands for a disease).

Concepts. Concepts stand for sets of objects, and there are two possibilities to express them in ACE. *Common nouns* are the most straight-forward way. The noun 'protein', for example, can stand for the concept of all proteins. As a second possibility we can use *adjectives* (in their positive form). The adjective 'organic', for example, can be used for the concept of all organic substances.

Roles. Roles stand for binary relations between objects, and they can be expressed in four different ways. First of all, we can use *transitive verbs* for expressing roles. For example, we can use 'interacts-with' to express a relationship between proteins. Next, we can combine transitive verbs with *adverbs*. For example, we can use the adverb 'directly' together with the transitive verb 'interacts-with' to express the role 'directly interacts-with'. As a third possibility we can use *of-constructs* like 'is a part of'. Due to the syntax of ACE, 'of' is the only allowed preposition for

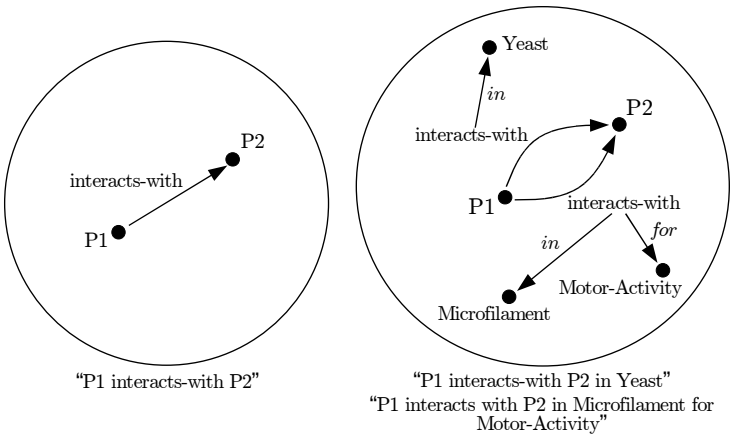


Figure 2: Context information

nouns. Finally, we can use *constructs with comparative forms of adjectives* like 'is larger than'. Such constructs typically represent transitive relationships.

Context Information. The examination of the results of scientific papers on protein interactions showed that normal roles are often not sufficient to express the needed information. We can express simple statements like 'P1 interacts-with P2', but we cannot express statements with contextual information like 'P1 interacts-with P2 in Yeast' or 'P1 interacts-with P2 in Microfilament for Motor-Activity'. In order to be able to express such results, we want to allow roles to have such additional information. In natural English we usually express such information with prepositional phrases, and this is exactly the way we will do it in ACE. Figure 2 illustrates the examples without and with context information.

Using these ontology elements, we can state for example

P1 is a protein and directly interacts-with P2 in Yeast.

where 'P1', 'P2', and 'Yeast' are individuals, 'protein' stands for a concept, and 'directly interacts-with' stands for a role. The phrase 'is a' is used to assign the individual 'P1' to the concept 'protein'. The conjunction 'and' connects the statements flanking left and right. The preposition 'in', finally, connects to the context 'Yeast'.

2.4.3 Ontology for Protein Interactions

Since we found no existing ontology that fits our needs, we had to create our own ontology for protein interactions. First, we defined concepts that allow us to make statements about the structure of proteins and protein-complexes. For the sake of a clear structure, we introduced the concept *protein-unit*, which is either a protein or a protein-complex, and *protein-component*, which is either a protein-unit or a region of a protein. In order to describe the structure of such regions, we defined concepts like 'residue', 'secondary-structure', and 'domain'.

Next, we defined the roles for the description of interactions between proteins like 'interacts-with' or 'binds'. We can also express more complicated interactions like 'increases the phosphorylation of'.

Furthermore, we defined some concepts for expressing additional information about proteins, like the localization to a certain cellular component or the participation in a certain process. The big picture of this ontology is shown in Fig. 3.

2.5 ACE Summaries

Our goal is to show how scientists could write formal summaries of their results. There are some questions that naturally arise: What are these results about? How complex is it to formulate them in a formal language? In the following we present an empirical study of the feasibility of our approach.

2.5.1 ACE Summaries for 89 Selected Articles

Since we want to show how results of papers about protein interactions could have been written in ACE in the first place, we picked 89 *Elsevier*-articles that concern protein interactions. Such articles mostly have a section called "Results" which is subdivided into subsections. The headings of these subsections are short descriptions of the corresponding results. It turned out that these headings are highly suitable for a manual translation into ACE. Please note that the intended methodology is *not* to express the results first in natural language and then to translate them into ACE. We do this just to demonstrate the feasibility of our approach.

The 89 articles contain 457 such headings. 184 of them are ignored, because they are not formulated as facts (e.g. "Functional characterization of Pellino2"[7]) or because they contain information that is not about protein interactions.

total:		457	*(100%)*
ignored:	(not a fact)	87	*(19%)*
	(off-topic)	97	*(21%)*
used:		273	*(60%)*

We then tried to translate the 273 remaining headings into ACE. For 154 of them there is a perfect match, which means that the complete information can be expressed in ACE; e.g. the heading "Interaction of Act1 with TRAF6"[8] can be rephrased perfectly as "Act1 interacts-with TRAF6". For another 62 headings only a part of the information is expressed; e.g. the heading "The mtFabD protein is part of the core of the FAS-II complex"[9] can only partially be rephrased as "MtFabD is a subunit of FAS-II". For the remaining 57 headings there is no translation at all.

used:		273	*(100%)*
matched:	(perfect)	154	*(56%)*
	(partial)	62	*(23%)*
unmatched:		57	*(21%)*

Let us take a closer look at the reason, why 119 headings cannot be rephrased in ACE at all, or only partially. 56 of them could not be rephrased because their content is not covered by our model, but they could be expressed with an extended model. Another 21 headings describe relations of relations, like the heading "Kal-GEF1 activation of Pak does not require GEF activity"[10]. In this case, there is a relation between two objects ("Pak activates Kal-GEF1") and this relation itself stands in another relation ("... does-not-require GEF-activity"). At the moment, we cannot express such structures in

[7] see article *PMID 12860405*
[8] see article *PMID 12459498*
[9] see article *PMID 16213523*
[10] see article *PMID 15950621*

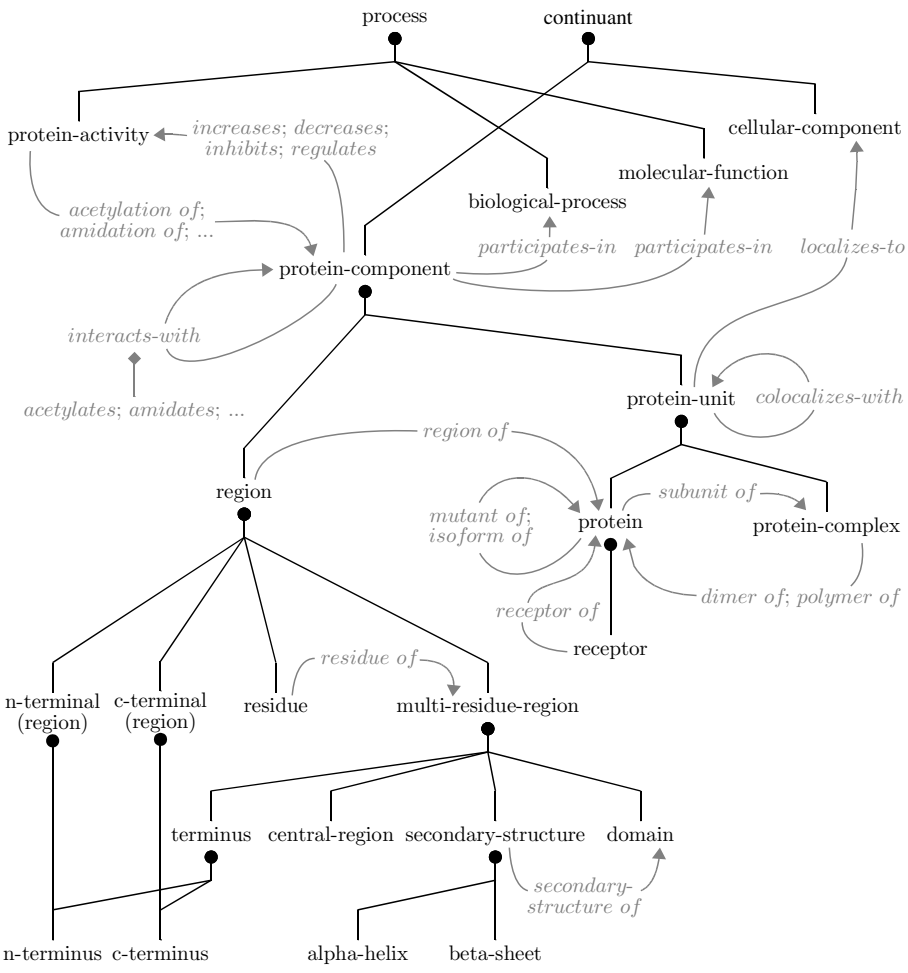


Figure 3: The structure of the ontology for protein interactions

ACE in a satisfying way. But there are attempts to extend the language ACE, and we hope that we will be able to express such statements in the future. Furthermore there are 11 headings with fuzzy statements (e.g. "ANKRD contains potential CASQ2 binding sequences ..."[11]) and 31 headings that we could not understand without reading the whole article.

not perfectly matched:	119	*(100%)*
not covered by our model:	56	*(47%)*
relations of relations:	21	*(18%)*
fuzzy:	11	*(9%)*
not understood:	31	*(26%)*

Thus, altogether we could rephrase 79% of the relevant headings, either partially or perfectly. This makes us confident that our approach is feasible for practical use. The reason, why 119 headings are not rephrased perfectly, is mostly our simple model and our lack of understanding. If we used a more detailed model, and if we let the scientists themselves express their own results in ACE, then we expect to be able to express much more than 79% of the results.

2.5.2 ACE Summary as an Integral Part of an Article

Since ACE looks like natural English, every reader of a scientific article is able to understand ACE texts. Thus the ACE summary of the results could be an integral part of the article. Figure 4 shows how an article with an ACE summary could look like[12]. Figure 5 shows the corresponding logical representation as a Discourse Representation Structure (consult [6] for details). As we see, the natural looking ACE summary can be translated automatically into a formal representation which is processable by computers.

Together with the abstract and a keyword list, the ACE summary gives a concise insight into the content. In contrast to the abstract, the ACE summary is readable by both, humans and machines; and in contrast to the keyword list, the ACE summary does not only mention the objects of interest, but describes the relations among them. Thus, every published article could be a contribution to a constantly growing knowledge base.

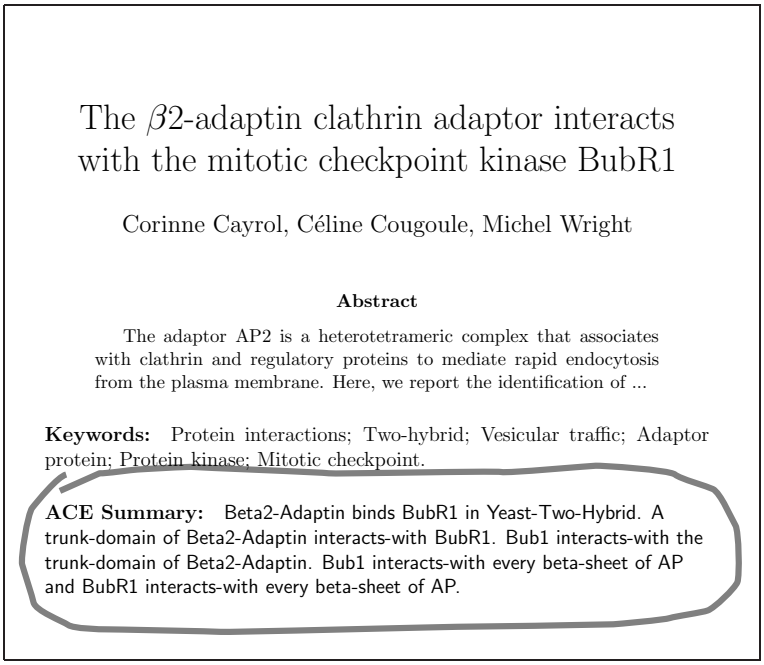
2.5.3 Authoring Tool

Now we sketch a tool that would help writing ACE texts. It would guide the user step by step and would need almost no training. Similar systems are the look-ahead editor *ECOLE* [17, 18], the natural language interface *LingoLogic* [19], and the *Ginseng*-system [1]. Our tool would solve several problems:

- The tool would help the user to comply with the standard nomenclature. The user would only be allowed to use the defined words. It would also prevent typing errors.
- It would make sure that the created sentences comply with the ACE syntax. At every stage, the tool would allow to proceed only in a way that leads to a correct ACE sentence. Thus the user would not need to know about the syntax of ACE.
- The tool would be aware of the structure of the ontology. In this way it would make sure, for example, that the domains and ranges of roles are respected.

