Logic Programming Systems
The SWI-Prolog environment

Jan Wielemaker

Human-Computer Studies Laboratory, University of Amsterdam,
Kruislaan 419, 1098 VA Amsterdam, The Netherlands,
wielemak@science.uva.nl

Abstract. Development of the SWI-Prolog environment has started in 1985. Its developments was started in the context of interactive application development. More recently the development is guided by Semantic Web application development and contributions from the world wide community. In this article we will briefly introduce the SWI-Prolog community, touch the many features and outline our future plans.
The primary aim of the *Logic Programming Systems* series is to provide an overview of system features. We deliberately concentrate more on the context in which the development of SWI-Prolog started, how this is currently shaped and what our plans are. A big table is more suitable for a product comparison. An outline of the community is hopefully more pleasant to read and provides more stable information about the system.

1 The SWI-Prolog history and community

We started SWI-Prolog in a EU project (KADS) where we built a workbench for Knowledge Engineering. Several partners had a background in AI and where used to Prolog. These were the days where graphical user interfaces just started to emerge, in our case SunView on SUN workstations. Anjo Anjewierden created an object-oriented connection between Prolog and SunView, called PCE [7]. The initial system used Quintus Prolog which, in those days,[1] could call C, but could not be called from C. This limitation severely handicapped PCE.

In the meanwhile SWI-Prolog was created as a toy Prolog system, but designed with a bi-directional C interface from the start. Just for play, we connected it to PCE and added enough built-ins to run our workbench-under-development. Enhanced functionality of PCE resulting from the bi-directional interface, a very fast compiler and the **make/0** feature to recompile modified source files quickly moved the whole consortium to use this new Prolog. As we had no concrete plans with it, we put it the source on the anonymous ftp server under a non-commercial-use license.

Somehow the sources were picked up, as it turned out mostly by Universities as the default system for teaching Prolog classes. Most likely because the system was simple, small, installed easily on most Unix systems in common use and it could be used

[1] Quintus, as most other todays Prolog systems offer a good bidirectional C interface.

without going through the legal and financial departments. We established a mailinglist, continued development in various projects and frequently released new versions.

PCE turned into XPCE when X11 became fashionable and the whole system was ported to Windows using Windows-NT 3.51. (X)PCE was distributed separately as a licensed system with a limited free demo for Windows. With help from Richard Stallman from the Free Software Foundation we established a transparent license schema that allowed for using the GNU readline library for editing input lines. After convincing the University we re-released the combined XPCE/SWI-Prolog system, using the LGPL for all C-code and the GPL with a statement originating from the GCC runtime libraries for the Prolog code. This statement allows using the Prolog code from the kernel and libraries alongside proprietary code. SWI-Prolog 5.0/XPCE 6.0 was released under these conditions early 2002. This proved a big step forwards[2], inviting both commercial usage and a community that contributed code in addition to bug reports and feature requests.

Commercial users sponsored the development of the SSL (Secure Socket Layer) interface, big integer and rational number support, literate programming (PlDoc) and the testing environment (PlUnit).

In 2003 Bart Demoen convinced us to add attributed variables based on the hProlog approach for dynamic attributes, after which the Leuven team would port their constraint programming tools to SWI-Prolog. Tom Schrijvers has created CHR for SWI-Prolog, Leslie de Koning ported clp(q) and clp(r). Markus Triska added several specialised constraint solvers and is currently developing a full fledged clp(fd) system. Paul Singleton developed and maintains JPL, the Java interface.

In the meanwhile internal projects started concentrating on the Semantic Web and Web services, resulting in libraries for RDF storage [11, 9] and a comprehensive multi-threaded HTTP server infrastructure [10].

The central focus of the community is the mailinglist with approximately 600 members exchanging about 800 messages annually (2007). Second is a Bugzilla server for bug reports which functions well. We have been running a Wiki for several years to invite exchange of ideas and code. It turned out the contribution by spammers was far larger than the community contribution, forcing us to disable write access. Possibly we were too early. We are still looking for a platform that promotes exchange and improvement of reusable Prolog code in the community like Perl's CPAN[3] network.

