An Intro to Execution Models and Environments Through the Ciao System

Manuel Hermenegildo
herme@fi.upm.es
http://www.clip.dia.fi.upm.es/herme

IMDEA Software
Tech. University of Madrid
U. of New Mexico

The UPM work presented is a joint effort with members of the CLIP group at the UPM School of Computer Science and IMDEA Software including: Francisco Bueno, Daniel Cabeza, Manuel Carro, Amadeo Casas, Pablo Chico, Jess Correas, María José García de la Banda, Manuel Hermenegildo, Pedro López, Mario Mndez, Edison Mera, Jos Morales Jorge Navas, and Germán Puebla.

Motivation and Approach

- Objective (one of the main (and hardest) challenges in CS):
  - *Facilitating* the process of *building complex software components/systems*, *in as short a time as possible,*
  - while obtaining guaranteed *safety, reliability, and efficiency.*
- Approach:
  - Develop (CLP-based) next-generation, high-level, *multiparadigm* programming languages and execution models.
  - *Develop improved progr. devel. environments,* which perform *as part of compilation*:
    - Verification / debugging, which can detect bugs or offer *guarantees* of *safety, reliability, and efficiency.*
    - Optimization (optimized compilation, parallelization, ...).
- Use throughout techniques that are at the same time *rigorous* and *practical*.
- Apply in a concrete system, with real users –reality check! (In our case: *Ciao/CiaoPP* system.)

Modular, Evolvable Design

- Ciao is built in layers over a small, pure (LP-based) *kernel*.
  - Allows designing *syntactic and semantic extensions* in a simple, flexible, and scalable way.
  - Also, building small, fast executables and embeddability (non-needed parts of the language and libraries are not included).
- Fundamental enabler –its module/class system:
  - Designed from the ground up to be *extensible* and *analysis-friendly*.
  - Most language features (loops, conditionals, functions, ...) are not built-in, but rather in libraries.
  - Language extension libraries ("packages") affect only the modules in which they are loaded (e.g., operators and expansions are local to modules, etc.).

→ Allows modular program development, separate/incremental compilation.
→ Allows extensive global analysis for detecting errors and optimizing code.
→ Support for programming "in the large."

Multiparadigm

- *Logic programming:*
  - Certainly ISO-Prolog (one of the best Prologs!) –but *via a library*; and also:
  - Pure LP.
  - Various comp. rules: breadth-first, iterative-deepening, Andorra, *tabling*, etc.
- *Functional programming:*
  - Function definitions and function calls and functional syntax for predicates.
  - *Higher-order* and *lazyness* for functions and predicates.
- *Constraint programming:* clpr, clpq, fd, Leuven CHR.
- *Objects:* a naturally embedded notion of classes and objects.
- *Concurrency, parallelism, distributed execution*.
- *Imperative features*: mutable data struct., assignment, loops, cases, etc.
- *Assertion language*, consistent across paradigms; with many uses!

+ many other packages: types, records, DCGs, negations, appl.-specific languages, ...

The Ciao Module System

- Ciao implements a module system [10] which meets a number of objectives:
  - High extensibility in syntax and functionality:
    allows having pure logic programming and many extensions.
  - Makes it possible to perform modular (separate) processing of program components (without “makefiles”).
  - Greatly enhanced error detection (e.g., undefined predicates).
  - Facilitates (modular) global analysis.
  - Support for meta-programming and higher-order.
  - Predicate based-like, but with functor/type hiding.

  while at the same time providing:

  - High compatibility with traditional standards (Quintus, SICStus, ...).
  - Backward compatible with files which are not modules.

Defining modules and exports

- `:- module(`*module_name*`, `*list_of_exports*`, `*list_of_packages*`).`

  Declares a module of name *module_name*, which exports *list_of_exports* and loads *list_of_packages* (packages are syntactic and semantic extensions).

- Example: `:- module(lists, [list/1, member/2], [functions]).`

- Examples of some standard uses and packages:

  - `:- module(`*module_name*`, [`*exports*`], []).`
    ⇒ Module uses (pure) kernel language.