[11] see article *PMID 15698842*

[12] article *PMID 12419313* is used for this example



The β2-adaptin clathrin adaptor interacts with the mitotic checkpoint kinase BubR1

Corinne Cayrol, Céline Cougoule, Michel Wright

Abstract

The adaptor AP2 is a heterotetrameric complex that associates with clathrin and regulatory proteins to mediate rapid endocytosis from the plasma membrane. Here, we report the identification of ...

Keywords: Protein interactions; Two-hybrid; Vesicular traffic; Adaptor protein; Protein kinase; Mitotic checkpoint.

ACE Summary: Beta2-Adaptin binds BubR1 in Yeast-Two-Hybrid. A trunk-domain of Beta2-Adaptin interacts-with BubR1. Bub1 interacts-with the trunk-domain of Beta2-Adaptin. Bub1 interacts-with every beta-sheet of AP and BubR1 interacts-with every beta-sheet of AP.

Figure 4: Article with ACE summary

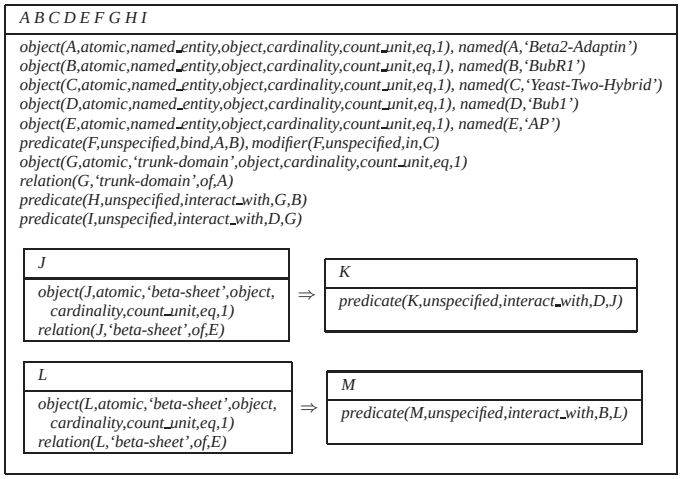


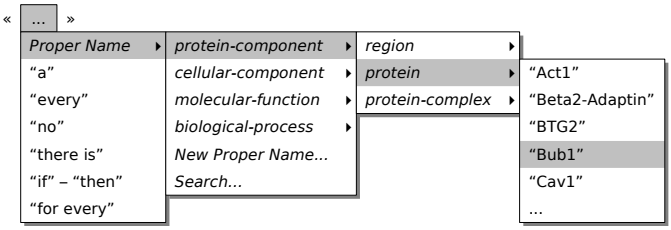
Figure 5: DRS-representation of the ACE summary

We give now an example how this tool could be used. Suppose that an author of the article that is shown in Fig. 4 wants to write down the fact that the protein *Bub1* interacts with the protein *β2-Adaptin* via its *trunk domain*.

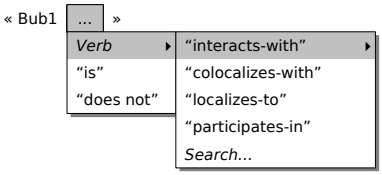
The sentences are created step by step by a simple menu. At the beginning there is just an empty sentence that might look like this:

« ... »

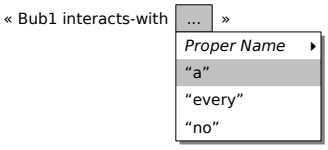
The quotes indicate the beginning and the end of the sentence and the box in the middle is used to create the content. If the user clicks on it, then a menu is displayed that shows the different options for beginning a sentence. Since we want to talk about the protein *Bub1* we first insert 'Bub1' as a proper name. This looks as follows.



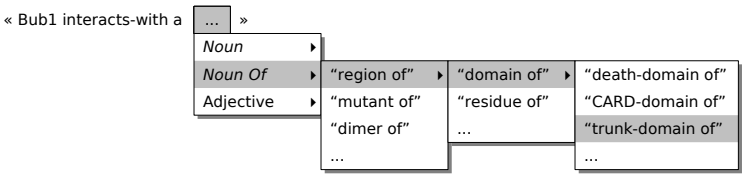
Proper names are hierarchically structured and the menu allows to navigate through this hierarchy. Alternatively, we can use the search option to find a certain term, or we can create a new proper name on-the-fly. In the next step we get



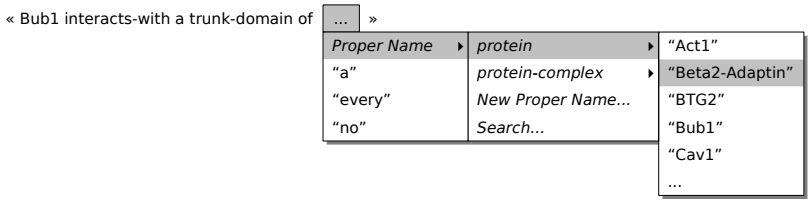
where the proper name 'Bub1' is now fixed as the beginning of the sentence, and we have a new menu with different entries. We want to express the interaction with another protein, and thus we choose the verb 'interacts-with'. Like proper names, verbs are hierarchically structured and we can navigate through this hierarchy. In the next step we get



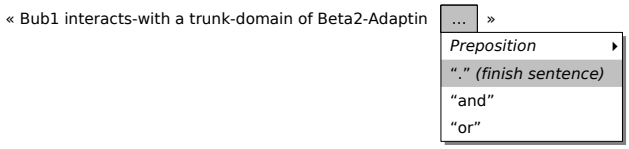
where we can define the second participant of the protein-interaction. Since we want to state that the interaction goes via a *trunk domain* of the protein *β2-Adaptin*, we first have to add the article 'a'. Then we get



where we can choose the 'trunk-domain of'-relation. Like proper names, such of-relations are structured in hierarchies through which we can navigate (the same holds for nouns and adjectives). After that we get



where we can specify the second protein 'Beta2-Adaptin'. Finally we get



where we could use prepositions to add context information, e.g. to specify the organism in which the interaction takes place. In our example, we now finish the sentence with a full stop.

For the creation of this sentence we did not need any further knowledge about ACE. Every person that is familiar with English and knows how to handle a simple menu, is able to create ACE texts. However, to make such a tool really user-friendly we will need a lot of usability testing, as it is done – with promising results – for the *Ginseng*-system [1].

2.6 The Benefits of our Approach

The preceding sections showed what needs to be done to express scientific results about protein interactions in ACE. Now it is time to take a look at the benefits.

Today there are many databases that contain life science data, but they are mostly unsynchronized, incomplete, and often not up-to-date. With our approach it would be much easier to provide complete and consistent databases.

Imagine that all the scientific papers about protein interactions summarize their results in ACE. We could use these formal summaries to build up a dynamically growing knowledge base about protein interactions. Of course, we would also have to collect all the knowledge that is contained in old papers. For these, we still need some form of classical text mining. But once we have such a knowledge base that is continuously updated with the results of new papers, then we would be able to answer many questions. We present now some examples.

Are some results consistent with an existing knowledge base or with other papers? We can check, whether an ACE summary is consistent with an existing knowledge base. If this knowledge base contains common knowledge, then the results should be consistent, or otherwise it can be seen as an appeal against the common knowledge.

Without formal declarations, it is impossible to check a paper for consistency. Probably there exist scientific papers that contain results which are inconsistent with the common knowledge. But since this can be very difficult to find out, neither the author nor the readers might realize the special status of the results.

type	Bub1
supertypes	Kinase – Enzyme – Protein – Molecule
subtypes	BubR1
interacts directly with	Beta2-Adaptin, Cdc20, Mad3
interacts indirectly with	Mad2, APC
associates with	Cdc20, Mad3
phosphorylates	Bub1, Bub3
localizes to	Kinetochore, Chromosome
participates in	Cell-Communication, Signal-Transduction

Figure 6: Overview over the object 'Bub1'

In the same way we can check, whether there exist papers that contradict a certain paper. That would mean that different researchers claim contradictory results. Being aware of such a contradiction might lead to a dialogue between the corresponding scientists, which might entail better and consistent results.

Are some results (or parts of it) already known? With our formal approach we can check whether a certain result, or a part of it, is already known. Results that are already considered common knowledge are usually not worth to be described as results of scientific papers (unless they contain more detailed information or if additional evidence is given). Thus it is very valuable to be able to run a check, whether a certain result is already contained in the knowledge base or not.

Furthermore a researcher might want to check, whether there exists scientific literature that has arrived at the same or similar results. Altogether our approach would help the researchers to save a lot of time, since they would not need to search "manually" for the relevant literature.

Is there a known answer for a certain question? If someone – researcher or not – has a specific question about the domain (e.g. protein interactions), then we would be able to give automatically an answer.

Indeed, there exist already systems like *MEDIE*[13] that provide some sort of answer extraction using natural language processing. But such systems have serious shortcomings: There is always a trade-off between precision and recall, and only very simple queries are allowed. Furthermore, we cannot find answers that are spread over multiple articles.

What is known about a certain object of interest? In some cases we do not want to ask a specific question, but we rather want to get an overview of a single object of interest (e.g. the protein *Bub1*). If we ask for information about such an object then we might get something as shown in Fig. 6. Such an overview could be used for a dynamic hypertext representation. This would allow us to navigate through the whole knowledge base, e.g. with an ordinary web browser. New papers that are submitted can be integrated *automatically* and thus such a web interface would be always up-to-date.

How are some objects of interest related? Instead of focusing on one single object, we might want to have an overview of the interrelations of a certain group of objects. We could extract, for example, the *interacts-with*-relations of all proteins and use this data for further examination, like the detection

[13] `http://www-tsujii.is.s.u-tokyo.ac.jp/medie/`

of clusters or hot-spots. Such examinations are already common in the research on proteins (e.g. [10], [16]), but only with restricted data. With our approach we could consider every interaction that has been published.

2.7 Outlook

We suggest an approach of using controlled natural language for making the results of scientific papers readable and – to some degree – understandable by computers. But in order to achieve this goal, there is still a lot of work to do. For example, we need an authoring tool as sketched in Sect. 2.5.3, that would support the authors of scientific papers in the creation of ACE summaries. A prototype of such a tool does already exist. Furthermore, we need tools for the definition of ontologies and for the collection and management of knowledge.

Besides all these technical requirements, there are also political ones. There must be a commitment among the scientists of the corresponding field of research – or at least among a large part of them – that scientific articles get summarized in ACE. If such a summary is optional then there is little hope that it gets established.

This is the point where the publishers and editors have to come into play. The publishers would have to make ACE summaries a mandatory part of the articles, and the editors would have to check whether these summaries are correct and complete. The creation of a formal summary should be an additional requirement to consider when writing a scientific article, besides all the requirements that already exist today (e.g. about the abstract, the keyword list, and the reference list). The formal summaries can also be seen as a robust indicator for the value of a scientific paper. Information that is already known and redundant information could be ignored automatically, and wrong statements are likely to be detected at some later point in time. Thus we could use the formal summaries to quantify and qualify the contribution of a certain author, institute, or journal.

Due to the immense benefits such a system would bring along, we believe in the great potential of our approach. It could be a first step towards better communication and persistence of biomedical knowledge.

