Evolution of Prolog Compilers for Multiprocessor Systems and GRIDs

Ecaterina–Irina Grosu*, Dan Mancas, Nicolae–Iulian Enescu***, Ileana Hamburg****,**

**CS AEIC SA, Craiova, Romania,*
(e-mail: catya_ace@yahoo.com)
***Faculty of Automation, Computers and Electronics, University of Craiova,*
Craiova, Romania, (e-mail: dmancas@ucv.ro)
**** Faculty of Automation, Computers and Electronics, University of Craiova,*
Craiova, Romania, (e-mail nenescu@cs.ucv.ro)
*****Institut Arbeit und Technik, Gelsenkirchen, Germany*
(e-mail hamburg@iat.eu)

Abstract: Starting with the first interpreter written in 1972 and then the de facto standard model for Prolog, the story of this logic language is continuing even today, integrating itself in the new techniques. As nowadays the complexity of software systems is increasing and the attention is on extensions and tailoring for more flexibility, the implementation in such a language is a proposal that may reach efficiency faster than object-oriented languages. In this idea, different open source projects regarding compiling and interpreting of this language were launched and their development will be presented in the paper.

Keywords: architecture, software, efficiency, performance, programming, compiler, language.

1. INTRODUCTION

In the Introduction chapter, a presentation of the two main ideas, Prolog and GRID, will be made.

1.1 Prolog

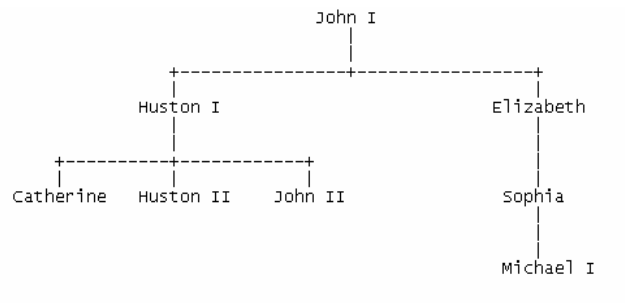
Prolog is a general purpose logic programming language that associates with artificial intelligence and computational linguistics. The name is the short for PROgramming LOGic.

Prolog has its main ground in formal logic, and it is a declarative programming language, that is its syntax uses facts and rules expressed by terms of relations to create the logic. When a result is needed, a query is made in order for the logic to perform the computation over the relations to offer the result.

The language was first conceived by a group around Alain Colmerauer in Marseille, France, in the early 1970s and the first Prolog system was developed in 1972 by Alain Colmerauer and Phillipe Roussel.

Prolog was one of the first logic programming languages, and remains among the most popular such languages today, with many free and commercial implementations available. If initially it aimed at natural language processing, the language has since then developed into other areas like theorem proving, expert systems, games, automated answering systems, ontologies and sophisticated control systems. Modern Prolog environments support the creation of graphical user interfaces, as well as administrative and networked applications, as Kowalski, R. A..sais in “The early years of logic programming“.

As a simple example we have the family tree:



The rules would then be:

male(john1).
male(huston1).
male(huston2).
male(john2).
male(michael1).
female(catherine).
female(elizabeth).
female(sophia).

```
parent(huston1, john1).
parent(elizabeth, john1).
parent(huston2, huston1).
parent(catherine, huston1).
parent(john2, huston1).
parent(sophia, elizabeth).
parent(michael1, sophia).
```

1.2 GRID

The term grid computing originated in the early 1990s as a metaphor for making computer power as easy to access as an electric power grid in Ian Foster's and Carl Kesselman's seminal work, "The Grid: Blueprint for a new computing infrastructure" (1999).

The first definition was given by them as in “A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.”

In the same book Ian Foster appraised that Grids have moved from the obscurely academic to the highly popular. The information we can read about is vast and gathers field like Compute Grids, Data Grids, Science Grids, Access Grids, Knowledge Grids, Bio Grids, Sensor Grids, Cluster Grids, Campus Grids, Tera Grids, and Commodity Grids.