  - `:- module(`*module_name*`, [`*exports*`], [`*packages*`]).`
    ⇒ Module uses kernel language + some packages.

  - `:- module(`*module_name*`,[`*exports*`], [functions]).`
    ⇒ Functional programming.

  - `:- module(`*module_name*`,[`*exports*`],[assertions,functions]).`
    ⇒ Assertions (types, modes, etc.) and functional programming.

Defining modules and exports (Contd.)

- (ISO-)Prolog:
  - `:- module(`*module_name*`, [`*exports*`], [iso]).`
    ⇒ Iso Prolog module.
  - `:- module(`*module_name*`,[`*exports*`], [classic]).`
    ⇒ “Classic” Prolog module
    (ISO + all other predicates that traditional Prologs offer as “built-ins”).
  - Special form:
    `:- module(`*module_name*`, [`*exports*`]).`
    Equivalent to:
    `:- module(`*module_name*`, [`*exports*`], [classic]).`
    ⇒ Provides compatibility with traditional Prolog systems.

Defining modules and exports (Contd.)

- Useful shortcuts:
  - `:- module(_, list_of_exports).`

    If given as "_" module name taken from file name (default).
    Example: `:- module(_, [list/1, member/2]).` (file is `lists.pl`)
  - `:- module(_, _).`

    If "_" all predicates exported (useful when prototyping / experimenting).
- "User" files:
  - Traditional name for files including predicates but no module declaration.
  - Provided for backwards compatibility with non-modular Prolog systems.
  - Not recommended: they are *problematic* (and, essentially, deprecated).
  - Much better alternative: use `:- module(_, _).` at top of file.
    - As easy to use for quick prototyping as "user" files.
    - Lots of advantages: much better error detection, compilation, optimization, ...

Importing from another module

- Using other modules in a module:
  - `:- use_module(`*filename*`).`
    Imports all predicates that *filename* exports.
  - `:- use_module(`*filename*`,`*list_of_imports*`).`
    Imports predicates in *list_of_imports* from *filename*.
  - `:- ensure_loaded(`*filename*`).` —for loading user files (deprecated).
- When importing predicates with the same name from different modules, module name is used to disambiguate:

```
:- module(main,[main/0]).
:- use_module(lists,[member/2]).
:- use_module(trees,[member/2]).

main :-
     produce_list(L),
     lists:member(X,L),
     ...
```

The Ciao Module System (Contd.)

- Some more specific characteristics [10]:
  - Syntax, flags, expansions, etc. are *local to modules*.
  - Compile-time and run-time code is clearly separated (e.g., expansion code is compile-time and does not go into executables).
  - "Built-ins" are in libraries and can be loaded into and/or unloaded from the context of a given module.
  - Dynamic parts are more isolated.
  - Directives are not queries.
  - Richer treatment of meta-predicates and higher-order.
  - The entry points to modules are statically defined.
  - Module qualification used only for disambiguating predicate names.
  - All module text must be available or in related parts.
- A resulting notion: **packages** (see later).

Compiler

- Compiler (standalone –`ciaoc`– or embedded in top level):
  - Follows dependencies and (re-)compiles modules separately automatically (no need for make- or project-style files).
  - Creates small, standalone executables and libraries.
  - Can also be used in conjunction with make-style tools.
- Produces several types of executables:
  - *Static:* all libraries included.
  - *Dynamic:* libraries loaded dynamically at run-time.
  - *Lazy load:* as dynamic, but libraries are loaded only if actually used.

  The modes above can be mixed.
- Also, Ciao "scripts":
  - Useful for small applications, CGI scripts, etc.
  - Used to make installation procedures platform-independent.

  → Support for programming "in the small."

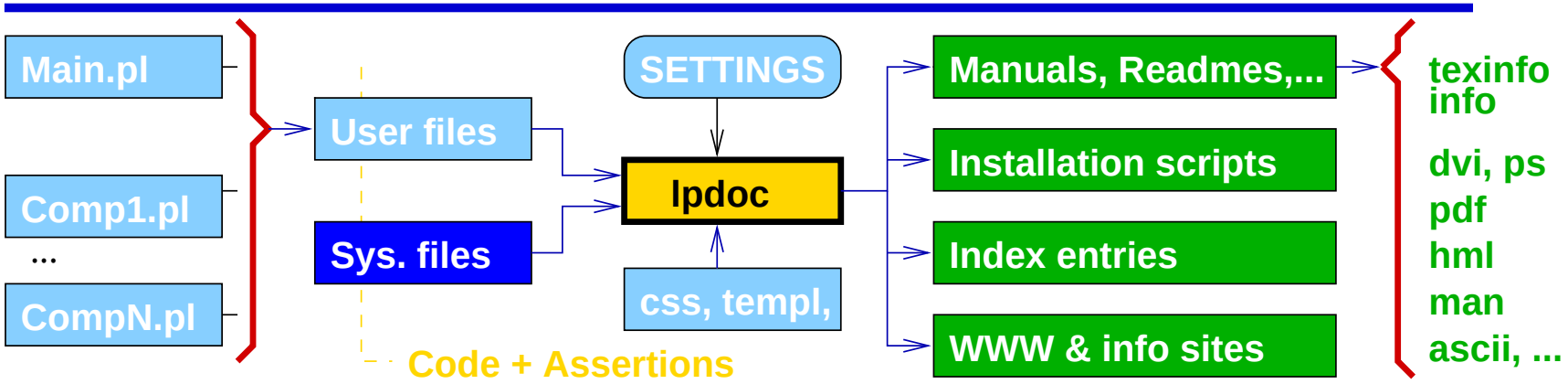
Advanced Development Environment

- *Emacs* and *eclipse* versions.
- Top-level, source debugger, standalone optimizing compiler, script interpreter, ...
- Autodocumenter, large set of libraries, ...
- Preprocessor (`ciaopp`):
  - Input: program, optionally w/assertions declaring properties such as types, modes, det, nf, sizes, cost, resources, etc.
  - Output: *error/warning messages* + *transformed program*, with
    - Results of static checking of assertions / error detection / verification. (and certificates for Abstraction Carrying Code).
    - Assertion run-time checking code.
    - High-level optimizations (specialization, slicing, parallelization).
    - Results of analysis (as assertions): used for low-level optimizations.
  - Technology: modular polyvariant abstract interpretation/specialization.

Some Features of The Ciao Development Environment

- Provides:
  - Incremental syntax highlighting of source code.
  - Direct access to on-line documentation (help and completion on what the cursor is on).
  - Direct, interactive access to compiler, top-level, preprocessor, etc.
  - Location of errors from compiler (and preprocessor) on source code.
  - Source code debugging.
  - Direct access to *auto-generation* of documentation.
  - Menu-driven access + also keyboard shortcuts and toolbar.
  - User extensible.
  - Plus many other features!
- Built as a powerful extension of `emacs`.
- Also eclipse pluings have been developed (as contribs).

Autodocumenter: LPdoc



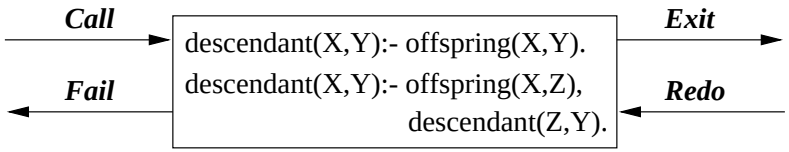
- Uses:
  - All the information that the compiler has.
  - Assertions.
  - Comment declarations:

```
:- comment(title,"Complex numbers library").
:- comment(summary,"Provides an ADT for complex numbers.").
```

- Markup language, close to LaTeX/texinfo.
  With indices, references, figures, ...