References

[1] Abraham Bernstein, Esther Kaufmann, Christian Kaiser. *Querying the Semantic Web with Ginseng: A Guided Input Natural Language Search Engine*. Department of Informatics, University of Zurich, 2005

[2] Grady Booch, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide*, First Edition. Addison Wesley, 1998

[3] Aaron M. Cohen, William R. Hersh. *A survey of current work in biomedical text mining*. In *Briefings in Bioinformatics*, 6(1):57-71, 2004

[4] Nikolai Daraselia, Anton Yuryev, Sergei Egorov, Svetalana Novichkova, Alexander Nikitin, Ilya Mazo. *Extracting human protein interactions from MEDLINE using a full-sentence parser*. In *Bioinformatics*, 20(5):604-611, 2004

[5] Andreas Doms, Michael Schroeder. *GoPubMed: exploring PubMed with the Gene Ontology*. In *Nucleic Acids Research*, 33:W783-W786, 2005

[6] Norbert E. Fuchs, Stefan Hoefler, Kaarel Kaljurand, Tobias Kuhn, Gerold Schneider, Uta Schwertel. *Discourse Representation Structures of ACE 4 Sentences*, Technical Report ifi-2006.07. Department of Informatics, University of Zurich, 2006,
`ftp://ftp.ifi.unizh.ch/pub/techreports/TR-2006/ifi-2006.07.pdf`

[7] Norbert E. Fuchs, Kaarel Kaljurand, Gerold Schneider. *Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces*. The 19th International FLAIRS Conference (FLAIRS'2006), 2006

[8] Norbert E. Fuchs, Uta Schwertel, Rolf Schwitter. *Attempto Controlled English – Not Just Another Logic Specification Language*. In *Logic-Based Program Synthesis and Transformation*, Eighth International Workshop LOPSTR'98, Lecture Notes in Computer Science 1559, Springer, 1999,
`http://www.ifi.unizh.ch/attempto/publications/papers/LOPSTR98.pdf`

[9] Melvin Fitting. *First-Order Logic and Automated Theorem Proving*, Second Edition. Springer, New York, 1996

[10] L. Giot, J. S. Bader, C. Brouwer, A. Chaudhuri, et al. *A Protein Interaction Map of Drosophila melanogaster*. In *Science*, 302(5651):1727-1736, 2003

[11] Thomas R. Gruber. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. In *International Journal of Human-Computer Studies*, 43(5-6):907-928, 1995

[12] Lynette Hirschman, Jong C. Park, Junichi Tsujii, Limsoon Wong, Cathy H. Wu. *Accomplishments and challenges in literature data mining for biology*. In *Bioinformatics Review*, 18(12):1553-1561, 2002

[13] Stefan Hoefler. *The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0*, Technical Report ifi-2004.03. Department of Informatics, University of Zurich, 2004,
`ftp://ftp.ifi.unizh.ch/pub/techreports/TR-2004/ifi-2004.03.pdf`

[14] Deborah L. McGuinness, Frank van Harmelen. *OWL Web Ontology Language Overview*. W3C Recommendation, 2004,
`http://www.w3.org/TR/2004/REC-owl-features-20040210/`

[15] Daniele Nardi, Ronald J. Brachman. *An Introduction to Description Logics*. In *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003

[16] Benno Schwikowski, Peter Uetz, Stanley Fields. *A network of protein-protein interactions in yeast*. In *Nature Biotechnology*, 18:1257-1261, 2000

[17] Rolf Schwitter, Anna Ljungberg, David Hood. *ECOLE: A Look-ahead Editor for a Controlled Language*. In *Proceedings of EAMT-CLAW03, Controlled Language Translation*, Dublin City University, 141-150, 2003

[18] Rolf Schwitter, Marc Tilbrook. *Let's Talk in Description Logic via Controlled Natural Language*. To be presented at: Logic and Engineering of Natural Language Semantics 2006 (LENLS2006), Japan, 2006

[19] Craig W. Thompson, Paul Pazandak, Harry R. Tennant. *Talk to Your Semantic Web*. In *IEEE Internet Computing*, 9(6):75-79, 2005

[20] Mike Uschold, Michael Gruninger. *Ontologies: Principles, Methods and Applications*. In *Knowledge Engineering Review*, 11(2), 1996

[21] Alexander Yeh, Alexander Morgan, Marc Colosimo, Lynette Hirschman. *BioCreAtIvE Task 1A: gene mention finding evaluation*. In *BMC Bioinformatics*, 6, 2005

3 Querying Semantic Web Contents: A Case Study In Bioinformatics(A2/I4)

3.1 Abstract

Semantic web technologies promise to ease the pain of data and system integration in the life sciences. The semantic web consists of standards such as XML for mark-up of contents, RDF for representation of triplets, and OWL to define ontologies. We discuss three approaches for querying semantic web contents and building integrated bioinformatics applications, which allows bioinformaticians to make an informed choice for their data integration needs. Besides already established approach such as XQuery, we compare two novel rule-based approaches, namely Xcerpt - a versatile XML and RDF query language, and Prova - a language for rule-based Java scripting. We demonstrate the core features and limitations of these three approaches through a case study, which comprises an ontology browser, which supports retrieval of protein structure and sequence information for proteins annotated with terms from the ontology.

3.2 Introduction

Bioinformatics is a rapidly growing field in which innovation and discoveries often arise by the correlative analysis of massive amounts of data from widely different sources. The Semantic Web and its promises of intelligent integration of services and of information through 'semantics' can only be fulfilled in the life sciences and beyond if its technologies satisfy a minimum set of pragmatic requirements:

Ease of use A language must be as simple as possible. Users will go for a not so powerful but comfortable solution instead of a very rich language that is too complicated to use.

Platform independence Operating system idiosyncrasies are increasingly becoming a nuisance, the internet is universal, and so must be a language for the semantic web.

Tool support Nowadays, it is not enough to provide language specifications and the corresponding compilers and/or interpreters. Programmers require proper support tools like code-aware editors, debuggers, query builders and validation tools.

Scalability The volume of information being manipulated in bioinformatics is increasing exponentially, the runtime machinery of a language for integrating such data must be able to scale and cope with the processing needs of today and tomorrow.

Modularity Modularity is a very fundamental idea in software engineering and should be part of any modern programming language.

Extensibility Languages should be as user extensible as possible to accommodate unforeseen but useful extensions that users might need and be able to implement.

Declarativeness The language should be high-level and support the specification of what needs to be computed rather than how.

3.3 Data Integration in Bioinformatics

The amount of available data in the life sciences increases rapidly and so does the variety of data formats used. Bioinformatics has a tradition for legacy text-based dataformats and databases such as UniProt [2] for protein sequences, PDB [3] for 3D structures of proteins, or PubMed [6] for scientific literature.

UniProt, PDB, PubMed Today, many databases, including the above are available in Extensible Markup Language (XML)[14].

Due to its hierarchical structure, XML is a flexible data format. It is a text-based format, is human-readable, and its support for Unicode ensures portability throughout systems. Together with XML a whole family of languages[15] support querying and transformation (XPath,XQuery, and XSLT). Additionally APIs such as JDOM[16], an implementation of the Document Object Model (DOM)[17], and the Simple API for XML (SAX)[18] were developed in support of XML.

Beside the need of technologies for data handling, a major task in bioinformatics is the one of data integration. The required mapping between entities from different data sources can be managed through the use of an ontology.

Ontologies in Bioinformatics Currently there is no agreed vocabulary used in molecular biology. For example, gene names are not used in a consistent way. EntrezGene [11] addresses this problem by providing aliases. EntrezGene lists for example eight aliases for a gene that is responsible for breast cancer (*BRCAI*; *BRCC1*; *IRIS*; *PSCP*; *RNF53*; *breast cancer 1, early onset*; *breast and ovarian cancer susceptibility protein 1*; and *breast and ovarian cancer susceptibility protein variant*).

At the time of writing, searching PubMed for *PSCP* returns 2417 relevant articles. Searching for *papillary serous carcinoma of the peritoneum*, returns 89 articles. However, searching for both terms returns only 19 hits. In general, there is a pressing need in molecular biology to use common vocabularies. This need has been addressed through the ongoing development of biomedical ontologies. Starting with the GeneOntology[19] [1], the Open Biomedical Ontologies effort[20] currently hosts 59 biomedical ontologies ranging from anatomy over chemical compounds to organism specific ontologies.

Gene Ontology (GO) A core ontology is the Gene Ontology[1], which contains over 20000 terms describing biological processes, molecular functions, and cellular components for gene products. The biological process ontology deals with biological objectives to which the gene or gene product contributes. A process is accomplished via one or more ordered assemblies of molecular functions. The molecular function ontology deals with the biochemical activities of a gene product. It describes what is done without specifying where or when the event takes place. The cellular component ontology describes the places where a gene product can be active. The GO ontologies have become a de facto standard and are used by many databases as annotation vocabulary and are available in various formats: flat files, the Extensible Mark-up Language (XML), the resource description format (RDF), and as a MySQL database.

[14] `www.w3.org/XML/`
[15] `www.w3.org/TR`
[16] `www.jdom.org/`
[17] `www.w3.org/DOM/`
[18] `www.saxproject.org/`
[19] `www.geneontology.org`
[20] `obo.sourceforge.net`

3.4 Case Study: ProteinBrowser

Biological databases are growing rapidly. Currently there is much effort spent on annotating these databases with terms from controlled, hierarchical vocabularies such as the Gene Ontology. It is often useful to be able to retrieve all entries from a database, which are annotated with a given term from the ontology. The ProteinBrowser use-case shows how typically one needs to join data from different sources. The starting point is the Gene Ontology (GO), from which a hierarchy of terms is obtained. Using the Gene Ontology Annotation (GOA) database, the user can link GO terms to the UniProt identifiers of proteins that have been annotated with biological processes, molecular functions, and cellular components. After choosing a specific protein, the user can, remotely, query additional information from the UniProt database, for example the sequence of the protein. In turn, the PDB database can be remotely queried for still additional information. Finally, using the PubMed identifier of the publication in which the structure of the protein was published, one can query PubMed and obtain the title and abstract of the publication.

As shown in Fig. 7, the ProteinBrowser example is specified by the following workflow:

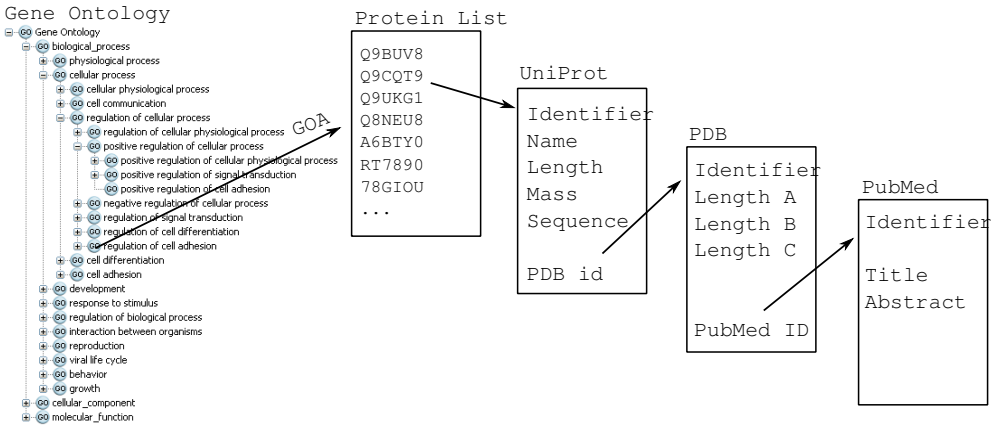


Figure 7: ProteinBrowser Workflow: from GO to PubMed via GOA, UniProt and PDB

1. A term is chosen from the Gene Ontology tree. The Gene Ontology exists in various formats: MySql database, XML, RDF.

2. All relevant proteins associated through the GOA[21] database are listed.

3. A protein is chosen from the list

4. UniProt is queried for information about this protein. The protein's name, its sequence length, mass, sequence, and corresponding PDB identifier can be retrieved by querying the XML file linked by the following parametrized URL:
   `http://www.ebi.uniprot.org/entry/⟨UniprotId⟩?format=xml&ascii`

5. PDB is queried for additional information. The three lengths $width$, $height$ and $depth$ and the PubMed identifier of the publication in which the structure was described, can be obtained by

[21] `http://www.ebi.ac.uk/GOA/`

querying the XML file linked by the following parametrized URL:
`http://www.rcsb.org/pdb/displayFile.do?fileFormat=XML&structureId=⟨PDBid⟩`

6. Retrieve PubMed abstract title and text where the structure was published. This uses the Pubmed ID (if available) and queries the website of NCBI with the PubMed Id at this address:
`http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&retmode=xml&rettype=full&id=⟨PubM`

As shown in Fig. 8, this workflow involves accessing local and remote databases, in the form of files, possibly in XML format and of 'pragmatic' web-services in the form of parametrized URLs linking to XML files (also known as REST-style Web Services).

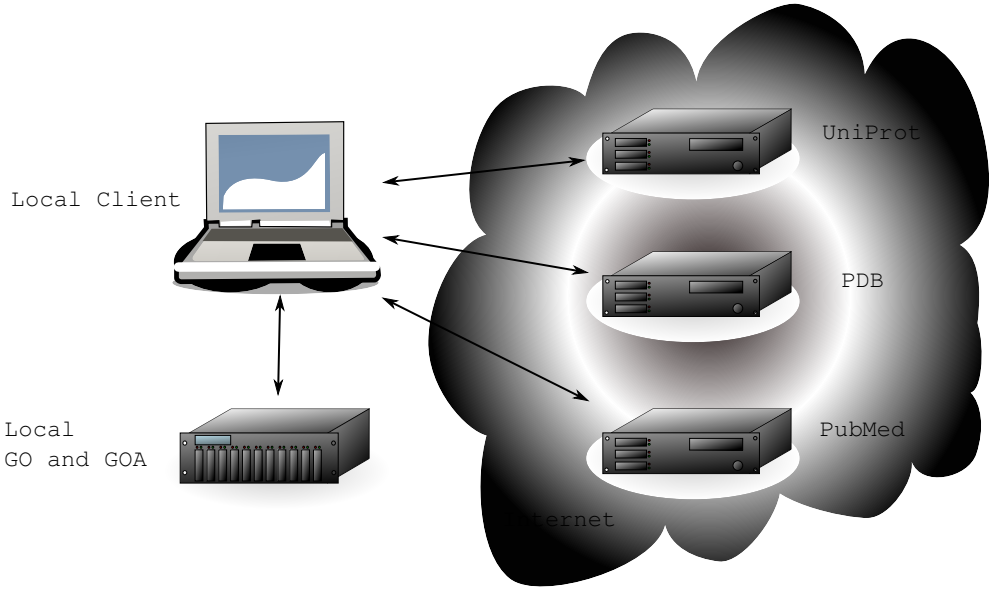


Figure 8: ProteinBrowser: integrates data from GO, UniProt, PDB and PubMed

We will compare three approaches to implement this workflow. The first is based on a novel hybrid object-oriented and declarative programing language, Prova. The second is based on standard XML technologies such as XQuery and XPath. The third is based on a novel declarative query language for XML documents: Xcerpt.

Prova	`http://www.prova.ws`
XQuery/XPath	`http://www.w3.org`
Xcerpt	`http://www.xcerpt.org`

3.4.1 Prova

Prova [10] is a rule-based Java scripting language. The use of rules allows the declarative specification of integration needs at a high-level, separately from implementation details. The transparent integration of Java caters for easy access and integration of database access, web services, and many other Java services. This way Prova combines the advantages of rule-based programming and object-oriented programming. Prova satisfies the following design goals:

- Combine the benefits of declarative and object-oriented programming;
- Merge the syntaxes and semantics of Prolog, as rule-based language, and Java as object-oriented languages;
- Expose logic as rules;
- Access data sources via wrappers written in Java or command-line shells like Perl;
- Make all Java API from available packages directly accessible from rules;
- Run within the Java runtime environment;
- Be compatible with web- and agent-based software architectures;
- Provide functionality necessary for rapid application prototyping and low cost maintenance.

3.4.2 Workflow solved with Prova

The Prova code closely resembles a declarative logic program. Rules are written down in the form `conclusion :- premise` where `:-` is read 'if'. Instead of relying on an internal knowledge base, which needs to be loaded entirely into memory, Prova can access external knowledge wrapped as predicates. Thus there is a clean separation between the details needed to access the external data and the way this data is joined in the workflow. Prova applies so-called backward-chaining to evaluate queries.

Wrapping the Gene Ontology and the Gene Ontology Annotation For the Prova implementation of the ProteinBrowser we use the Gene Ontology and the protein annotations in their relational database format. Accessing databases from Prova is very simple:

Listing 1: Wrapping the Gene Ontology database and the isa relationship.

```
% Imports some utility functions
:-eval(consult("utils.prova")).

% Define database location
location(database,"GO","jdbc:mysql://server","guest","guest").

% T2 is-a T1 if there is a corresponding entry in the term2term table of the database
isaDB(T2,T1) :-
  dbopen("GO",DB),
  concat(["term1_id=",T1," and relationship_type_id=2"],WhereClause),
  sql_select(DB,term2term,[term2_id,T2],[where, WhereClause]).

% A term T is-a T
isa(T,T).

% Recursive definition of is-a: A term T2 is a T1 if T3 is a T1 and T2 is a T3
isa(T2,T1) :-
  isaDB(T3,T1),
  isa(T2,T3).
```

After importing some utility predicates for connecting to databases, the $location$ predicate is used to define a database location, the $dbopen$ predicate is used to open a connection to the database, and the sql_select predicate provides a nice and practical declarative wrapping of the select statement of relational databases. In order to obtain all sub-terms of a given term, we simply compute the transitive closure of the subterm relationship defined by the recursive predicate isa.

Finally, in order to retrieve the UniProt identifiers corresponding to a given gene ontology term, we need the following predicate:

Listing 2: Wrapping the Gene Ontology Annotation database

```
name2UniProtId(Term,UniProtId) :-
        dbopen("GO",DB),
        concat(["uni.GOid = ", Term],Where),
        concat(["go.term as term, goa.goa_human as uni"],From),
        sql_select(DB,From,['uni.DB_Object_ID',UniProtId],[where,Where]).
```

Wrapping UniProt, PDB and Medline The three databases UniProt, PDB and Medline can be remotely accessed through a very simple web interface: a parametrized URL links to an XML file containing the relevant information for a given identifier. The following three predicates wrap the downloading and parsing of the XML files in a few lines:

Listing 3: Wrapping UniProt, PDB and Medline

```

urlUniProtPrefix("http://www.ebi.uniprot.org/entry/")
urlUniProtPostfix("?format=xml&ascii")
urlPDB("http://www.rcsb.org/pdb/displayFile.do?fileFormat=XML&structureId=")
urlPubMed("http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed
&retmode=xml&rettype=full&id=")

% Query UniProt by giving a UniProt Id and getting the length, mass, sequence, and PDB id
queryUniProt(UniProtId,Name,Length,Mass,Sequence,PDBId):-
        urlUniProtPrefix(URLpre),
        urlUniProtPostfix(URLpost),
        concat([URLpre,UniProtId,URLpost],URLString),
        retrieveXML(URLString,Root),
        children(Root,"entry",EntryNode),
        children(EntryNode,"protein",ProteinNode),
        descendantValue(ProteinNode,"name",Name),!,
        descendant(EntryNode,"sequence",SequenceNode),
        nodeAttributeByName(SequenceNode,"length", Length),
        nodeAttributeByName(SequenceNode,"mass", Mass),
        nodevalue(SequenceNode,Sequence).

% Query PDB by giving a PDB Id and getting three lengths a,b,c and a PubMed id of a publication
queryPDB(PDBId,LA,LB,LC,PMID):-
        urlPDB(URL),
        concat([URL,PDBId],URLString),
        retrieveXML(URLString,Root),
        descendantValue(Root,"PDBx:length_a",LA),!,
        descendantValue(Root,"PDBx:length_b",LB),!,
        descendantValue(Root,"PDBx:length_c",LC),!,
        descendantValue(Root,"PDBx:pdbx_database_id_PubMed",PMID).

% Query pubMed by giving a PubMed Id and getting the text of the abstract
queryPubMed(PMID,AbstractTitle, AbstractText):-
        urlPubMed(URL),
        concat([URL,PDBId],URLString),
        retrieveXML(URLString,Root),
        descendantValue(Root,"ArticleTitle",AbstractTitle),!,
        descendantValue(Root,"AbstractText",AbstractText),!.
```

The previous predicates use the following utility predicates:

Listing 4: XML retrieval

```
retrieveXML(URLString,Root):-
        URL = java.net.URL(URLString),
        Stream = URL.openStream(),
        ISR = java.io.InputStreamReader(Stream),
```

```
5         XMLResult = XML(ISR),
6         Root = XMLResult.getDocumentElement().
```

The $retrieveXML$ predicate downloads an XML file from a specified URL, and returns the root DOM (Document Object Model) tree representation of the XML file.

In the following, a set of predicates provide functionality to query nodes and values from the DOM tree:

Listing 5: XML Querying

```
1
2 % Simulates an XPath traversal.
3 descendantsValue(Current,Name,Value):-
4         descendants(Current,Name,Node),
5         nodeValue(Node,Value),!.
6
7 % Definition of a descendant (any depth), similar XPath: //*
8 descendants(Node,Node).
9 descendants(Element,S2):-
10        children(Element,S1),
11        descendants(S1,S2).
12
13 % Definition of a descendant with given name, similar XPath: //Name
14 descendants(Node,Name,Descendant):-
15        descendants(Node,Descendant),
16        nodeName(Descendant,Name).
17
18 % Definition for a direct child, similar XPath: /*
19 children(Element,Child):-
20        Childs = Element.getChildNodes(), % Java call returning direct children
21        Childs.nodes(Child).
22
23 % Definition of a child with a given name, similar XPath: /Name
24 children(Node,Name,Child):-
25        children(Node,Child),
26        nodeName(Child,Name).
27
28 nodeName(Node,Name):-
29        Name = Node.getNodeName().
30
31 nodeValue(Node,Value):-
32        Data = Node.getFirstChild(),
33        Raw = Data.getNodeValue(),
34        Value = Raw.trim().
```

Assembling the Workflow Now that we have wrapped the GO and GOA databases, as well as the remote XML ressources for UniProt, PDB and PubMed. We can proceed with the assembly of the ProteinBrowser workflow:

Listing 6: Workflow

```
1 workflowStep1(GoTermName,UniProtId):-
2   name2term(GoTermName,GoTerm),
3   isa(GoTerm,Descendant),
4   name2UniProtId(Descendant,UniProtId),
5   java.lang.System.out.println(UniProtId).
6
7 workflowStep2(UniProtId):-
8   queryUniProt(UniProtId,Name,Length,Mass,Sequence,PDBId),
9   java.lang.System.out.println(Name),
10  java.lang.System.out.println(Length),
11  java.lang.System.out.println(Mass),
12  java.lang.System.out.println(Sequence),
13  queryPDB(PDBId,LA,LB,LC,PMID),
```

```
14   java.lang.System.out.println(LA),
15   java.lang.System.out.println(LB),
16   java.lang.System.out.println(LC),
17   queryPubMed(PMID,AbstractTitle, AbstractText),
18   java.lang.System.out.println(AbstractTitle),
19   java.lang.System.out.println(AbstractText).
20
21   % Given the name N, get the term id T
22 name2term(N,T) :-
23   dbopen("GO",DB),
24   concat(["name like ",N],WhereClause),
25   sql_select(DB,term,[id,T],[where, WhereClause]).
```

The first step is simply to enumerate all UniProt identifiers $UniProtId$ annotated with terms and subterms of a given Gene Ontology term $GoTermName$. The second step uses the chosen protein UniProt identifier and starts a cascade of three remote queries to the UniProt, PDB and PubMed web sites. All relevant information collected is printed out.

3.4.3 XQuery and XPath

XPath[22] allows the user to address certain parts of an XML document. Beside many applications it is used in XQuery, which is a declarative query- and transformation language for semi-structured data. It is widely used to formulate queries on RDF and XML documents. These documents can be provided as XML files, as XML views onto a XML database or created by a middleware. XQuery 1.0 is a W3C Candidate Recommendation and is already supported by many software vendors [23] (e.g. IBM DB2, Oracle 10g Release 2, Tamino XML Server).

3.4.4 The Workflow Solved with XQuery

An XQuery implementation of the workflow works on XML data only and can be realized with all program logic specified as XQuery. We note, that XQuery as described in the language standard, is expressive enough to aggregate data from different data sources, locally or remotely.

Recursive traversal of the Gene Ontology With XQuery the recursive traversal of the GO has to be programmed explicitely. In listing 7 the functions `local:getDescendants` and `local:getChildren` show how this simple recursion can be specified with XQuery. The locally available GO OWL file is loaded using the `doc()` function, which also works for remote resources of plain XML content. It takes approximately 45s to load the GO with GOA included and build the DOM tree. By using XQuery from within Java it is possible to preserve the DOM tree, so that it only has to be loaded once.

Listing 7: Recursive XQuery to create the transitive closure over the sub-class relations.

```
1
2  declare function local:getChildren( $term , $context)
3  {
4          for $my_term in $context//go:term
5          where $my_term/go:is_a/@rdf:resource = $term/@rdf:about
6          return
7              $my_term
8  };
9
10 declare function local:getDescendants( $term, $context)
11 {
12     for $my_term in local:getChildren($term, $context)
```

[22] http://www.w3.org/TR/xpath

[23] http://www.w3.org/XML/Query/#implementations

```
13        return
14        <descendants>
15        {
16            local:getDescendants($my_term , $context), $my_term
17        }
18        </descendants>
19 };
```

Assembling the Workflow Listing 8 shows the complete workflow as a batch process. Given a GO accession number like ”GO:0000001” an XML document is created which contains all proteins associated with the specified term or any of its child terms. For all these proteins additional information is retrieved from UniProt. Further, database references to structural data in PDB is used, if found in UniProt. For the interactive browser these parts are separated and the functions are called once the GO term or protein is selected in the GUI.

Listing 8: Recursive XQuery to aggregate proteins associated with a GO term or any of its children. The result gets enriched with Uniprot and PDB data.

```
1  xquery version "1.0";
2  declare namespace go = "http://www.geneontology.org/dtds/go.dtd#";
3  declare namespace rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#";
4  declare namespace fn = "http://www.w3.org/2005/xpath-functions";
5  declare namespace uniprot = "http://uniprot.org/uniprot";
6  declare namespace PDBx = "http://deposit.pdb.org/pdbML/pdbx.xsd";
7  declare namespace xsi="http://www.w3.org/2001/XMLSchema-instance";
8
9  declare variable $GO as xs:string external;
10
11 (: function from www.w3c.org :)
12 declare function local:distinct-nodes-stable ($arg as node()*) as node()*
13 {
14    for $a at $apos in $arg
15    let $before_a := fn:subsequence($arg, 1, $apos - 1)
16    where every $ba in $before_a satisfies not($ba is $a)
17    return $a
18 };
19
20 declare function local:getChildren( $term , $context) { ... };
21
22 declare function local:getDescendants( $term, $context) { ... };
23
24 declare function local:queryUniprot($uniprotID) { ... };
25
26 declare function local:queryPDB($pdbID) { ... };
27
28 (: Construct a result set for one GO term :)
29 <terms>
30 {
31     let $root :=doc("/data/go_200605-assocdb.rdf-xml")
32     for $term in $root//go:term
33     where $term/go:accession/text() = $GO
34     return
35         <result query_term_acc="{$term/go:accession/text()}">
36          {
37          let $terms := ($term, local:getDescendants($term,$root))
38          for $d_term in $terms
39          return
40              for $dbxref in $d_term//go:dbxref
41              where $dbxref/go:database_symbol/text()="UniProt"
42              return
43                      for $uniprot_id in local:distinct-nodes-stable($dbxref/go:reference)
44                      return
45                             local:queryUniprot($uniprot_id/text())
46         }
47         </result>
```

```
}
</terms>
```

Obtain additional information for proteins For all proteins identified, the UniProt database is queried selecting data sets for a specific UniProt identifier (see listing 9). Additional information from the PDB is retrieved as shown in listing 10.

Listing 9: Querying the Uniprot database with XQuery for information on the names, sequence, sequence length, sequence mass and structures of a protein

```
declare function local:queryUniprot($uniprotID)
{
    let $url := concat(concat("http://www.ebi.uniprot.org/entry/", $uniprotID), ...
     ..."?format=xml&amp;ascii")
    for $entry  in doc($url)//uniprot:entry
    let $sequence:= $entry/uniprot:sequence
    return
        <protein uniprot_id="{$uniprotID}">
            {
            for $name in $entry/uniprot:protein//uniprot:name
            return
                <name>{$name/text()}</name>
             }
            <sequence_length>{$sequence/@length}</sequence_length>
            <sequence_mass>{$sequence/@mass}</sequence_mass>
            <sequence>{ $sequence/text() }</sequence>
             {
                for $pdbID in $entry//uniprot:dbReference[@type="PDB"]/@id
                return
                   local:queryPDB($pdbID)
             }
        </protein>
};
```

Listing 10: Querying the PDB database with XQuery.

```
declare function local:queryPDB($pdbID)
{
    let $url := concat("http://www.rcsb.org/pdb/downloadFile.do? ...
     ...fileFormat=xml&amp;compression=NO&amp;structureId=",$pdbID)
    for $item  in doc($url)/PDBx:datablock/PDBx:cellCategory/PDBx:cell
    return
        <pdb_structure pdb_id="{$pdbID}">
            <length_a>{$item/PDBx:length_a/text()}</length_a>
            <length_b>{$item/PDBx:length_b/text()}</length_b>
            <length_c>{$item/PDBx:length_c/text()}</length_c>
        </pdb_structure>
};
```

3.4.5 Xcerpt

Xcerpt [12] is a declarative rule based query- and transformation language for semi-structured data in general and for RDF and XML in particular. Xcerpt does not natively query relational data bases, but relies on the XML, RDF or OWL serializations of the Gene Ontology and the Protein Databank. These serializations in general being graph structured and highly heterogeneous, Xcerpt provides a comfortable way to query possibly incomplete subpatterns of the data.

Xcerpt builds upon *simulation unification* and rule chaining for program evaluation. Xcerpt uses three kinds of terms to carry out its computations: *data terms*, *query terms* and *construct terms*. Data

terms are semi-structured data serving as an abstraction from various tree- and graph shaped data-formats such as RDF and XML. Dataterms can be used to represent any kind of semi-structured data, while still taking care of XML specificities such as attributes, namespaces and references.

Query terms are data terms augmented by logical variables and enriched by constructs that allow the specification of various forms of incompleteness, which are used to match highly heterogeneous data. Incompleteness specifications include incompleteness in depth (the descendant construct and arbitrary length traversal path expressions), incompleteness in breadth (there may be more subterms in the queried data than which are specified by the query term) and optional subterms. Query terms are matched with data terms via simulation unification to produce *substitution sets* (sets of sets of variable bindings). Substitution sets are then applied to construct terms by filling in the bindings for variable occurrences.

3.4.6 The Workflow solved with Xcerpt

In order to select all proteins produced by a certain term referenced in the Gene Ontology, the following Xcerpt rules could be used. Since we are not only interested in the proteins produced by exactly the term provided by the user, but also in those proteins which are produced by processes which are subterms of the given term, and in additional information obtained from UniProt, PDB and PubMed, the task is split into several parts:

Extracting subterm relationships from the Gene Ontology Database In a first step (realized by Listing 11), the direct subterm relationships are extracted from the database. They are retrieved from the `is_a` elements given in the Gene Ontology. In the special `attributes`-element the form of the `rdf:resource`-attribute of the `is_a`-element is specified, demanding that it ends with a GO-Term identifier. Note that since Xcerpt programs are evaluated in a backward chaining manner, the binding of the logical variable `Term2` is passed on from the second and third rule below. Curly braces in the query term indicate that the order in which the siblings occur within the data is not important. This concept is called *Incompleteness with respect to order*. Double curly braces are used to allow also further siblings in the data besides those explicitly specified – this concept is known as *incompleteness in breadth* in Xcerpt. Xcerpt's `desc` construct matches with descendants of the enclosing term that exhibit the specified pattern (*incompleteness in depth*). Since there is no enclosing element for the `go:term` element in the query term, it matches with all data nodes that have at least a `go:accession` and a `go:is_a` sub-element (of the specified form).

Listing 11: Extracting subterm relationships from the Gene Ontology

```
CONSTRUCT
  subterm { var Term1, var Term2 }
FROM
  in {
     resource {
       "http://archive.godatabase.org/full/2006-05-01/
        go_200605-assocdb.rdf-xml.gz" },
     desc go:term {{
       go:accession { var Term1 },
       go:is_a{{
         attributes{{
           rdf:resource {
             "http://www.geneontology.org/go#" ++ var Term2 }
       }}
     }}
  }
END
```

Computing the transitive closure of the subterm relationship In a second rule (given in Listing 12), the transitive closure of the subterm relationship is computed. Since all direct subterms are considered as transitive subterms, the second disjunct of the body of this second rule matches with the head of the first rule.

Listing 12: Computing the transitive closure of the subterm-relationship with an Xcerpt rule

```
CONSTRUCT
  transitive_subterm { var Term1, var Term3 }
FROM
  or {
    and {
      subterm { var Term1, var Term2 },
      transitive_subterm { var Term2, var Term3 }
    },
    subterm { var Term1, var Term3 }
  }
END
```

Finding all the proteins associated with a term of the Gene Ontology In the third rule (see Listing 13) for each of the subterms of the given term `Term`, the associated proteins are looked up in the GOA database and rendered as a list of links to their Uniprot entries in an HTML file. The binding for the variable `Term` is provided by the user as a command line parameter (e.g. `xcerpt -D Term=GO:0051260`, where `GO:0051260` is the identifier of *protein homooligomerization*).

The first conjunct of the body of this rule matches with the second rule above and passes the `Term`-variable on to the head of the second rule. In this way, all of its subterms are bound to the variable `SubTerm`.

The second conjunct of the rule looks up all associated proteins for the subterm, which have a Gene Ontology database symbol of type `UNIPROT`. Each of these proteins is bound to the variable `PROTEIN`.

Note that also the second conjunct of the query term may match multiple times with the database for a single binding of the variable `SubTerm`, thus producing a set of pairs of variable bindings in which `SubTerm` is always bound to the same variable given in the query, and `Protein` is bound once for each protein produced by the given concept.

In the construct part of the rule (framed by the keywords `GOAL` and `FROM`) the proteins are grouped by the subterms which they are associated with in the Gene Ontology. This is achieved by the grouping construct `all`. The string-concatenation function "`++`" is used to construct the URL pointing at the Uniprot entry. The construct term is a template of the HTML page rendered by the browser to form part of the user-interface.

Listing 13: Constructing an HTML list of proteins for a GO term

```
GOAL
  html [
    head [ title [ "Proteins produced by" ++ var Term ] ],
    body [
      all span [
        h3 [ "Proteins produced by the subterm " ++ var SubTerm ],
        ul [
          all li [
            attributes{ href {
              "http://www.ebi.uniprot.org/entry/" ++ var Protein ++
                "?format=xml&ascii" } },
            var Protein ]
        ]
      ]
  ] ]
FROM
```

```
  and {
    transitive_subterm { var SubTerm, var Term },
    in {
      resource {
        "http://archive.godatabase.org/full/2006-05-01/
         go_200605-assocdb.rdf-xml.gz" },
      desc go:term{{
        go:accession{ var SubTerm },
        go:association{{
          go:gene_product{{
            desc go:database_symbol{ "UNIPROT" },
            desc go:reference{ var Protein }
          }}
        }}
      }}
    }
  }
END
```

Extracting relevant information about Proteins from the Uniprot and PDB Files Xcerpt's patterns are well-suited to extract the name, length, mass and the sequence of amino acids for a given protein from the UniProt database and to reassemble them within an HTML fragment as specified in Listing 14. The second conjunct of the same rule is used to additionally extract information from the PDB database about the physical dimensions of the crystals of the Protein and PubMed identifiers of research papers dealing with the given protein. This data is to be combined with the information from UniProt. Note that the `PDB_ID` is extracted from the UniProt database, which means that the first conjunct is evaluated before the second one. The rule could be called via a system call from within a CGI script. Many of the PDB files about proteins additionally supply PubMed identifiers of research articles treating the protein, but this is not mandatory. Xcerpt's `optional`-construct allows to select optional data that does not have to be present for the query to succeed. Since their may be multiple references to PubMed identifiers, these references are wrapped into an unordered +HTML+ list using the grouping construct `all`. These references could be easily encoded as hyperlinks in a similar way as in listing 13, which has been omitted for brevity.

Listing 14: Combining information from the PDB and the UniProt database for the same Protein

```
CONSTRUCT
  div [
    h3 [ 'Information about protein', span[ var Protein ] ],
    p [ "Name: " ++ var Name ],
    p [ "Length: " ++ var Length ],
    p [ "Mass: " ++ var Mass ],
    p [ "Sequence: " ++ var Sequence ],
    p [ "length_a: " ++ var LengthA ],
    p [ "length_b: " ++ var LengthB ],
    p [ "length_c: " ++ var LengthC ],
    optional p [ 'PubMed References', ul [ all li[ var PubMedID ] ] ]
  ]
FROM
  and {
    in {
      resource {
        "http://www.ebi.uniprot.org/entry/" ++ var SubTerm ++
          "?format=xml&ascii" },
      entry {{
        protein {{ name {{ var Name }} }},
        sequence {{
          attributes {{ length { var Length }, mass { var Mass } }},
          var Sequence
        }}
        dbReference { attributes {{
```

```
26            type { "pdb accesion" },
27            value { var PDB_ID }
28          }} }
29        }}
30      },
31      in {
32        resource {
33          "http://www.rcsb.org/pdb/downloadFile.do?fileFormat=xml&
34            compression=NO&structureId=" ++ var PDB_ID },
35        PDBx:datablock {{
36          desc PDBx:cell {{
37            PDBx:length_a{{ var LengthA }},
38            PDBx:length_b{{ var LengthB }},
39            PDBx:length_c{{ var LengthC }}
40          }},
41          optional PDBx:pdbx_database_id_PubMed { var PubMedID }
42      }
43    }
44  END
```

Retrieving the PubMed Abstract and Title The final step in the workflow of the Protein Browser consists of retrieving the PubMed abstract and title for a given PubMed identifier retrieved by the rule in Listing 14. The PubMed identifiers may either be queried directly from the pdb file of a given protein or they may originate from the results of the previous rule. In Listing 15 the second alternative is presented.

Listing 15: Retrieval of Abstract and Titles of PubMed entries

```
1  CONSTRUCT
2    html [ head [ title [ 'Articles for Protein' ++ var Protein ] ],
3           body [
4             all p [ h3 [ var Title ], div [ var Abstract ] ]
5           ]
6    ]
7  FROM
8    and (
9      div [[ h3 [[ span [ var Protein ] ]],
10            p [[ ul [[ li [ var PubMedId ] ]] ]]
11     ]],
12     in {
13       resource { ...
              ...'http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&retmode=xml&rettype=full&id='...
              ... ++ var PubMedId },
14       PubMedArticle {{
15         desc AbstractText { var Abstract },
16         desc ArticleTitle { var Title }
17       }}
18     }
19 END
```

The given rule finds all PubMed identifiers from the previously created HTML fragment, retrieves the PubMed documents for these articles and assembles a new HTML page containing a list of article titles and abstracts.

3.5 Comparison

In the following, we compare the three approaches according to several criteria. Some criteria are subjective, for example how easy or difficult it is to learn and use the approach. Other criteria are of a pragmatic nature and relate to the availability of supporting tools like editors and debuggers. From a technical point of view, it is also important to evaluate the scalability, modularity, and extensibility of an approach.

Learning curve Prova requires basic understanding of both Prolog and Java. This might make it more complicated to understand than Java or Prolog separately. The Prova syntax integrates aspects from both paradigms in a very elegant way. If one assumes basic knowledge in both Java and Prolog, Prova is then a good way to profit from both worlds. XQuery adapts standard programming paradigms like FOR loops or IF-THEN-ELSE statements and uses XPath to address nodes in the Document Object Model (DOM) tree. Nevertheless the syntax and especially the usage of functions requires some time to learn. Xcerpt can be used to query and transform any XML application, thus also XML serializations of RDF and Topic Maps. Therefore it is very well-suited for data integration. Being a very declarative pattern- and rule-based language, potential errors are kept to a minimum and authoring queries in Xcerpt is straightforward. Xcerpt is especially easy to learn for users with experience in logic programming or with pattern based query languages such as Query By Example or to a certain extent XPath.

Platform independence Prova is Java-based and as such is platform-independent. XQuery and XPath standard implementations are available as libraries written in Java[24] and can be used from any platform which supports Java. Additionally many database systems come with XPath or XQuery build in. Xcerpt is currently implemented in Haskell and compiled with the Glasgow Haskell Compiler, which is available for Linux, Solaris, Windows, FreeBSD and MacOS X. Thus Xcerpt can be used on any of these platforms. Future versions of Xcerpt will be written in Java to further increase platform independence.

Availability Prova is a GNU Lesser General Public License (LGPL) open source project and thus can be used in any context, it can be freely downloaded from `www.prova.ws`. XQuery, XPath and RQL are available within commercial products or for free under the Berkeley Software Distribution (BSD) license. Xcerpt is current at a prototype stage of development and is available at `www.xcerpt.org` under the terms of the GNU General Public License.

Tool support Prova, because of its relative youth, has almost no support for editing or debugging tools. XPath is simple enough to be written with a plain text editor. However it is strongly recommended to use specialized editors for XQuery. There exist mature tools for several software platforms which come with editing support, validation and debugging functionalities. Xcerpt is accompanied by a visual query authoring and execution tool called *visXcerpt*. It features a web-based graphical interface, running on top of a web server such as the Apache HTTP server[25] and allows to dynamically browse both XML data and the Xcerpt rules. Support for debugging and code completion in Xcerpt is not available yet.

Scalability Prova is arguably at most as scalable as Java and its libraries. Java is itself a very mature language in terms of performance. Starting with version 1.3, the Java Virtual Machine has been based on HotSpot, a technology that allows dynamic compilation of performance bottlenecks at execution time. For this reason Java itself cannot be thought as an interpreted language. So even though the rule engine behind Prova is essentially interpreted, all the heavy duty work can be delegated to Java classes and one can thus expect near-compiled performance. On a machine powered by a Intel Xeon 3GHz, Saxon's XQuery engine needs approximately 50 seconds to prepare the 300 MB large Gene Ontology RDF file for XQuery execution. Xcerpt programs are currently being evaluated in memory. Thus it is not yet possible to process large amounts of XML data. With 512 megabytes of random access memory, an XML file of a size at most 40 megabytes can be effectively processed. Research geared toward more efficient implementations is being carried out.

[24] e.g. http://saxon.sourceforge.net/

[25] `http://www.apache.org/`

Modularity Prova inherits the modularity of Java. XQuery allows for user-defined functions that can be used to modularize the code and improve its maintainability. Xcerpt is being developed with a module system.

Extensibility Prova is based on Java and can construct Java objects and call any of their methods. Xcerpt being available under an open source license, it can be easily extended and adapted to ones own needs.

3.6 Discussion and Conclusion

In this article we have shown how the combination and integration of biological data from different resources on the Web may be realized with different technologies. XML is a suitable way for sharing and exchanging data across different systems interconnected over the Internet. XML query languages are an accepted means for extracting relevant information and for processing and transforming XML data.

XML and best practices. Biological data is often stored in relational database engines and must be serialized before it can be processed by XML query languages. Additionally, huge amounts of biological data are already available and transferring entire databases over the network takes a significant amount of time. As a result, XML queries should be processed close to the data they operate on as far as possible, taking advantage of relational database indexes. Several commercial database products already support the local execution of XQuery programs. To minimize transfer and processing time, only the results of locally executed queries should be transfered over the network as XML. In many cases, however, queries cannot be executed locally in their entirety, because joins over entries located at different sites are necessary.

As can be seen in the exemplary workflow described in section 3.4, several transformations of XML data may be stringed together to achieve complex restructuring tasks. In such cases it is advisable to minimize intermediate serializations of XML data independently of the query language being used. In other words embedding several Xcerpt, XQuery or XSLT programs taking XML as input and producing XML as output in a host language is inefficient when compared to joining these programs to a single one, because processing time is lost for parsing and serializing XML data.

The advantages of using XML query languages for data integration versus the direct usage of relational databases increase with the amount of different data sources that must be integrated and with the degree of heterogeneity of the encountered data. The more heterogeneous the data, the harder it is to fit it into a relational database schema. Moreover, XML query languages (especially Xcerpt) provide a rich set of language constructs to deal with various kinds of heterogeneity of the data, which means that several SQL queries operating on a relational database can be combined to form a single Xcerpt query on XML data.

In picking the right XML technology for a bioinformatics project, maturity of the language is an important issue. Xcerpt being a research prototype, is currently not recommended for use in large projects. On the other hand XQuery is a W3C recommendation and several robust implementations are already available.

Beyond XML ? It is not yet clear if XML will eventually become the universal format for data exchange. Relational databases, flat files, and other idiosyncratic formats might subsist and limit, in practice, the applicability of pure XML query languages. We have shown how practical Prova is for

assembling workflows involving heterogeneous sources of data. Prova is also able to delegate XML processing tasks to XQuery which has itself a Java implementation based on the Saxon library [26]. Xcerpt will also be eventually reimplemented in Java, and thus it will also be possible in the future to run Xcerpt queries from a Prova program. It can be argued that the need for a generic and possibly declarative programming language will remain. Simply because from a pragmatic point of view, there will always be some tasks that will be simply too cumbersome to deal with any specialized languages. A user should always be able to fall-back to a standard programming approach.

Conclusion In all cases, it is clear that independently of the technologies used, the trend is toward remote querying of data. Maintaining and synchronizing local databases is cumbersome and should not be necessary. As we have seen, several databases like UniProt, PDB and PubMed offer their data through URL links in XML format. Prova, Xquery/Xpath and Xcerpt are ready to process them.

Acknowledgements: We acknowledge the financial support of the EU projects Sealife (FP6-IST-027269) and REWERSE (FP6-IST-506779).

References

[1] The Gene Ontology (GO) project in 2006. *Nucleic Acids Research*, 34(Database issue):D322–6, 12 2005.

[2] Bairoch A, Apweiler R, Wu CH, Barker WC, Boeckmann B, Ferro S, Gasteiger E, Huang H, Lopez R, Magrane M, Martin MJ, Natale DA, O'Donovan C, Redaschi N, and Yeh LS. The Universal Protein Resource(UniProt). *Nucleic Acids Res.*, 33:D154–159, 2005.

[3] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Res*, 28(1):235–242, 2000.

[4] F. Bry and P. Patranjan. Reactivity on the web: Paradigms and applications of the language xchange. In *Proceedings of 20th Annual ACM Symposium on Applied Computing (SAC'2005)*. ACM, March 2005.

[5] J. Dietrich, A. Kozlenkov, M. Schroeder, and G. Wagner. Rule-based agents for the semantic web. *Journal on Electronic Commerce Research Applications*, 2(4):323–38, 2003.

[6] Wheeler DL, Chappey C, Lash AE, Leipe DD, Madden TL, Schuler GD, Tatusova TA, and Rapp BA. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.*, 28:10–4, 2000.

[7] Ernest Friedman-Hill. *Jess in Action Java Rule-based Systems*. Manning, 2003.

[8] GeneOntologyConsortium. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Res.*, 1(32):D258–61, 2004.

[9] A. Kozlenkov and M. Schroeder. PROVA: Rule-based Java-scripting for a bioinformatics semantic web. In E. Rahm, editor, *International Workshop on Data Integration in the Life Sciences DILS*, Leipzig, Germany, 2004. Springer.

[26] `http://saxon.sourceforge.net/`

[10] Alexander Kozlenkov and Michael Schroeder. PROVA: Rule-based Java-Scripting for a Bioinformatics Semantic Web. In E. Rahm, editor, *International Workshop on Data Integration in the Life Sciences DILS*, Leipzig, Germany, 2004. Springer.

[11] Donna Maglott, Jim Ostell, Kim D Pruitt, and Tatiana Tatusova. Entrez Gene: gene-centered information at NCBI. *Nucleic Acids Res*, 33(Database issue):D54–D58, 2005.

[12] Sebastian Schaffert and François Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proceedings of Extreme Markup Languages 2004, Montreal, Quebec, Canada (2nd–6th August 2004)*, 2004.

4 Prova: Rule-based Java Scripting for Distributed Web Applications: A Case Study in Bioinformatics (A2/I5)

4.1 Abstract

Prova is a language for rule-based Java scripting to support information integration and agent programming on the web. Prova integrates Java with derivation and reaction rules supporting message exchange with various comminication frameworks. Prova supports transparent access to databases, retrieval of URLs, access to web services, and querying of XML documents. We briefly illustrate Prova and show how to implement a distributed bioinformatics application, which includes access to an ontology stored in a database and to XML data for protein structures. Finally, we compare Prova to other event-condition-action rule systems.

4.2 Introduction

Prova is a language for rule based Java scripting for information integration, and agent programming [9, 5]. Prova is suitable for use as a rule-based backbone for distributed web applications in biomedical data integration. It has been designed to meet the following goals:

- Combine the benefits of declarative and object-oriented programming;
- Combine the syntaxes of Prolog and Java to appeal to programmers in both worlds;
- Expose logic and agent behaviour as rules;
- Access data sources via wrappers written in Java or command-line shells like Perl;
- Make all Java APIs from available packages directly accessible from rules;
- Run within the Java runtime environment;
- Enable rapid prototyping of applications;
- Offer a rule-based platform for distributed agent programming with common messaging protocols.

These design goals are especially important for integration tasks where location and format transparency are important. The latter means that the language should support the work with databases, RDF, HTML, XML, and flat file formats. Prova's rule-based approach is particularly important for two applications: derivation rules to reason over ontologies and reaction rules to specify reactive behaviour of possibly distributed agents.

Let us consider examples to illustrate these two types of rules. As a declarative language with derivation rules Prova follows a Prolog-style syntax as the next example shows:

Example 1 *(Declarative programming)*
Graph traversal is a typical example for declarative programming. A standard example is the same generation problem, in which all nodes in a tree are returned, which belong to the same generation. Two nodes are in the same generation if they are siblings or if their parents are in the same generation. The corresponding Prova programme is identical to standard Prolog, the execution semantics of Prova follow the usual top-down, left to right resolution.

Listing 16: Prova example

```
parent(anna, gerda).
parent(anna, fritz).
parent(asif, anna).
parent(asif, yanju).
parent(yanju, anja).

sg(X,Y) :- parent(Z,X), parent(Z,Y).
sg(X,Y) :- parent(Z1,X), parent(Z2,Y), sg(Z1,Z2).
```

The query `:- solve(sg(gerda,X)).` *will return* `X=gerda`, `X=fritz`, *and* `X=anja`.

Thus, Prova follows classical Prolog closely by declaratively specifying relationships with facts and rules. Now let us consider two examples, where access to Java methods is directly integrated into rules.

Example 2 *(Object-oriented programming)*
The code below represents a rule whose body consists of three Java method calls: the first to construct a String object, the second to append something to the string, and the third to print the string to the screen.

Listing 17: Prova example

```
hello(Name):-
  S = java.lang.String("Hello "),
  S.append(Name),
  java.lang.System.out.println(S).
```

4.3 Prova and Reactivity

Prova's reaction rules can comprise events, conditions, and actions in any order, as both events and actions are raised by built-in predicates for receiving and sending messages. Both allow for various communication frameworks such as the agent messaging language Jade, the Java messaging system JMS [27] or even Java events generated by Swing components. Due to the natural integration of Prova with Java, Prova's reaction rules offer a syntactically economic and compact way of specifying agents behaviour while allowing for efficient Java-based extensions to improve performance of critical operations. JMS in general has the advantage of being a guaranteed delivery messaging platform. Intuitively it means that when computer A sends a message to computer B the latter is not required to be operational. Once B goes online the messages will be delivered.

4.3.1 Main features of Prova's reaction rules

Prova provides three main constructs for enabling agent communication:

- sendMsg predicates, which can be used as actions anywhere in the body of a derivation or reaction rule,
- reaction rules, which have a blocking rcvMsg in the head and which fire upon receipt of a corresponding event, and
- inline reactions, which are encoded by blocking receipt of messages using rcvMsg or rcvMult anywhere in the body of derivation or reaction rules.

[27] `java.sun.com/products/jms/`

Communication actions with sendMsg. The sendMsg predicate can be embedded into the body of an arbitrary derivation or reaction rule. It can fail only if the parameters are incorrect and the message could not be sent due to various other reasons including the dropped connection (note that in the JMS case, the message may be sent anyway even if the network is down).

The format of the predicate is:

```
sendMsg(Protocol,Agent,Performative,[Predicate|Args]|Context)
```

or

```
sendMsg(Protocol,Agent,Performative,Predicate(Args)|Context)
```

where Protocol can currently be either jade, jms, self, or queue. Jade and JMS use the corresponding communication layers, while self and queue send the message to the agent itself or to another agent running locally in the same process but in another thread. Agent denotes the target of the message. For the self, jade, and jms methods, Agent is the name of the target agent. For the queue option, Agent is the object representing the message queue of the target agent. For Jade messages, the agent name takes the form agent@machine while for jms messages the agent locations are read from configuration files and are not specified in the Agent parameter. Performative corresponds to the semantic instruction the broad characterisation of the message. A standard nomenclature of performatives is FIPA Agents Communication Language ACL (www.fipa.org).

[Predicate|Args] corresponds to the bracketed form and Predicate(Args) corresponds to functional form of the message content sent in the message envelope. The first form can be useful to match any literal including arity-0 predicates (in which case, query() is the represented as [query]) or arity-1 predicates (in which case, query(arg1) is represented as [query,arg1]). The problem with the functional form is that it is impossible to specify a general pattern accommodating predicates of arbitrary arity while the bracketed version is compatible with any arity. Context includes an arbitrary length list of comma-separated parameters that can be used to identify the message or to distinguish the replies to this particular message from other messages. In particular, it can be useful to include the protocol as part of context for the recipient of the message to be able to reply by using the same protocol.

The following code shows a complete rule that sends a code base (a fragment of Prova code) from an external File to the agent Remote that will then assimilate the rules being sent. The rules are encapsulated in a serializable Java StringBuffer object and sent with the literal for the built-in predicate consult. The particular version of consult will then read on the Remote machine the Prova statements from a StringBuffer (in contrast to the standard version of consult that reads statements from the specified file provided as an input string).

Listing 18: sendMsg Example

```
1
2 % Upload a rule base read from File to the host at address Remote
3
4 upload_mobile_code(Remote,File) :-
5         % Opening a file returns an instance of java.io.BufferedReader in Reader
6         fopen(File,Reader),
7         Writer = java.io.StringWriter(),
8         copy(Reader,Writer),
9         Text = Writer.toString(),
10        % SB will encapsulate the whole content of File
11        SB = StringBuffer(Text),
12        sendMsg(jade,Remote,eval,consult(SB)).
```

Reaction rules with rcvMsg. In Prova, reaction rules are implemented as rules whose head consists of a rcvMsg predicate, which has the same syntax as the sendMsg predicate:

```
rcvMsg(Protocol,To,Performative,[Predicate|Args]|Context)
```

The agent reacts to the message based on its pattern including the protocol, sender, performative, message content, and context. The following code shows a general purpose but simplified reaction rule for the FIPA queryref performative. The first rule triggers a non-deterministic derivation of the literal [Pred|Args] sent as the message content. Based on the agent's knowledge-base derive will instantiate Pred|Args and send corresponding replies. The second rule sends a special end_of_transmission message to inform the querying agent of the completion of the query. The Protocol parameter available as the first parameter allows the recipient of queryref to know the protocol (jade, jms etc.) that should be used for replies.

Listing 19: rcvMsg Example

```
% Reaction rule to general queryref
rcvMsg(Protocol,From,queryref,[Pred|Args]|Context) :-
        derive([Pred|Args]),
        sendMsg(Protocol,From,reply,[Pred|Args]|Context).
rcvMsg(Protocol,From,queryref,[Pred|Args],Protocol) :-
        sendMsg(Protocol,From,end_of_transmission,[Pred|Args]|Context).
```

Now we will show how to deploy Prova's derivation and reaction rules to implement a distributed web-based bioinformatics application.

4.4 The GoProtein tool

Biological databases are growing rapidly. Currently there is much effort spent on annotating these databases with terms from controlled, hierarchical vocabularies such as the GeneOntology [8]. It is often useful to be able to retrieve all entries from a database, which are annotated with a given term from the ontology. We want to build such a query engine according to the scheme shown in Fig. 9. The application consists of four agents, whose interaction is driven by reaction rules. The agents are a thin client, which contains nothing but a GUI to interact with the user, a server, which handles queries of the client, a database server, which contains the ontology and the protein IDs annotated with the ontology terms, and a protein database which contains detailed information on the protein in XML format. The client's GUI displays the ontology. If the user selects a term from the ontology, an event is fired, which triggers a request being sent to the GoProtein server. The server in turn queries the GeneOntology database server for protein IDs, which have been annotated with the ontology term. If the user selects a specific protein on the GUI, a query is sent to the server, which reacts by retrieving an XML file from the remote protein database and by extracting relevant information from the file and returning it to the client.

For this specific implementation of the GoProtein workflow we want to use the GeneOntology [8] as annotation vocabulary and the Protein Databank PDB [3] as protein database. The Gene ontology (GO) contains over 19.000 terms organised in three sub-ontologies relating to biological processes, molecular functions, and cellular components. GeneOntology is available in XML, OWL, or database dump. Here we use the database dump of the GeneOntology. The protein databank PDB is a database with over 25.000 3D protein structures. Entries contain protein names, species, literature references, and most important the 3D coordinates of all the atoms of the protein. PDB is available as flat file format and XML.

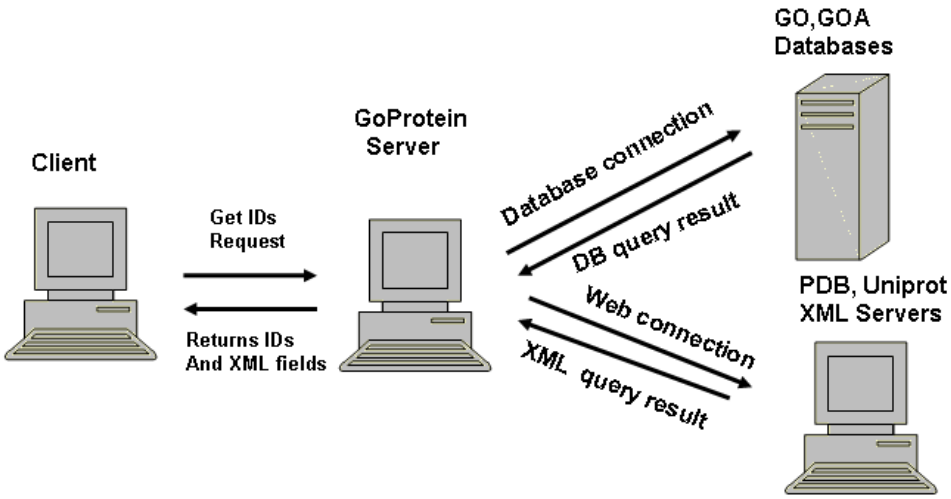


Figure 9: Sketch of the GoProtein tool workflow: The user interacts locally with a GUI on a client machine. Queries for all proteins annotated with a given term from the ontology are sent to the server. The server can access a database server to obtain protein IDs, which are annotated with the given term. The remote protein database returns an entry for protein as XML file given the protein ID. The protein database is used to display relevant information to the user.

4.5 Prova code for GoProtein

The client agent comprises the GUI and therefore makes heavy use of Java's Swing methods. For example, the MutableTreeNode class is used to display the GeneOntology tree as shown on Fig. 10.

Listing 20: Client agent

```
gui() :-
        println(["==========Window Loading=========="]),
        % create the tree and a placeholder for the IDs
        Node1 = javax.swing.tree.DefaultMutableTreeNode("all"),
        TreeModel = javax.swing.tree.DefaultTreeModel(Node1),
        Tree = javax.swing.JTree(Node1),
        Panel1 = javax.swing.JScrollPane(Tree),
        IdList = javax.swing.JList(),
        Panel2 = javax.swing.JScrollPane(IdList),
    ...
```

For large knowledge bases such as the 19.000 GeneOntology terms it is important to keep data on disc and load it into main memory only as needed. For this purpose, the code snippet below defines the location of the database and uses built-in predicates such as dbopen to open a database connection and sql_select to issue database queries. The concat statements are used to assemble the query string.

Listing 21: The Server Agent

```

:- eval(consult("utils.prova")).

```

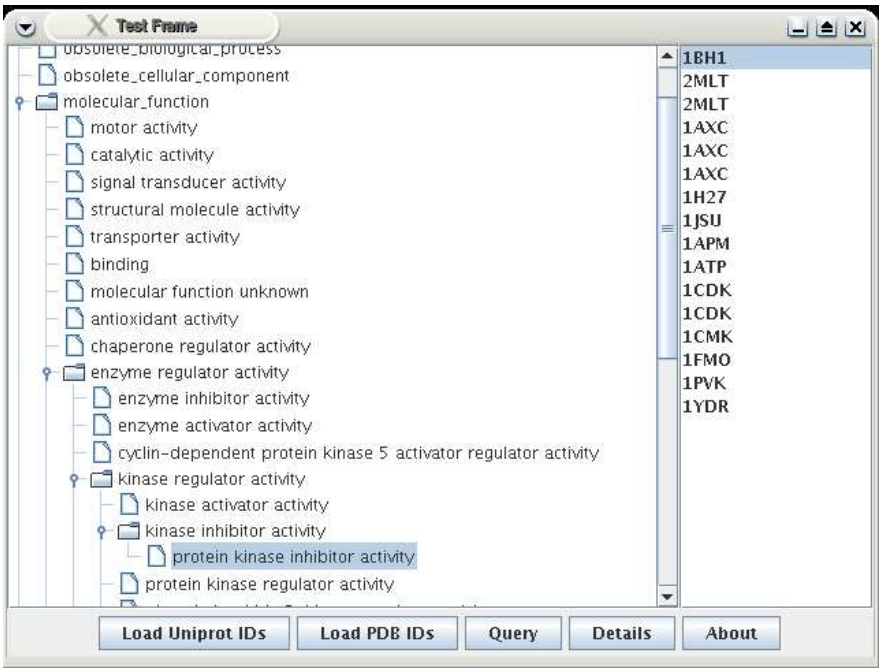


Figure 10: Screenshot of GoProtein: The left panel shows the ontology including the term ”protein kinase inhibitor” and on the right the PDB entries annotated with the term.

```
% Define database location
location(database,"GO","jdbc:mysql://myserver","guest","guest").

% get description of term
desc(Term, Desc) :-
        dbopen("GO",DB),
        unescape("\'", Quote),
        concat(["term_definition.term_id = term.id AND term.name =", Quote, Term, Quote],A),
        concat(["term, term_definition"],From),
        sql_select(DB,From,['term_definition.term_definition',Desc],[where,A]).
```

The user can also issue a request to extract specified fields from URLs of XML entries for a selected term. The code below shows the ability of Prova to connect to different URLs, process their XML contents and retrieve the requested fields using the built-in predicate descendantValue.

Listing 22: XML Handler

```

searchPDB("http://pdbbeta.rcsb.org/pdb/displayFile.do?structureId=").

% get the xml file
searchPDB(Query,XML):-
        print(["Query for ",Query," at PDB"]),
        searchPDB(BaseURL),
        concat([BaseURL,Query, "&fileFormat=xml"],URLString),
        retrieveXML(URLString,XML),
        println(["done"]).

% search for "sequence length" values in the xml file of a PDB ID
doSearchPDB(Term, Lst):-
```

```
14          searchPDB(Term,XML),
15          PDB = "PDBx:",
16          concat([PDB,"length_a"],La),
17          descendantValue(XML,La,A),!,
18          concat([PDB,"length_b"],Lb),
19          descendantValue(XML,Lb,B),!,
20          concat([PDB,"length_c"],Lc),
21          descendantValue(XML,Lc,C),!,
22          Lst = [A,B,C].
23
24  %%%%%%%%%%%%%%%% UTILITIY predicates %%%%%%%%%%%%%%%%%%%%%%
25
26  retrieveXML(URLString,Root):-
27          URL = java.net.URL(URLString),
28          print(["."]),
29          Stream = URL.openStream(),
30          print(["."]),
31          ISR = java.io.InputStreamReader(Stream),
32          XMLResult = XML(ISR),
33          Root = XMLResult.getDocumentElement(),
34          print(["."]).
```

The communication between the client and server agents is performed by using the Prova massaging and reaction rules to specify behaviour of the two agents. The predicate remote in line 1 takes as an argument the specification of the target machine we are communicating with. The reaction rule in line 5, 8, 10 are triggered by an event from the GUI's Swing component, while the one on line 15 is triggered by a message sent by the server. One of the actions triggered by the reaction rule in line 5 is a message sent to the server (last line).

Listing 23: The Client Agent

```
1  remote("ils_assign_server@servername").
2
3  % message transfer for the listeners:
4  % Reaction to button actions
5  rcvMsg(XID,Protocol,From,swing,[action,Cmd,Source|Extra]) :-
6          process_button(Source,Cmd).
7  % Reaction to incoming swing mouse clicked messages.
8  rcvMsg(XID,Protocol,From,swing,[mouse,clicked,Src|Extra]) :-
9          process_mouse(clicked,Src|Extra).
10 rcvMsg(XID,Protocol,From,swing,[mouse,entered,Src|Extra]) :-
11         process_mouse(entered,Src|Extra).
12
13 % message transfer with the server
14 % actions after receiving the results of a query
15 rcvMsg(XID,Protocol,From,reply_qry,[IDs]|Context) :-
16         mainlist(List),
17         buildList(List, IDs).
18
19 % process executed when the "Load Uniprot IDs" button is clicked
20 % it finds the selected node, finds all its associated Uniprot IDs, and loads them in the List
21 process_button(Button, "Load Uniprot IDs") :-
22         Tree = Button.getTree(),
23         Path = Tree.getSelectionPath(),
24         Node = Path.getLastPathComponent(),
25         Term = Node.toString(),
26         List = Button.getList(),
27         buildList(List, ["contacting server...","please wait"]),
28         % ask for the list of associated uniprod IDs
29         remote(Remote),
30         sendMsg(XID,jade,Remote,uniprot,[Term],"context").
```

4.6 Comparison and Conclusion

The World Wide Web is a rich heterogenous media following a pattern of growth that is uncentralized, directed by trends, and resistant to initiatives to enforce strong conformance to standards. A language for reactivity on the Web should be simple, offer 'out of the box' ability to handle most current de facto standards and offer specification robustness through clear declarative semantics. The many recent efforts that have been initiated to bring proper semantics to the Web - the Semantic Web - must also be kept in mind, as they delineate what the Web could eventually become. One can thus enumerate some 'must have' features for a Reactive Web Language:

- Ability to read and write XML, RDF, OWL, RSS and their variants;
- Possibility to interface to systems written in Java and/or embed java code;
- Connectivity to public/private databases, through different media (direct, web or web-services);
- Simple access to URL-based resources: Web page, XML file, RSS feed;
- Reactivity through the listening, processing and sending of events and actions;
- Declarative semantics and Event-Condition-Action paradigm.

In the following, we briefly compare Prova with other languages to address the problem of reactivity on the web: JESS, a java based rule engine, XChange based on the Xcerpt web query language, and ruleCore, an XML-based active rule engine.

4.6.1 JESS

Jess is a forward-chaining rule engine based on the Rete algorithm for the Java platform [7]. Jess supports the development of rule-based systems which can be tightly coupled to code written in the Java language. The syntax of the Jess language is Lisp-based. Java functions can be called from Jess, and Jess can be extended by writing Java code. Jess rules can be embedded in Java applications. Jess inherits from Java all the XML libraries to read, process and write XML data. However, it does not provide rule-based wrappers that provide these facilities in a transparent manner. The same holds for connectivity to databases: it is possible through Java libraries but not truly integrated in the system. This is one of the main differences between Prova and Jess, Prova has specialized predicates that allow easy and transparent access to databases, XML data, message exchange frameworks, and even events from Swing components. The fact that Jess is essentially a rule engine, provides a very natural setup to write Event-Condition-Action rules in the context of event propagation in the Web. Thus Business rules can be stated in a declarative and transparent manner.

4.6.2 XChange

XChange is a declarative language built upon the declarative web query language Xcerpt [4]. It provides Web-specific capabilities such as propagation of changes on the Web and event-based communications between Web sites. XChange is a research project and has a prototype implementation as proof-of-concept implementation. Among the interesting characteristics of XChange is its use of explicit temporal constructs to describe sequences of events, their overlapping and composition. Reactivity is achieved by having Event-Condition-Action rules at the core of the language. The main difference between Prova and XChange is that Prova is a full featured programming language built-upon the robustness and richness of Java, whereas XChange is geared towards XML and HTML contents.

4.6.3 ruleCore

Of proved industrial strength, the ruleCore Engine (`www.rulecore.com/`) is a robust implementation of an active rule engine server. The ruleCore Engine implements Event-Condition-Action rules that are organised in situation trees. The goal of ruleCore is to detect situations that arise as the temporal and logical composition of events. The rule engine itself does not rely on a generic programming language as in the case of Prova and Jess, but instead on the definition of situations as event detector trees. Connectivity to other media and systems is achieved through the use of event and action wrappers, most of which are provided 'out of the box' for databases and standard industrial messaging frameworks like XML-RPC, Web Services, TIBCO Rendezvous, plain sockets or IBM WebSphere MQ. The main difference between Prova and the ruleCore engine is, as in the case of XChange, that Prova is a generic rule language extending Java, whereas ruleCore is a language-independent rule engine.

Prova is the choice of a Java programmer with Prolog experience who aims to develop a system which needs a possibly thin layer of rules for reasoning with backward chaining and for defining business rules and workflows with agent communication. Prova is available at www.prova.ws.

References

[1] The Gene Ontology (GO) project in 2006. *Nucleic Acids Research*, 34(Database issue):D322–6, 12 2005.

[2] Bairoch A, Apweiler R, Wu CH, Barker WC, Boeckmann B, Ferro S, Gasteiger E, Huang H, Lopez R, Magrane M, Martin MJ, Natale DA, O'Donovan C, Redaschi N, and Yeh LS. The Universal Protein Resource(UniProt). *Nucleic Acids Res.*, 33:D154–159, 2005.

[3] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Res*, 28(1):235–242, 2000.

[4] F. Bry and P. Patranjan. Reactivity on the web: Paradigms and applications of the language xchange. In *Proceedings of 20th Annual ACM Symposium on Applied Computing (SAC'2005)*. ACM, March 2005.

[5] J. Dietrich, A. Kozlenkov, M. Schroeder, and G. Wagner. Rule-based agents for the semantic web. *Journal on Electronic Commerce Research Applications*, 2(4):323–38, 2003.

[6] Wheeler DL, Chappey C, Lash AE, Leipe DD, Madden TL, Schuler GD, Tatusova TA, and Rapp BA. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.*, 28:10–4, 2000.

[7] Ernest Friedman-Hill. *Jess in Action Java Rule-based Systems*. Manning, 2003.

[8] GeneOntologyConsortium. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Res.*, 1(32):D258–61, 2004.

[9] A. Kozlenkov and M. Schroeder. PROVA: Rule-based Java-scripting for a bioinformatics semantic web. In E. Rahm, editor, *International Workshop on Data Integration in the Life Sciences DILS*, Leipzig, Germany, 2004. Springer.

[10] Alexander Kozlenkov and Michael Schroeder. PROVA: Rule-based Java-Scripting for a Bioinformatics Semantic Web. In E. Rahm, editor, *International Workshop on Data Integration in the Life Sciences DILS*, Leipzig, Germany, 2004. Springer.

[11] Donna Maglott, Jim Ostell, Kim D Pruitt, and Tatiana Tatusova. Entrez Gene: gene-centered information at NCBI. *Nucleic Acids Res*, 33(Database issue):D54–D58, 2005.

[12] Sebastian Schaffert and François Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proceedings of Extreme Markup Languages 2004, Montreal, Quebec, Canada (2nd–6th August 2004)*, 2004.

5 Conclusion

The previous chapters document some results of I-working groups that are applicable to the A2 bioinformatics application. These are examples, where particulary deep integration has been achieved, but they are not exclusive examples of A2-relevant I-research. However, a fundamental difficulty in applying I results in A2 (in contrast to other applications) is that A2 requires deep understanding of the application domain before any collaboration can start. The next deliverable will document the state of the bioinformatics demonstrators and will have a specific focus on protein interaction networks. On the one hand, it will show case how typing as investigated by I3 is used in the Biocham System and how consistency checking for logic programmes as developed originally by the Lisbon group (I5) can be used to reason over metabolic networks.