



UNIVERSITY OF AMSTERDAM

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.

Contents

Preface	vii
1 Introduction	1
1.1 Application context	1
1.2 Logic Programming	3
1.3 Project context	6
1.4 Research questions	7
1.5 Approach	9
1.6 Outline	9
I Infrastructure for knowledge-intensive applications	13
2 Using triples for implementation: the Triple20 ontology-manipulation tool	15
2.1 Introduction	15
2.2 Core technology: Triples in Prolog	16
2.3 Design Principles	17
2.4 Architecture	18
2.4.1 Rules to define the GUI	20
2.5 An overview of the Triple20 user interface	21
2.6 Implementation	23
2.6.1 The source of triples	23
2.7 Scalability	23
2.8 Related work	25
2.9 Discussion	26

3	Prolog-based Infrastructure for RDF: Scalability and Performance	27
3.1	Introduction	27
3.2	Parsing RDF/XML	28
3.3	Storing RDF triples: requirements and alternatives	30
3.3.1	Requirement from integrating different ontology representations	30
3.3.2	Requirements	31
3.3.3	Storage options	32
3.4	Realising an RDF store as C-extension to Prolog	33
3.4.1	Storage format	33
3.4.2	Concurrent access	35
3.4.3	Persistency and caching	35
3.4.4	API rationale	36
3.5	Querying the RDF store	38
3.5.1	RDFS queries	38
3.5.2	Application queries	40
3.5.3	Performance of rdf/3 and rdf_has/4	40
3.6	Performance and scalability comparison	41
3.6.1	Load time and memory usage	43
3.6.2	Query performance on association search	45
3.7	Conclusions	47
4	An optimised Semantic Web query language implementation in Prolog	49
4.1	Introduction	49
4.2	Available components and targets	51
4.3	RDF graphs and SerQL queries graphs	51
4.4	Compiling SerQL queries	52
4.5	The ordering problem	54
4.6	Estimating the complexity	57
4.7	Optimising the conjunction	58
4.8	Optional path expressions and control structures	59
4.9	Solving independent path expressions	60
4.10	Evaluation	61
4.11	Related Work	63
4.12	Discussion	64
4.13	Conclusions	64
5	An architecture for making object-oriented systems available from Prolog	67
5.1	Introduction	67
5.2	Approaches	68
5.3	Basic Prolog to OO System Interface	69
5.4	Extending Object-Oriented Systems from Prolog	72
5.5	Transparent exchange of Prolog data	75

5.5.1	Passing Prolog data to a method	76
5.5.2	Storing Prolog data in an XPCE instance variable	77
5.5.3	An example: create a graphical from a Prolog tree	78
5.5.4	Non-deterministic methods	78
5.6	Performance evaluation	80
5.7	Events and Debugging	80
5.8	Related Work	81
5.9	Conclusions and discussion	82
6	Native Preemptive Threads in SWI-Prolog	85
6.1	Introduction	85
6.2	Requirements	86
6.3	What is a Prolog thread?	87
6.3.1	Predicates	87
6.3.2	Synchronisation	88
6.3.3	I/O and debugging	89
6.4	Managing threads from Prolog	89
6.4.1	A short example	89
6.4.2	Prolog primitives	90
6.4.3	Accessing Prolog threads from C	93
6.5	Implementation issues	93
6.5.1	Garbage collection	94
6.5.2	Message queues	95
6.6	Performance evaluation	95
6.6.1	Comparing multi-threaded to single threaded version	96
6.6.2	A case study: Speedup on SMP systems	97
6.7	Related Work	100
6.8	Discussion and conclusions	100
7	SWI-Prolog and the Web	103
7.1	Introduction	104
7.2	XML and HTML documents	105
7.2.1	Parsing and representing XML and HTML documents	105
7.2.2	Generating Web documents	107
7.2.3	Comparison with PiLLow	110
7.3	RDF documents	110
7.3.1	Input and output of RDF documents	112
7.3.2	Storing and indexing RDF triples	114
7.3.3	Reasoning with RDF documents	116
7.4	Supporting HTTP	117
7.4.1	HTTP client libraries	118
7.4.2	The HTTP server library	120

7.5	Supporting AJAX: JSON and CSS	122
7.5.1	Producing HTML head material	124
7.5.2	Representing and converting between JSON and Prolog	126
7.6	Enabling extensions to the Prolog language	127
7.6.1	Multi-threading	128
7.6.2	Atoms and UNICODE support	128
7.7	Case study — A Semantic Web Query Language	129
7.8	Case study — XDIG	132
7.8.1	Architecture of XDIG	133
7.8.2	Application	134
7.9	Case study — Faceted browser on Semantic Web database integrating multiple collections	135
7.9.1	Multi-Faceted Browser	135
7.9.2	Evaluation	137
7.10	Conclusion	138
II Knowledge-intensive applications		141
8	PIDoc: Wiki style literate Programming for Prolog	143
8.1	Introduction	143
8.2	An attractive literate programming environment	145
8.3	An example	147
8.4	Description of PIDoc	149
8.4.1	The PIDoc syntax	149
8.4.2	Publishing the documentation	151
8.4.3	IDE integration and documentation maintenance cycle	151
8.4.4	Presentation options	153
8.5	User experiences	153
8.6	Related work	155
8.7	The PIDoc L ^A T _E X backend	156
8.8	Implementation issues	157
8.8.1	Collecting documentation	158
8.8.2	Parsing and rendering comments	158
8.8.3	Porting PIDoc	158
8.9	Conclusions	159
9	Semantic annotation and search	161
9.1	Ontology-Based Photo Annotation	161
9.1.1	Introduction	161
9.1.2	Our approach	162
9.1.3	Developing ontologies	162

9.1.4	Annotating photographs using our multimedia information analysis tool	168
9.1.5	Discussion	170
9.2	Supporting Semantic Image Annotation and Search	173
9.2.1	Introduction	173
9.2.2	Approach	173
9.2.3	Background Knowledge	174
9.2.4	An Application Scenario	177
9.2.5	Discussion	180
9.3	Lessons learned	182
10	Thesaurus-based search in large heterogeneous collections	185
10.1	Introduction	186
10.2	Materials and use cases	186
10.2.1	Metadata and vocabularies	186
10.2.2	Use cases	187
10.3	Required methods and components	190
10.3.1	Using a set of fixed queries	190
10.3.2	Using graph exploration	190
10.3.3	Term search	191
10.3.4	Literal matching	192
10.3.5	Using SPARQL	192
10.3.6	Summary of requirements for search	193
10.4	The ClioPatria search and annotation toolkit	194
10.4.1	Client-server architecture	195
10.4.2	Output formats	195
10.4.3	Web services provided by ClioPatria (API)	196
10.5	Discussion and conclusion	198
11	Conclusions	203
11.1	The research questions revisited	204
11.2	Architectures	206
11.3	Discussion: evaluation of our infrastructure	206
11.3.1	Key decisions about the infrastructure	208
11.4	Challenges	212
	Bibliography	215
	Summary	227
	Samenvatting	233
	SIKS Dissertatiereeks	239