Debugging in Ciao

- The traditional interface to the “Byrd–box” debugger is available:




```
+  13  7  Call: T user:descendant(dani,_123) ?
```

- In addition, source-level tracking of the debugging process is supported:
  - Simultaneous visualization of tracing messages and byrd-box ports in the source code.
  - Placing break-points directly on the source code.
- Debugging modes can be toggled on a per-module basis: `debug_module/1`, `nodebug_module/1`, `debug_module_source/1`
- Easiest: use the Emacs environment.
- The debugger is also a library → Debugging also available in standalone executables! (see later).

Multiparadigm

- *Logic programming:*
  - Certainly ISO-Prolog (one of the best Prologs!) –but *via a library*; and also:
  - Pure LP.
  - Various comp. rules: breadth-first, iterative-deepening, Andorra, *tabling*, etc.
- *Functional programming:*
  - Function definitions and function calls and functional syntax for predicates.
  - *Higher-order* and *lazyness* for functions and predicates.
- *Constraint programming:* clpr, clpq, fd, Leuven CHR.
- *Objects:* a naturally embedded notion of classes and objects.
- *Concurrency, parallelism, distributed execution*.
- *Imperative features*: mutable data struct., assignment, loops, cases, etc.
- *Assertion language*, consistent across paradigms; with many uses!

+ many other packages: types, records, DCGs, negations, appl.-specific languages, ...

Using Other Computation Rules

- Libraries which replace the default depth-first, left to right computation rule of Ciao (and Prolog).
- Compile-time transformations ("Compiling Control" techniques).
- Useful in search problems when a complete proof procedure is needed. (e.g., for teaching pure logic programming!)
- Computation rules currently implemented:
  - Breadth-first (`bf` and `'bf/bfall'` packages).
  - Iterative-deepening (`id` package).
  - Depth-First search with limited depth (`id` package).
  - Fuzzy LP. Mycin.
  - "And-fair" breadth-first (`bf/af` package).
  - Andorra (in Beta).
  - Tabling (in Beta).
- `pure` package + `bf` (or `id` etc.) ideal for teaching LP!

CiaoPP: The *Ciao* System Preprocessor

- *CiaoPP* is a preprocessor for the standard Ciao clause-level compiler.
- Performs error detection, verification, and source-to-source transformations:
  - Input: logic program (optionally w/assertions & syntactic extensions).
  - Output: *error/warning messages* + *transformed logic program*, with
    - Results of static checking of assertions / verification.
      (and certificates for Abstraction Carrying Code).
    - Assertion run-time checking code.
    - High-level optimizations (specialization, slicing, parallelization).
    - Results of analysis (as assertions): used for low-level optimizations.
- Underlying technology:
  - Modular polyvariant abstract interpretation.
  - Modular abstract multiple specialization.
- See specific tutorial on the CiaoPP system.

Some Other Features

- “Industry standard” performance, with upcoming highly optimizing compiler.
- (Semi-automatic) interfaces to and from Ciao for C, Java, tcl-tk, etc.
- Ciao engine can be included as a (possibly dynamic) library in applications.
- Support for concurrency, parallelism, and distributed execution.
- Persistent predicate-based interface to relational databases.
- Infinite precision integers (“bignums”) and floating point numbers.
- Many others...

Why Ciao?

- Why is the system called “Ciao”?
- It is one hand an acronym:
  - *CIAO*: **C**onstraint Programming with **I**ndependent **A**nd + **O**r parallelism.
- But the name also represents the *spirit* of the system:
  - Ciao is an interesting word that means *both Hello* and *Goodbye*.
  - “Ciao Prolog:”
    - is aimed at introducing programmers to Prolog and LP/CLP
      –the “Hello Prolog” part,
    - but it also represents really a new-generation programming language and environment (with FP, HO, assertions, global analysis, objects, ...)
      –the “Goodbye Prolog” part.

Some Members of The Ciao Forge

- Ciao is really a widely distributed collaborative effort:
  - Directly within the CLIP Group:

    M. Hermenegildo, K. Muthukumar, M. García de la Banda, F. Bueno, G. Puebla, M. Carro, D. Cabeza, P. López-G., E. Albert, J. Navas, M. Méndez, A. Casas, J. Correas, J. Morales, E. Mera, D. Trallero, C. Ochoa, P. Pietrzak, P. Arenas, S. Genaim

  - Plus lots of contributors worldwide:

    G. Gupta (UT Dallas), E. Pontelli (NM State University), P. Stuckey and M. García de la Banda (Melbourne U.), K. Marriott (Monash U.), M. Bruynooghe, A. Mulkers, G. Janssens, and V. Dumortier (K.U. Leuven), S. Debray (U. of Arizona), J. Maluzynski and W. Drabent, (Linkoping U.), P. Deransart (INRIA), J. Gallagher (Roskilde University), C. Holzbauer (Austrian Research Institute for AI), M. Codish (Beer-Sheva), SICS, ...

Downloading the systems

- Downloading Ciao, CiaoPP, LPdoc, and other CLIP software:
  - Standard distributions:
    `http://www.clip.dia.fi.upm.es/Software`
  - Some betas (in testing or completing docs – ask `webmaster` for info) in:
    `http://www.clip.dia.fi.upm.es/Software/Beta`
    Mail list stored in
    `http://www.clip.dia.fi.upm.es/Mail/ciao-users/`
  - Please contact us for SVN access.
- User's mailing list:
  `ciao-users@clip.dia.fi.upm.es`

  Subscribe by sending a message with only `subscribe` in the body to
  `ciao-users-request@clip.dia.fi.upm.es`

Some Bibliography on Ciao, CiaoPP, and LPdoc

[1] E. Albert, P. Arenas, G. Puebla, and M. Hermenegildo. Reduced Certificates for Abstraction-Carrying Code. In *22nd International Conference on Logic Programming (ICLP 2006)*, number 4079 in LNCS, pages 163–178. Springer-Verlag, August 2006.

[2] E. Albert, G. Puebla, and M. Hermenegildo. Abstraction-Carrying Code. In *Proc. of LPAR'04*, volume 3452 of *LNAI*. Springer, 2005.

[3] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla. The Ciao Prolog System. Reference Manual (v1.8). The Ciao System Documentation Series–TR CLIP4/2002.1, School of Computer Science, Technical University of Madrid (UPM), May 2002. System and on-line version of the manual available at `http://www.ciaohome.org`.

[4] F. Bueno, D. Cabeza, M. Hermenegildo, and G. Puebla. Global Analysis of Standard Prolog Programs. In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.

[5] F. Bueno, M. García de la Banda, and M. Hermenegildo. Effectiveness of Abstract Interpretation in Automatic Parallelization: A Case Study in Logic Programming. *ACM Transactions on Programming Languages and Systems*, 21(2):189–238, March 1999.

[6] F. Bueno, M. García de la Banda, M. Hermenegildo, K. Marriott, G. Puebla, and P. Stuckey. A Model for Inter-module Analysis and Optimizing Compilation. In *Logic-based Program Synthesis and Transformation*, number 2042 in LNCS, pages 86–102. Springer-Verlag, March 2001.

[7] I. Caballero, D. Cabeza, S. Genaim, J.M. Gomez, and M. Hermenegildo. persdb_sql: SQL Persistent Database Interface. Technical Report CLIP10/98.0, December 1998.

[8] D. Cabeza and M. Hermenegildo. Implementing Distributed Concurrent Constraint Execution in the CIAO System. In *Proc. of the AGP'96 Joint conference on Declarative Programming*, pages 67–78, San Sebastian, Spain, July 1996. U. of the Basque Country. Available from `http://www.cliplab.org/`.

[9] D. Cabeza and M. Hermenegildo. WWW Programming using Computational Logic Systems (and the PiLLoW/Ciao Library). In *Proceedings of the Workshop on Logic Programming and the WWW at WWW6*, San Francisco, CA, April 1997.

[10] 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.

[11] 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.

[12] 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.

[13] D. Cabeza, M. Hermenegildo, and J. Lipton. Hiord: A Type-Free Higher-Order Logic Programming Language with Predicate Abstraction. In *Ninth Asian Computing Science Conference (ASIAN'04)*, number 3321 in LNCS, pages 93–108. Springer-Verlag, December 2004.

[14] D. Cabeza, M. Hermenegildo, and S. Varma. The PiLLoW/Ciao Library for INTERNET/WWW Programming using Computational Logic Systems, May 1999. See http://www.cliplab.org/Software/pillow/pillow.html.

[15] 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.

[16] M. Carro and M. Hermenegildo. A simple approach to distributed objects in prolog. In *Colloquium on Implementation of Constraint and LOgic Programming Systems (ICLP associated workshop)*, Copenhagen, July 2002.

[17] A. Casas, D. Cabeza, and M. Hermenegildo. A Syntactic Approach to Combining Functional Notation, Lazy Evaluation and Higher-Order in LP Systems. In *The 8th International Symposium on Functional and Logic Programming (FLOPS'06)*, pages 142–162, Fuji Susono (Japan), April 2006.

[18] S. K. Debray, P. López-García, M. Hermenegildo, and N.-W. Lin. Lower Bound Cost Estimation for Logic Programs. In *1997 International Logic Programming Symposium*, pages 291–305. MIT Press, Cambridge, MA, October 1997.

[19] S.K. Debray, P. López-García, and M. Hermenegildo. Non-Failure Analysis for Logic Programs. In *1997 International Conference on Logic Programming*, pages 48–62, Cambridge, MA, June 1997. MIT Press, Cambridge, MA.

[20] Thom Frühwirth. Theory and Practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1-3), October 1998.

[21] J.M. Gomez, D. Cabeza, and M. Hermenegildo. persdb: Persistent Database Interface. Technical Report CLIP9/98.0, December 1998.

[22] 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.

[23] 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.

[24] M. Hermenegildo, F. Bueno, G. Puebla, and P. López-García. Program Analysis, Debugging and Optimization Using the Ciao System Preprocessor. In *1999 Int'l. Conference on Logic Programming*, pages 52–66, Cambridge, MA, November 1999. MIT Press.

[25] M. Hermenegildo, D. Cabeza, and M. Carro. Using Attributed Variables in the Implementation of Concurrent and Parallel Logic Programming Systems. In *Proc. of the Twelfth International Conference on Logic Programming*, pages 631–645. MIT Press, June 1995.

[26] 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.

[27] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Abstract Verification and Debugging of Constraint Logic Programs. In *Recent Advances in Constraints*, number 2627 in LNCS, pages 1–14. Springer-Verlag, January 2003.

[28] M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Program Development Using Abstract Interpretation (and The Ciao System Preprocessor). In *10th International Static Analysis Symposium (SAS'03)*, number 2694 in LNCS, pages 127–152. Springer-Verlag, June 2003.

[29] M. 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.

[30] M. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey. Incremental Analysis of Logic Programs. In *International Conference on Logic Programming*, pages 797–811. MIT Press, June 1995.

[31] M. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey. Incremental Analysis of Constraint Logic Programs. *ACM Transactions on Programming Languages and Systems*, 22(2):187–223, March 2000.

[32] C. Holzbaur. Metastructures vs. Attributed Variables in the Context of Extensible Unification. In *1992 International Symposium on Programming Language Implementation and Logic Programming*, pages 260–268. LNCS631, Springer Verlag, August 1992.

[33] C. Holzbaur. *SICStus 2.1/DMCAI Clp 2.1.1 User's Manual*. University of Vienna, 1994.

[34] P. López-García and M. Hermenegildo. Efficient Term Size Computation for Granularity Control. In *International Conference on Logic Programming*, pages 647–661, Cambridge, MA, June 1995. MIT Press, Cambridge, MA.

[35] P. López-García, M. Hermenegildo, and S. K. Debray. A Methodology for Granularity Based Control of Parallelism in Logic Programs. *Journal of Symbolic Computation, Special Issue on Parallel Symbolic Computation*, 21(4–6):715–734, 1996.

[36] J. Morales and M. Carro. Improving the Compilation of Prolog to C Using Type Information: Preliminary Results. In M. Carro, C. Vaucheret, and K.-K. