



UvA-DARE (Digital Academic Repository)

Logic programming for knowledge-intensive interactive applications

Wielemaker, J.

Publication date
2009

[Link to publication](#)

Citation for published version (APA):

Wielemaker, J. (2009). *Logic programming for knowledge-intensive interactive applications*. [Thesis, fully internal, Universiteit van Amsterdam].

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

Summary

Logic programming is the use of mathematical logic for computer programming. In its pure form, a logic program is a declarative description of a solution that is completely separated from the *task* of solving the problem, the procedural aspect. Lacking an efficient resolution strategy for expressive logics and the difficulty of expressing problems that are by nature procedural limit the usability of pure logic programs for building real-world applications.

An important step towards a usable language for logic programming was the invention of the programming language 'Prolog'. This language is based on a simplified form of first order logic (Horn Clauses) and a simple resolution strategy (SLD resolution). Prolog programs can, at the same time, be read as a declarative description of a solution and as a procedural strategy to find this solution. In other words, Prolog can describe both knowledge and procedures. Interaction is a typical aspect of applications that have procedural aspects. The combination of declarative and procedural aspects in one language (Prolog) is what makes this language promising for the applications that are considered in this thesis: knowledge-intensive interactive applications.

There are dozens of implementations of the Prolog language. Most concentrate on Prolog as a rule-based system and leave the interaction task to other programming environments. This implies that an interactive application can only be built using a *hybrid* environment. The required bridge between Prolog and the language used for the interactive aspects of the application makes programming, and in particular prototyping of interactive applications, complicated and can cause a significant performance degradation. In this thesis, we study the requirements, design and implementation of an integrated Prolog-based environment that can be used for developing knowledge-intensive interactive applications.

Building applications is important in this study. The primary goal of these applications was *not* their architecture but, for example, tools to support knowledge-engineers with modelling knowledge or explore how differently structured databases that contain collections from different museums can be joined and searched together. The architecture of these applications was irrelevant to projects in which they were developed, but the search for adequate building blocks for these applications forms the basis of this thesis. While developing these applications we identified reusable components and added these to Prolog and its libraries. In

subsequent implementations we tested these extensions and refined or, if necessary, replaced them. Stable extensions have been described in scientific publications and are distributed as Open Source. Feedback from the Open Source community provided suggestions for alternatives or further refinement. In this study we particularly pay attention to the following three problems:

- *Knowledge representation*

Although Prolog programs can be read as a declarative specification, the language is limited. Many Prolog programs that are a correct declarative description of a problem do not terminate or do not terminate within a reasonable time. In addition to the termination and performance problem, logic-based languages are considered inferior to frame-based languages (e.g., KL-ONE) for knowledge-representation that supports interactive inspection and manipulation.

At the start of this century, the Semantic Web community proposed the RDF language for knowledge-representation. This language provides a very simple data-model: triples of the format {subject, predicate, value}. More expressive languages such as RDFS and OWL are layered on top of this datamodel. The simple relational model of RDF fits perfectly with Prolog and now forms the basis of our knowledge-representation.

- *Graphical applications*

Most Prolog implementations do not provide graphics and use a bridge to a conventional graphical environment. In our opinion this harms productivity too much, while the declarative aspects of Prolog are useful for the description of interfaces and user-interaction. Most graphical environments are structured as an object-oriented system. We realised a generic bridge between Prolog and an external object-oriented system and described the conditions to use this architecture with other object-oriented systems.

- *Web applications*

In the recent past the web developed from a web of documents to a web of interactive applications. The use of knowledge in such applications is an active research topic. The integration of museum data mentioned earlier is an example. Because RDF is particularly suitable for integrating diverse knowledge-bases and RDF fits well with Prolog, the use of Prolog for the knowledge-aspects of this research is promising. If we embed Prolog in a traditional web-server and page-generation language (e.g., Apache with PHP), we lose many of Prolog's advantages for fast interactive development, i.e., we need a more 'Prolog-friendly' way to generate web-applications.