With SWI-Prolog, we aim at providing a scalable robust and portable Prolog system to the educational, research and commercial community. We release about every two weeks or immediately after a bug has been fixed that is likely to affect many users or is a blocker for just a few users. This guarantees everyone has access to the latest bug fixes and features. We adopt the Open Source policies as outlined by Linus Torvalds, where we release new features in an early and often incomplete state, waiting for early adopters in the community to help settling the final specifications and make the library stable. This approach has worked notably well for the ODBC interface, multi-threading, unbound integer and rational number support, RDF and the HTTP services. This approach appears to work because

[2] http://www.swi-prolog.org/statistics.html
[3] http://www.cpan.org/

1. Initial development of a prototype generally takes only 10%-20% of total development time in traditional development. After this initial development the community starts to contribute with feature requests (saving specification time) and bug reports (filling the test suite).
2. Early adopters can help shaping the final system. If the developers make sure issues are fixed quickly (hours, at most a day), the users stay satisfied. Perceived missing functionality is discussed between developer and user and efficiently resolved either by extending the library or discovering how the problem can be solved appropriately using the available functionality.
3. Early adopters generally install the required software to build SWI-Prolog from source and follow the version from the SCM system (currently GIT, see below) to bypass the bi-weekly release cycle.

Since October 2007 we moved revision control from CVS to GIT.[4] GIT provides a much better web interface, is much faster and allows for distributed and offline version management. These features make it much easier for developers to cooperate or fork their own version without the need for elaborate right management.

2 SWI-Prolog by feature

2.1 Core features

SWI-Prolog's kernel is loosely based on the ZIP [1, 3] virtual machine. The Prolog engine is not designed for speed, but has all commonly seen optimisations to avoid memory exhaustion in 24*7 services: last call optimisation, garbage collection and atom garbage collection. The kernel is designed to impose few limits. Atoms length, integer size and term arity is bound by memory only and atoms cover the full Unicode character set, including 0-characters.

Emerging from the need in interactive applications and network services we support for multi-threading. Multi-threading [5] is currently under discussion in the ISO WG17 group where the initial specification is based on the SWI-Prolog manual. The current draft is implemented in SWI-Prolog, XSB and YAP.

The primitives offered by SWI-Prolog aim at satisfying a wide user community. Built-ins and libraries inherits from different Prolog traditions: Edinburgh, ISO and Quintus/SICStus. Language constructs include mutable terms, global variables, attributed variables and coroutining based on attributed variables. The system supports both backtrackable and non-backtrackable mutable terms and global variables. Non-backtrackable mutable data on the stacks is often used as a thread and exception-safe building block for primitives whose implementation requires data to survive backtracking, such as variations on the all-solution predicates (e.g. resource bounded **findall/3**).

Portability has always been an important asset. The current core system depends on the Ansi C99 standard for fixed-width integers and Unicode handling, POSIX threads (using pthreads-win32[5] mixed with Windows primitives for performance on Windows),

[4] http://git.or.cz/

[5] http://sourceware.org/pthreads-win32/

the GNU GMP library for unbounded integers and rational numbers and optionally the GNU readline library for input editing. XPCE graphics depends on either X11 or the Windows GDI. The system compiles on both 32-bit and 64-bit hardware, including the 64-bit editions of Windows XP and Vista. Database connectivity depends on ODBC, using UnixODBC[6] on Unix systems. Secure Socket connections depend on OpenSSL[7]. Official releases include binaries for Linux, 32-bit and 64-bit Windows and MacOS X.

2.2 Development environments

The standard development environment is merely a set of components that provide hooks to replace parts of it by alternatives. The downside is that there is no integrated central graphics window that exposes all features. There are some alternatives that provide for a common front-end. Notably SWI-Prolog Editor[8] by Gerhard Röhner (Windows only) and an extensive GNU Emacs mode[9] by Markus Triska. These tools provide access to the native IDE components which we will briefly summarise here.

itemlabel *PceEmacs*
PceEmacs is an Emacs clone written in XPCE/SWI-Prolog. It allows for opening the edit buffer as a Prolog stream. This feature is used to realise syntax colouring based on cross-referencing and actual parsing. The clause under the cursor is re-parsed on every keystroke. It is not coloured if it cannot be parsed, providing adequate and immediate feedback on syntax errors. Colouring of goals is based on cross-referencing using the same library as **gxref/0** discussed below and distinguishes between undefined, local, imported, and dynamic predicates while providing a context sensitive menu. Colouring of variables signals singletons and highlights variables sharing with the one under the cursor.

itemlabel *Graphical tracer [6]*
The graphical tracer is a source-level tracer that exploits PceEmacs to show the source and current location. Additional windows provide variable bindings and a stack-view that includes choice points. Notably the latter proves efficient for fixing determinism problems.

itemlabel *Graphical cross referencer*
The **gxref/0** predicate analyses the currently loaded program, providing a graphical overview of file dependencies and a per-file dependency view including not-called predicates. The underlying cross-referencer can be called as a Prolog library. It provides hooks that allow for resolving complicated meta-calling.

itemlabel *Profiler [6]*
The execution profiler samples the execution, creating the actual call-tree and how time and calls propagate along the edges. A graphical front-end allows navigating this tree.

[6] http://www.unixodbc.org/
[7] http://www.openssl.org/
[8] http://lernen.bildung.hessen.de/informatik/swiprolog/indexe.htm
[9] http://stud4.tuwien.ac.at/~e0225855/ediprolog/ediprolog.html

itemlabel *PlDoc [8]*
PlDoc provides for literate programming support using Wiki-style comments. Feedback is immediate through the built-in web-server. PlDoc integrates the user documentation with the system libraries by processing the HTML system documentation. It also provides a LaTeX backend.

itemlabel *PlUnit*
PlUnit is a test environment. It provides management when tests are loaded and executed and various levels of feedback on the progress of the tests. Tests themselves look like an ordinary clause, where the head defines properties of the test such as whether it succeeds of fails, which bindings it must produce, which errors it raises, whether or not it is deterministic as well as optional setup and cleanup handlers. The clause body contains the code that is tested. PlUnit is now used for a large part of the system tests. PlUnit runs on SWI-Prolog and SICStus Prolog.

2.3 Libraries

SWI-Prolog aims at programming the the large. Its libraries include many of the de-facto libraries such as lists, ordered sets, association lists, graphs, etc. Above all though, it contains many libraries to deal with external data formats and protocols. Many of these are wrappers around C libraries, some are native Prolog. We will just reference supported protocols and formats using their acronyms. Full information is easily found on the web and SWI-Prolog manuals. SWI-Prolog provides bundled support for C, C++. CGI, Deflate (zlib), Graphics (XPCE connects to X11 or Windows), HTTP (client and server), Java, JSON, MIME, ODBC, RDF (XML and Turtle) [11], SGML/HTML/XML (ported to XSB), SSL, base64, sockets (TCP and UDP) and www-form-encoded.

3 SWI-Prolog plans

Project driven development currently concentrates on enhancing the Semantic Web and general web programming support. We think this is a fruitful domain for Prolog because of its natural handling of the RDF triple model. As web services get more rule based components, the use of Prolog for web programming becomes more important. Using AJAX[10] technology and a Prolog embedded HTTP server is becoming an interesting alternative for interactive Prolog applications.

A pilot project proved the need for and feasibility of restructuring the virtual machine for better modularity and faster execution. This project should also add tabling [4] to SWI-Prolog.

At the ICPL-07 (Porto), we discussed the options to merge with YAP[11] Prolog. The current plan is to make it step by step easier to write code that is portable between the two dialects. The first step has been implemented and established standards for conditional compilation and detecting of dialect and system features. The next step is largely agreed and extends the module system with re-export and import by another

[10] http://en.wikipedia.org/wiki/AJAX
[11] http://www.ncc.up.pt/ vsc/Yap/

name to make it easier to define new modules in terms of old modules, which we will use easily emulate our libraries and establish a common library structure. We also plan to arrive at a common foreign language interface. We discuss additions we think are of mutual interest so they are shared upfront and we tackle incompatibility issues on incident basis.