Another idea that appeared in the skeptic minds is if there is more to the Grid than, as one wag put it, a “funding concept” and, as industry becomes involved, a marketing slogan, Ian Foster said.

In a later publication, Ian Foster opens a checklist in order to better define the term Grid:

1) coordinates resources that are not subject to centralized control …

2) … using standard, open, general-purpose protocols and interfaces

3) … to deliver nontrivial qualities of service.

1.3 Our context

The Rogrid Consortium was set up in May 2002, at the initiative of the Ministry for Communication and Information Technology and the Ministry of Education and Research.

It has as main objectives:

• To increase awareness about Romanian Grid activities and benefits among potential users

• To encourage and facilitate the involvement of other interested and competent institutions nation wide

• To support the development of the Romanian Grid integrated project as a consistent and coherent part of the European R&D activity in this field.

The RoGRID Consortium represents as the National Grid initiative in the SEE-GRID eInfrastructure for regional eScience (SEE-GRID-SCI) is a European Commission co-founded project.

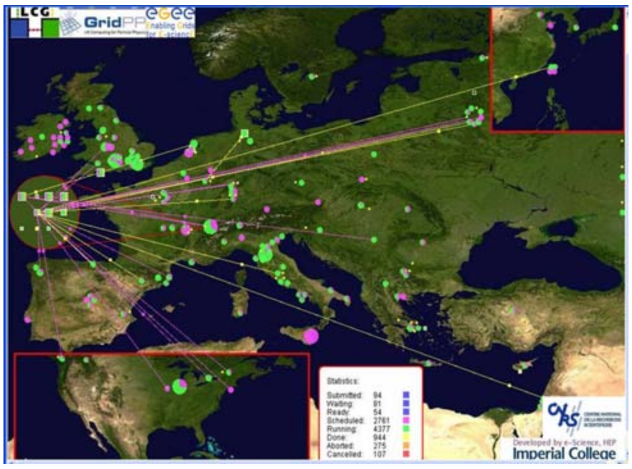


Fig. 2. GRID distribution in Europe

SEE-GRID intends to provide specific support actions to pave the way towards the participation of the SE European countries to the Pan-European and worldwide Grid initiatives.

SEE-GRID-SCI spans across 14 countries and operates 35 sites, from a mixture of research centres, Universities, companies and other interested bodies. The infrastructure runs on a wide range of different hardware, at present almost all computers in the SEE-GRID-SCI infrastructure run the CERN (The European Organization for Nuclear Research (French: Organisation Européenne pour la Recherche Nucléaire)) version of Scientific Linux.

In this context, it will be available for us a GRID center where we can test certain Prolog engines and further on implement our own that will work on multiprocessor systems.

2. ENGINES FOR PROLOG

In order to integrate the use of the Prolog language in the nowadays technologies, the development of compilers and interpreters that use GRID as a hardware platform has begun.

The nowadays implementation relate to the *old* single – processor compilers.

A parallel presentation of the objectives and requirements over these two types of compilers will be made, starting with some single-processor compilers presentation, and ending with one of the few usable multiprocessor compilers for Prolog.

2.1 tuProlog

tuProlog (also known as 2P) is an Open Source Prolog engine, as it is shown in the project's home page.

The project was built on top of the Java Virtual Machine. The design was created to provide logic-based technology as a core ingredient for Internet application components and infrastructures.

As in this context, logic programming languages have already proved to be effective both as communication and as coordination tools. Also, in facing specific issues like security in a declarative way the logic programming languages reach effectiveness.

As Internet application have a certain type of requirements, the purpose of playing a key role in the development and deployment of this type of applications and infrastructures, logic-based components were implemented.

The engineering requirements that had to be followed were respected accordingly so that tuProlog has been designed with scalability and interoperability in mind, and to be easily deployable, light-weight, statically and dynamically configurable.

