The Ciao Multiparadigm Language and Program Development Environment

The Ciao Development Team

Ciao is a modern, multiparadigm programming language with an advanced programming environment. It has a dual nature: on one hand it provides a high-performance, industrial quality, freely available, *ISO-standard-compliant* Prolog system. At the same time, its modular approach allows both *restricting* and *augmenting* the language through *libraries* in a well-controlled fashion. This allows providing significant extensions which make Ciao a truly *next-generation logic-programming language* as well as a *multiparadigm programming system.*

One of fundamental aspects of the Ciao approach is based on the observation that a single set of basic, well-chosen features (a *language kernel*) can effectively support several programming paradigms and styles [11,10]. This approach is, of course, not exclusive to Ciao, but in Ciao these facilities are uniformly available (and their use encouraged) from the system programmer level to the application programmer level.

The extensibility-based approach makes it possible to work with *fully declarative subsets* of logic programming and also to *extend* the core language both syntactically and semantically. Most importantly, these restrictions and extensions can be activated separately on a per-module basis without interfering with each other thanks to the notion of *packages* [4]. The different source-level constructs (and sub-languages / DSLs) are typically supported by compilation into the kernel language via *source-to-source* transformations, with the (rather infrequent) help of modules written in an external language using one of the several interfaces provided. Due to the existence of a common kernel, the programming styles that Ciao implements share much at both the semantic and implementation levels, and they naturally reuse significant portions of modern (C)LP implementation technology.

Ciao provides support for both *programming in the small* (by providing *scripts* and reduced-size executables, which include only those builtins and libraries used by the program) and *programming in the large.* Programming in the large is facilitated by its robust module system [4] and rich *assertion language* (another product of the "extensibility approach") combined with the capabilities of the Ciao preprocessor [2,12,13] for modular static verification, static debugging, and dynamic checking of such assertions.

Some of the principal distinguishing features of Ciao are:

ISO-Prolog: Ciao provides in its default mode an excellent Prolog system, without giving up on all of its "new-generation" features, thanks to the library-based approach. This distinguishes Ciao from other new-generation (C)LP systems that do not have an ISO-Prolog compliant mode.

Support for Multiple Programming Models and Paradigms: At the same time Ciao supports, *also via libraries*, different LP languages, several types of constraint domains (including CHR support), functional notation (including lazy evaluation), higher-order (with predicate/function abstractions), as well as several computation rules (Andorra model, breadth-first, iterative deepening, fuzzy Prolog) and object-oriented programming facilities. Modules written in ISO-Prolog can be combined freely with other modules written in these other supported paradigms, and some paradigms can even be mixed within the same module.

Powerful Preprocessor: A number of program analyzers, integrated in the Ciao preprocessor, allow inferring and checking many useful program properties. Most analyses are performed at the kernel language level, so that the same analyzers can be used for several of the supported programming models. The availability of this information provides quite unique functionality [13]:

- **Assertions and Program Debugging/Validation:** The properties that are statically inferred by the analyzers can be compared against programmer-provided (typically partial) specifications written using Ciao's unique *assertion language*. These properties include types/modes (data structure shape –including variable sharing– and instantiation state of variables), bounds on data sizes, determinacy, termination, non-failure, or bounds on resource consumption (time, space, or user-defined resources). The preprocessor *verifies* whether those properties are met by the actual code and *statically debugs* the program otherwise. Assertions that the system cannot prove nor disprove at compile-time can be optionally subject to *run-time checking*, with tests automatically added to the program. Both static and dynamic checking are safe in the sense that all errors flagged are violations of the specifications.
- **Assertions are not Compulsory:** A fundamental advantage of the Ciao approach in this context is that assertions *are not compulsory*. This distinguishes Ciao from other new-generation (C)LP systems where, e.g., type definitions or declarations are compulsory, and is of course instrumental in allowing Ciao to support Prolog.
- **Mobile Code Safety through Abstraction-Carrying Code:** When programs are validated the preprocessor can generate automatically certificates that can be attached to programs to be checked at the receiving end in order to guarantee compliance with a given safety policy [1].
- **Source-to-source Optimizations:** The information inferred by the global analyzers can be used to perform source-level code optimizations, including multiple abstract specialization, partial evaluation, dead code removal, goal reordering, parallelization with granularity control, reduction of concurrency / dynamic scheduling, low-level optimization, etc.

Rich Program Development Environment: In addition to all the facilities provided by the preprocessor, compiler, and top level, the programming environment includes:

- **Rich Graphical Development Interface:** based on the latest, graphical versions of Emacs (offering menu and widget-based interfaces with direct access to the top-level/debugger, preprocessor, and autodocumenter) as well as an embeddable source-level debugger with breakpoints, and several execution visualization tools. The environment provides also automated access to the documentation, extensive syntax highlighting, auto-completion, auto-location of errors in the source, etc., and is highly customizable. A plugin with very similar functionality is also available for the Eclipse environment.
- **Automatic Documentation Generation:** The assertions and directives present in the program and libraries, as well as all other program information available to the compiler, are used to generate automatically program documentation (including types, modes, machine-readable comments, etc.) by means of the Ciao autodocumenter [9].

Versatile, Incremental Compiler and Abstract Machine: A central piece of the system is the Ciao compiler, `ciaoc` [5] and its target abstract machine, which offer:

- **Modular, Incremental Compilation:** `ciaoc` performs automatically an incremental compilation which takes module dependencies into account without the need for *Makefiles*. Program modules can be linked statically, dynamically, or be automatically loaded on demand.
  The executables generated are competitive in both performance and size with current commercial and academic Prolog systems.
- **Versatile, Multiplatform Compilation Options:** Several types of executables can be built easily. In addition to the traditional Prolog top-level, the system offers support for the use of Ciao as a scripting language, for compilation to multiarchitecture *bytecode* executables, and for compilation to single-architecture, standalone executables. Multiple platforms are supported, including Windows, Linux, Mac Os X, and many other Un*x-based OSs. Optimizing compilation to native code (via C) is in a mature state [8] and will be part of the standard distribution in the near future.
- **Kernel Abstract Machine:** A comparatively simple, but optimized abstract machine supports the kernel language. It includes native support for *attributed variables* and for threading primitives, and a synchronization-enabled shared database [7].
- **Rich foreign interfaces:** Bidirectional foreign interfaces to C (with automatic generation of glue code), Java, TclTk, SQL databases (with a notion of predicate persistence), etc.

Support for Concurrency, Parallelism, and Distributed Execution: Ciao includes concurrency, parallelism, and distributed execution capabilities [3]. The notion of "active module" (or active object) allows compiling modules in such a way that they are ultimately mapped to a standalone process, which is transparently accessed by the rest of the application. In addition, the system also offers a full-fledged library for developing WWW-based applications [6].

Free Availability: Ciao is free software protected to remain so by the GNU LGPL license. It can be used freely to develop both free and commercial applications.

Finally, the system includes a **large set of libraries**, many of them contributed by users.

Probing Further

The reader is encouraged to explore the system, its documentation, and the tutorial papers that have been published on it. We are currently working on the new 1.14 system version which includes significant enhancements with respect to the previous version (1.10), including integration of the preprocessor and autodocumenter into the Ciao development tree as a single package (previously they had to be downloaded and installed separately). This version is available already on demand from the Ciao subversion repository.

Contact / download info:

`http://www.ciaohome.org`
`http://www.cliplab.org`
`ciao@clip.dia.fi.upm.es`

The Ciao Development Team
Technical U. of Madrid, Spain
U. of New Mexico, USA
U. Complutense de Madrid, Spain

Acknowledgments

The Ciao system is in continuous and very active development through the collaborative effort of numerous members of several institutions, including UPM, UNM, UCM, Roskilde U., U. Melbourne, Monash U., U. Arizona, Linköping U., NMSU, K. U. Leuven, Bristol U., Ben-Gurion U, INRIA, as well as many others. The development of the Ciao system has been supported by a number of European, Spanish, and other international projects (currently by EU IST-15905 *MOBIUS* project, the Spanish TIN-2005-09207 *MERIT* project, and the CAM *PROMESAS* program. Manuel Hermenegildo is also supported by the IST Prince of Asturias Chair at the University of New Mexico. The system documentation and related publications contain more specific credits for the many contributors to the system.

References

1. E. Albert, G. Puebla, and M. Hermenegildo. Abstraction-Carrying Code. In *Proc. of LPAR'04*, number 3452 in LNAI, pages 380–397. Springer-Verlag, 2005.
2. F. Bueno, P. Deransart, W. Drabent, G. Ferrand, M. Hermenegildo, J. Maluszynski, and G. Puebla. On the Role of Semantic Approximations in Validation and Diagnosis of Constraint Logic Programs. In *Proc. of the 3rd. Int'l Workshop on Automated Debugging–AADEBUG'97*, pages 155–170, Linköping, Sweden, May 1997. U. of Linköping Press.

3. D. Cabeza and M. Hermenegildo. Distributed Concurrent Constraint Execution in the CIAO System. In *Proc. of the 1995 COMPULOG-NET Workshop on Parallelism and Implementation Technologies*, Utrecht, NL, September 1995. U. Utrecht / T.U. Madrid. Available from `http://www.cliplab.org/`.
4. D. Cabeza and M. Hermenegildo. A New Module System for Prolog. In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 131–148. Springer-Verlag, July 2000.
5. D. Cabeza and M. Hermenegildo. The Ciao Modular, Standalone Compiler and Its Generic Program Processing Library. In *Special Issue on Parallelism and Implementation of (C)LP Systems*, volume 30(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier - North Holland, March 2000.
6. D. Cabeza and M. Hermenegildo. Distributed WWW Programming using (Ciao-)Prolog and the PiLLoW Library. *Theory and Practice of Logic Programming*, 1(3):251–282, May 2001.
7. M. Carro and M. Hermenegildo. Concurrency in Prolog Using Threads and a Shared Database. In *1999 International Conference on Logic Programming*, pages 320–334. MIT Press, Cambridge, MA, USA, November 1999.
8. M. Carro, J. Morales, H.L. Muller, G. Puebla, and M. Hermenegildo. High-Level Languages for Small Devices: A Case Study. In Krisztian Flautner and Taewhan Kim, editors, *Compilers, Architecture, and Synthesis for Embedded Systems*, pages 271–281. ACM Press / Sheridan, October 2006.
9. M. Hermenegildo. A Documentation Generator for (C)LP Systems. In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 1345–1361. Springer-Verlag, July 2000.
10. M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla. The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, Commack, NY, USA, April 1999.
11. M. Hermenegildo and The CLIP Group. Some Methodological Issues in the Design of CIAO - A Generic, Parallel, Concurrent Constraint System. In *Principles and Practice of Constraint Programming*, number 874 in LNCS, pages 123–133. Springer-Verlag, May 1994.
12. M. Hermenegildo, G. Puebla, and F. Bueno. Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging. In K. R. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: a 25–Year Perspective*, pages 161–192. Springer-Verlag, July 1999.
13. M. V. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). *Science of Computer Programming*, 58(1–2):115–140, October 2005.

Most of these and other papers and technical reports related to Ciao can be obtained from the Clip lab main WWW server, `http://www.cliplab.org/`.