4 Conclusions

We described the history, community, feature and plans around SWI-Prolog. SWI-Prolog was developed for in-house prototype development on behalf of our research. It grew big in education. Especially after a re-launch as integrated system under a clear Open Source license we have seen increasing contributions from the community and interest from research and commercial users. It aims at supporting a large part of the logic programming community with a portable, robust and feature rich implementation of the Prolog language and achieves these goals by maintaining a supportive community and short release cycle. We aim at improving the support by teaming up with YAP.

Acknowledgements

SWI-Prolog is a community. The text above already mentions the contributors of important parts of the system. Bob Wielinga is not only an active user, but also ensured projects in which SWI-Prolog could be deployed and developed. Paulo Moura builds the MacOS Intel binaries, provides Logtalk [2] and promotes standards in the Prolog community including the multi-threading API. Richard O'Keefe is invaluable in answering Prolog design issues on the mailinglist and providing background information for motivating built-ins and library primitives.[12] Bart Demoen is invaluable explaining implementation tricks, especially those that have not been published. Ulrich Neumerkel recently performed exhaustive tests, that helped fixing many obscure bugs.

References

1. D. L. Bowen, L. M. Byrd, and WF. Clocksin. A portable Prolog compiler. In L. M. Pereira, editor, *Proceedings of the Logic Programming Workshop 1983*, Lisabon, Portugal, 1983. Universidade nova de Lisboa.
2. Paulo Moura. *Logtalk - Design of an Object-Oriented Logic Programming Language*. PhD thesis, Department of Informatics, University of Beira Interior, Portugal, September 2003.
3. Ulrich Neumerkel. The binary wam, a simplified prolog engine. Technical report, Technische Universität Wien, 1993. http://www.complang.tuwien.ac.at/ulrich/papers/PDF/binwam-nov93.pdf.
4. I. V. Ramakrishnan, Prasad Rao, Konstantinos Sagonas, Terrance Swift, and David S. Warren. Efficient tabling mechanisms for logic programs. In Leon Sterling, editor, *Proceedings of the 12th International Conference on Logic Programming*, pages 697–714, Cambridge, june 1995. MIT Press.

[12] http://www.cs.otago.ac.nz/staffpriv/ok/pllib.htm

5. Jan Wielemaker. Native preemptive threads in SWI-Prolog. In Catuscia Palamidessi, editor, *Practical Aspects of Declarative Languages*, pages 331–345, Berlin, Germany, december 2003. Springer Verlag. LNCS 2916.
6. Jan Wielemaker. An overview of the SWI-Prolog programming environment. In Fred Mesnard and Alexander Serebenik, editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1–16, Heverlee, Belgium, december 2003. Katholieke Universiteit Leuven. CW 371.
7. Jan Wielemaker and Anjo Anjewierden. An architecture for making object-oriented systems available from Prolog. In Alexandre Tessier, editor, *Computer Science, abstract*, 2002. http://lanl.arxiv.org/abs/cs.SE/0207053.
8. Jan Wielemaker and Anjo Anjewierden. PlDoc: Wiki style literate programming for Prolog. In Patricia Hill and Wim Vanhoof, editors, *Proceedings of the 17th Workshop on Logic-Based methods in Programming Environments*, pages 16–30, 2007.
9. Jan Wielemaker, Michiel Hildebrand, and Jacco van Ossenbruggen. Using Prolog as the fundament for applications on the semantic web. In S.Heymans, A. Polleres, E. Ruckhaus, D. Pearse, and G. Gupta, editors, *Proceedings of the 2nd Workshop on Applicatiions of Logic Programming and to the web, Semantic Web and Semantic Web Services*, pages 84–98, 2007.
10. Jan Wielemaker, Zhisheng Huang, and Lourens van der Mey. SWI-Prolog and the Web. Accepted for publication in tplp, HCS, University of Amsterdam, 2006.
11. Jan Wielemaker, Guus Schreiber, and Bob Wielinga. Prolog-based infrastructure for RDF: performance and scalability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - Proceedings ISWC'03, Sanibel Island, Florida*, pages 644–658, Berlin, Germany, october 2003. Springer Verlag. LNCS 2870.