As in the project presentation, the key properties that tuProlog has been created to feature are:

Minimality: containing only the most essential characteristics of a Prolog engine, its pure inferential core is available as a Java object. Therefore, tuProlog is as thin and light-weight as possible. After this, the user needs are reflected in configurations that allow the required Prolog features (e.g. I/O predicates, DCG operators) to be added to or removed from a tuProlog engine. This property is particularly relevant for use in small devices such as PDAs or mobile phones.

Configurability: choosing minimality as a first feature for the tuProlog project a high degree of configurability is its necessary counterpart. Moreover, the provided feature, configurability, was made as to be dynamic, in order to face the continuous changes especially in the Internet environment, so that uniformly, static and dynamic component were developed and enabled. As the granularity is made at the library level, one can load and unload features such as recognition of various syntax elements, predicates, functors and operators in the core engine, as said earlier, in a static and/or dynamic fashion. Until now but also further on, developing libraries was made using either Prolog, or Java, or both languages, and can be either employed to configure a tuProlog engine when it is started up, or loaded (and then unloaded) dynamically at any time during the engine execution. At the moment, five libraries are included in the standard tuProlog distribution, that provide functionalities that correspond to the ISO Prolog Standard predicates; new libraries can be defined by the tuProlog user or developer as well, as using tuProlog would mean knowing at least Prolog, so that one can implement such a library in Prolog or Java.

Deployability: being developed in Java made tuProlog easy to deploy but also portable. All one needs to start is the standard Java Virtual Machine, and a Java invocation upon a single JAR file that will load the context and finally launch the project.

Interoperability: Interoperability is one of the features composed by other two already stated and achieved, and was from the beginning an important objective. So interoperability is achieved by respecting Internet standard patterns and by coordinating models. Another part of interoperability is the interaction with different media, as the Internet is a dynamic environment. For this, TCP/IP and RMI are supported, but also, tuProlog engine can also be provided as a CORBA service. But as interaction is a complex matter and using this approached may narrow the possibilities for coupling with the benefit of preventing more complex form of coordination other than peer-to-peer models to be enacted, in order not to disturb the users' needs and activity, tuProlog provides the possibility to adopt a logic-based abstraction as a unifying interaction metaphor:

- components of a tuProlog application can be organized around Java-based tuple spaces, logic tuple spaces, and ReSpecT tuple centres;
- then, tuProlog applications can exploit Internet infrastructures providing tuple-based coordination services, such as TuCSoN or LuCe.

Between the declarative/logic and the imperative/object-oriented programming paradigms to be found in the Prolog and Java worlds, the tuProlog creates a bound, an integration scheme that is found to be the most appealing to applications and systems developers as it is a full, bidirectional, easy-to-use integration scheme.

During the later development for the recent release, tuProlog's internal architecture has been restructured, to allow for more ease of both extension and modification. Also, sound engineering principles, such as object-oriented code structuring, reuse of established community knowledge under the form of patterns, loose coupling of composing elements, modularity, and a clear and clean separation of concerns were taken into account, as the tuProlog's architecture has been based upon a set of managers, operating around a minimal core shaped as a Finite State Machine, and handling control of sensible parts of the engine, such as built-in primitives, predicate libraries, and logic theories.

The result is giving rise to deeper flexibility and modification ability|, two properties that, to a certain extent, have always represented a strong asset on tuProlog's appealing side.

As a consideration, especially during the latest years, tuProlog has been involved as a basic component in a number of research projects who did benefit from its pliable nature.

For instance, tuProlog has been integrated into the DCaseLP environment for building heterogeneous multi-agent systems, thanks to the core extendibility provided by dynamically loadable predicate libraries; the engine's unification algorithm, distributed across the classes representing the Prolog terms hierarchy, in true object-oriented fashion, has been modified to support the PRACTIONIST framework for developing agents according to the Belief-Desire-Intention (BDI) model; the whole tuProlog has been tweaked to implement the AtuP argumentation engine as a non-monotonic reasoning component in Internet or agent-based applications.

Fig. 2. *aliCE research group,* Alma Mater Studiorum

tuProlog is developed and maintained by the aliCE research group at the Alma Mater Studiorum{Universita di Bologna, site of Cesena)

2.3 GNU Prolog

GNU Prolog is a free Prolog compiler with constraint solving over finite domains developed by Daniel Diaz.

GNU Prolog uses Prolog and constraint programs and produces native binaries (like gcc does from a C source). A main feature of GNU Prolog is that the obtained executable is then stand-alone. The size of this executable can be quite small since GNU Prolog can avoid linking the code of most unused built-in predicates.

The Prolog part of the GNU Prolog project assures two objectives:

- conforms to the ISO standard for Prolog
- offers many extensions very useful in practice (global variables, OS interface, sockets,...).

GNU Prolog also includes an efficient constraint solver over Finite Domains (FD). In this way, the user passionate about constrain logic programming can work combining the power of constraint programming to the declarativity of logic programming.

Its features are:

Prolog system:

- conforms to the ISO standard for Prolog (floating point numbers, streams, dynamic code,...).
- a lot of extensions: global variables, definite clause grammars (DCG), sockets interface, operating system interface,...
- more than 300 Prolog built-in predicates.
- Prolog debugger and a low-level WAM debugger.
- line editing facility under the interactive interpreter with completion on atoms.
- powerful bidirectional interface between Prolog and C.

Compiler:

- native-code compiler producing stand alone executables.
- simple command-line compiler accepting a wide variety of files: Prolog files, C files, WAM files,...
- direct generation of assembly code 15 times faster than wamcc + gcc.
- most of unused built-in predicates are not linked (to reduce the size of the executables).
- compiled predicates (native-code) as fast as wamcc on average.
- consulted predicates (byte-code) 5 times faster than wamcc.

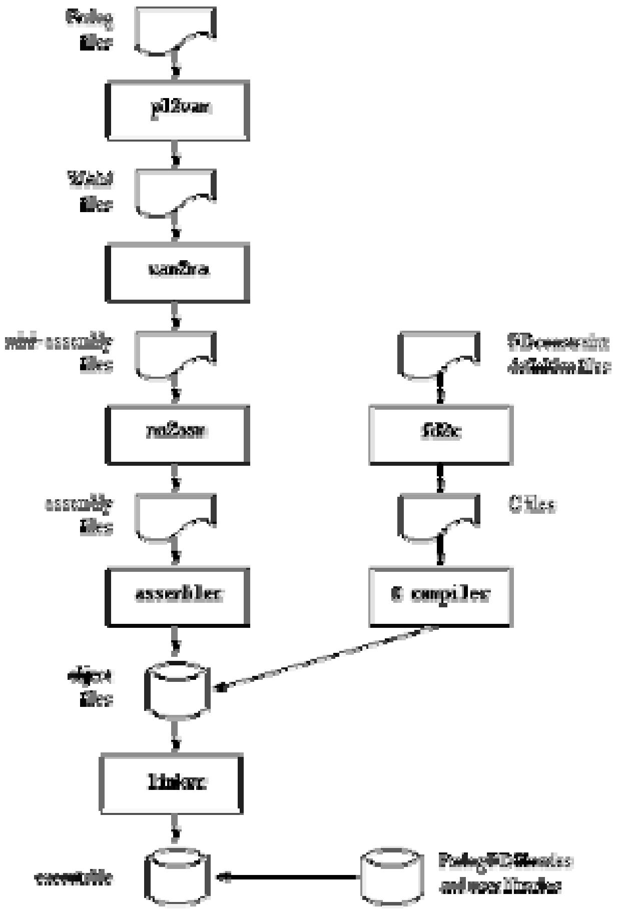


Fig. 3. Compilation: Compilation scheme

Constraint solver:

- FD variables well integrated into the Prolog environment (full compatibility with Prolog variables and integers). No need for explicit FD declarations.
- very efficient FD solver (comparable to commercial solvers).
- high-level constraints can be described in terms of simple primitives.
- a lot of predefined constraints: arithmetic constraints, boolean constraints, symbolic constraints, reified constraints, ...
- several predefined enumeration heuristics.
- the user can define his own new constraints.
- more than 50 FD built-in constraints/predicates.

The GNU Prolog compiler is based on the Warren Abstract Machine (WAM). In order to reach at the native stand alone executable, the Prolog program takes the WAM file and translates it to a low-level machine independent language called mini-assembly specifically designed for GNU Prolog. This mini-assembled file is then transformed in an object file by being translated to the assembly language of the target machine. This allows GNU Prolog to produce a native stand alone executable from a Prolog source (similarly to what does a C compiler from a C program). As the code of most unused built-in predicates can be excluded from the executables at link-time the GNU Prolog program can reach its main advantage of this compilation scheme, which is to produce native code and to be fast resulting in a small executable file.

2.3 LOGFLOW

The progress of this kind of compilers comes with the advancements in multiprocessor systems. As to fit the new technologies, compilers designed especially for these systems are being developed.

One project that reached into our attention is the LOGFLOW system.

According to its official description, LOGFLOW is a distributed Prolog system running on multi-transputer machines and workstation clusters. The 3DPAM engines (Distributed Data Driven Prolog Abstract Machines) execute the predicates of a Prolog program in parallel based on a fine-grain data driven execution scheme. Coarse- grain pieces of work are executed by traditional sequential WAM (Warren Abstract) machines, as it presented in "MOGUL: A Graphical Environment for Developing the LOGFLOW Parallel Prolog System" paper.

The main design issues in the development of LOGFLOW were as follow:

- correctness of the dataflow execution mechanism
- combining 3DPAM and WAM engines
- compiling Prolog programs into abstract codes both for the 3DPAM and WAM engines
- mapping of 3DPAM and WAM engines into the distributed processor space
- load balancing
- granularity control
- performance measurements

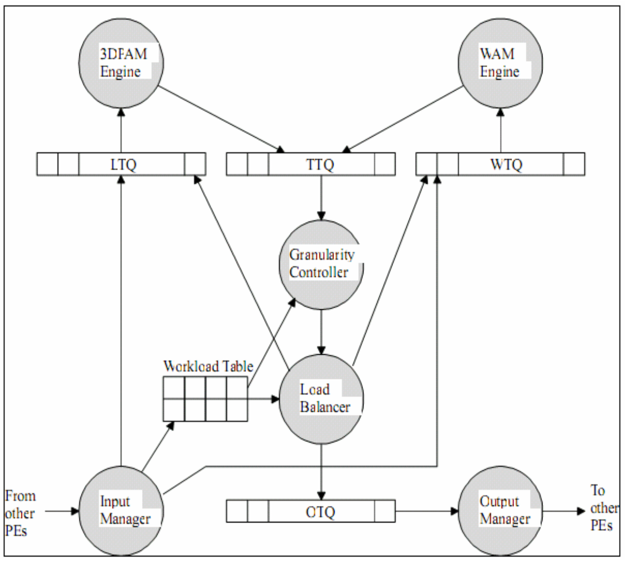


Fig. 4. LOGFLOW: structure of a processing element

In order to obtain all the design issues for the system, different implementations are made.

MOGUL is one of the implementations presented also in MOGUL: A Graphical Environment for Developing the LOGFLOW Parallel Prolog System" paper and is defined to be a framework for creating Prolog programs and makes LOGFLOW available for users other than the developers.

Basically, MOGUL is a framework for visualizing the relevant features of the parallel system design and has as main advantage of releasing the programmer in dealing with the collection of necessary information during development, so that the user can concentrate on the relevant Prolog related development issues.

In order to support the system designers or the Prolog application programmers, MOGUL offers visualization techniques and evaluation methods of the results.

MOGUL is based on the concept of projects. A project consists of the following steps:

1. Editing the Prolog program
2. Compiling the Prolog program
3. Executing the Prolog program
4. Visualizing the execution of the Prolog program

a. Animating the dataflow execution

b. Animating the decisions of the load balancer and granularity controller

We shall analyse the *compiling* part, as it is our point of discussion.

As defined, the main goal of the compiler window is to control and access the compiler program that can be located on any other machine in the network.

Without the use of **MOGUL** one should:

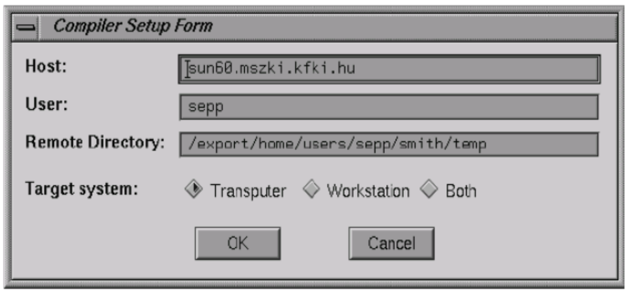
• copy the source files to a remote machine,

• login that machine,

• execute both compilers to create 3DPAM and WAM abstract code (with several additional files)

• execute the program which makes the connection between the two abstract codes (so the 3DPAM engine can do a sequential query for the WAM engine and can convert all the necessary data for this action)

• copy the result files back


Compiler Setup Form

Host: sun60.mszki.kfki.hu

User: sepp

Remote Directory: /export/home/users/sepp/smith/temp

Target system: ◆ Transputer ◇ Workstation ◇ Both

Fig. 5. Compiler: The compiler OPTIONS window

In the OPTIONS menu the user is allowed to make the following settings, see Figure 5:

• Address of remote machine (Host), where the compilers are placed

• Username for login the remote machine (User)

• Temporary directory in which the files are to be copied (Remote Directory)

• Target system for which the Prolog program is to be compiled (Transputer, Workstation, Both)

The target system should be selected before the compilation and not before the execution.

LOGFLOW implementations on different architectures use different WAM engines and therefore, different WAM compilers are invoked.

6. CONCLUSIONS

In an emerging technological development for applications on multiprocessor systems, the introduction of the logical programming language could lead to a very fast advancement and increase of performance.

The effect of user notation leads to this conclusion.

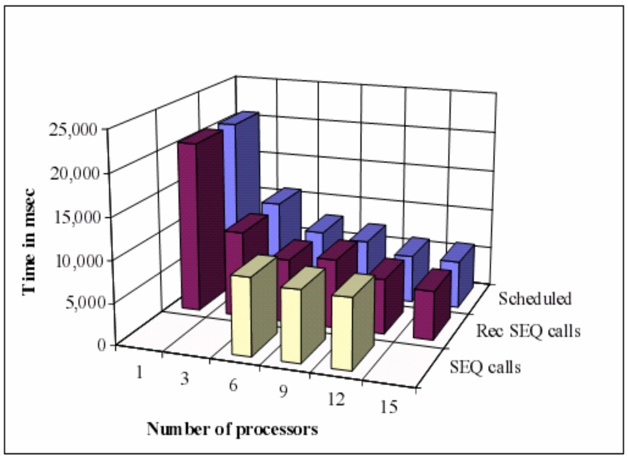


Fig. 6. User notations: Performance using multiprocessors

We sign on to follow this results and the development of these applications, following we will implement our own Prolog compiler to match or, why not, overcome the results of others cited in this paper.

REFERENCES

Diaz, D., (1996). *GNU Prolog*, Internet, http://www.gprolog.org/.

Foster, I. and Kesselman, C., (2002). *What is the GRID? A three point checklist*, GRIDToday, July 20, 2002

Piancastelli, G. and Omicini, A. (2007). tuProlog 2.0: One step beyond *ALP Newsletter Digest 20(1).*

Kacsuk, P. (2007), *Granularity Control In The Logflow Parallel Prolog System.*

Kacsuk, P., Kovács, J. And Podhorszki, N. (1997) *MOGUL: A Graphical Environment for Developing the LOGFLOW Parallel Prolog System*, ILPS'97 www.wikipedia.org