Our challenge is to extend Prolog and add libraries that make the language suitable for building large and innovative applications entirely in Prolog. In this architecture, only well understood primitives such as interfaces to the operating system, network and graphics are written in other languages and made available as a Prolog library.

Overview of this thesis Part I of this thesis describes extensions to Prolog and Prolog libraries that allow for writing knowledge-intensive interactive applications entirely in Prolog. Were possible, we compare our components with approaches used for similar problems in other (often Prolog) environments. Part II is a formative evaluation of these extensions: the described case-studies show that the programming environment can support the application, and at the same time both the development and evaluation of these applications have contributed to the development or refinement of the extensions described in part I.

Part I Chapter 2 describes Triple20, a tool for browsing and limited editing of ontologies. Triple20 is not part of Prolog and its libraries, which suggests it should not be in part I of this thesis. Although Triple20 can be used as a Prolog library for RDF-based graphical applications, the main reason to start this thesis with Triple20 is that this chapter describes how knowledge is represented in the RDF model and how we can use the RDF triple model as the foundation for an architecture to build interactive applications. This chapter describes an extension of the *model-view-controller* reference model for interactive applications which is necessary to bridge the large gap between the low-level knowledge-representation and the interface efficiently. Concurrency (threads) is one of the necessary building blocks. The design and implementation of Triple20 is based on requirements formulated by evaluating the ‘MIA’ tools, described in chapter 9.

The remainder of part I describes extensions to Prolog and Prolog libraries that made, for example, Triple20 possible: storage and query of RDF triples, creating execution plans for complex queries on this data, connection to graphical libraries, concurrency and an infrastructure for building web-applications.

The RDF triple model fits perfectly on the relational model of Prolog. A triple-database that is suitable for researching the aforementioned integration of museum collections contains at least 10 million triples, must be accessible from multiple threads and allow for efficient full-text search. RDF predicates are organised in one or more hierarchies, which must be used for reasoning over integrated knowledge-bases. This type of *entailment reasoning* must be supported with minimal overhead. The order in which individual RDF patterns are matched to the knowledge-base is of utmost importance for executing compound search requests (conjunctions). This optimisation problem, which can be compared to database ‘join’ optimisation, is the subject of chapter 4. It is based on metrics from the RDF database. Part of the requirements on the RDF store are too specific to base the implementation on an optimised version of the Prolog dynamic database. The implementation of the RDF store is the subject of chapter 3.

Chapter 5 discusses the bridge between Prolog and graphical libraries. Almost all such libraries are based on the object-oriented paradigm. First, we describe a compact interface to manipulate objects from Prolog. New functionality in object-oriented systems is typically realised by deriving new classes from the base-classes (sub-classing). This cannot be achieved using the above mentioned simple interface, which would imply we still need to program in both Prolog and the object-oriented language to realise an application. We can solve this for most object-oriented systems by describing the class in Prolog and use this description

to generate classes and *wrappers* that call the implementation in Prolog. The chapter continues with describing further integration of Prolog datatypes and non-determinism in the object-oriented environment and concludes with a critical evaluation of this object system for Prolog.

Chapter 6 discusses concurrency in Prolog. Traditionally, most research on concurrency in Prolog aims at executing a single program using multiple CPUs. The declarative nature of the language make people believe that Prolog is more suitable to automatic introduction of concurrency than procedural languages. Still, this research did not convince many software developers. We had more simple demands: background computations in graphical environments, scalability of web-servers and exploiting recent *multi-core* hardware for these tasks. On this basis we created a simple model of cooperating Prolog engines, which has been adopted by two other Prolog implementations and is the basis for standardisation within ISO WG17. Chapter 6 describes the model, consequences for the Prolog implementation and two performance evaluations.

Chapter 7 provides an overview of the developments that support web-applications. This chapter gives a summary of the previous chapters about RDF and positions this material in the wider context of web-applications. Parsing, representing and generating web-documents is an important topic in this chapter. Where possible, we compare our approach to PiLLOW, another infrastructure for web-programming in the Prolog community. We discuss a Prolog-based web-server (HTTP), stressing code organisation and scalability aspects. Chapter 7 enumerates necessary features of a Prolog implementation for this type of applications: concurrency, unlimited atoms, atom garbage collection and support for the international UNICODE character set.

Part II The second part discusses three applications that contributed to the development and evaluation of the infrastructure described in the first part.

The first application (PIDoc, chapter 8) is also part of the SWI-Prolog infrastructure. PIDoc is an environment for *literate programming* in Prolog. Its embedded web-server provides an interactive environment to consult and improve the documentation. PIDoc is part of the development environment that is necessary to create a productive programming environment. Building the development tools has contributed significantly to testing and refinement of the infrastructure, notably for graphical applications.

Chapter 9 describes two experiments using a prototype tool for annotation and search of multimedia collections supported by background knowledge. This chapter also describes the software architecture, which is based on XPCE (chapter 5). Apart from XPCE, the infrastructure described in part I did not yet exist. This was our first tool which used RDF for exchanging knowledge and where the RDF model was central to the design of the tool. This chapter was included for three reasons: description of the application context, stress that tool-development and—in this case—research to the role of knowledge in annotations are closely integrated and formulate requirements for the next generation of infrastructure and tools. This work led to the development of Triple20 and a scalable RDF store. It also provided an additional motivation to introduce concurrency in Prolog.

Part II of this thesis concludes with chapter 10 which describes the semantic search and annotation toolkit called ClioPatria. This chapter motivates the architecture of ClioPatria, which uses all technology described in part I.⁶ The *use-case* for ClioPatria is the development of a prototype application that makes descriptions of works of art from museums, together with available background information such as databases on artists, art vocabulary and geographical information, searchable. One of the challenges is the diversity in data formats, schemas and terminology used by the museums. The structure of the integrated knowledge is so complex that structured queries (e.g., SQL queries) cannot be realistically formulated. We made this data searchable by exploring the RDF graph based on *semantic distance*, after which we cluster the results based on the semantic relation between search-term and the works of art found. ClioPatria provides a modern web-based interactive interface based on AJAX technology and Yahoo! (YUI). The server is a Prolog web-server that provides all services.

Discussion The assumption in this thesis is that logic programming, and particularly Prolog, is suitable for the development of knowledge-intensive interactive software. The introduction of RDF and RDF-based knowledge-representation languages has contributed to the credibility of Prolog in this domain. Some properties of RDF models, such as their scale and required inferences, make a pure Prolog implementation less suitable. However, RDF models can be implemented as a library in another language (e.g., C), which acts as a natural extension to Prolog. The web has a large influence on the architecture of applications. Prolog is, extended with suitable libraries, an adequate language for web-applications.

Next to further standardisation of the Prolog language, it is of vital importance that the Prolog community establishes standards for representing and generation of (web-)documents in XML and HTML. Such standards are needed to share resources for web-applications within the Prolog community. The RDF model can play this role for knowledge-representation because accessing the RDF model from Prolog leaves few options.

This thesis is a direct consequence of how the SWI department (now called HCS) has integrated software development with research, providing room for the software developers to establish their own research agenda: software architecture. This approach has clear advantages: (1) it provides an opportunity for scientifically motivated software developers to carry out research into software architecture in the context of a concrete requirement for an application and (2) development of the target application is based on direct involvement. Direct involvement supports a short development and evaluation cycle, which is a fruitful process model for a large class of applications. This certainly applies to academic demonstrators.

With SWI-Prolog we realised a productive environment for our own research. Wide acceptance of the Open Source philosophy has contributed to the popularity of SWI-Prolog, through which we helped spreading the logic programming paradigm in academic and commercial environments. At the same time, this user community motivates us and feeds us with problems and solutions.

⁶The graphical interface is only used as part of the Prolog development environment.

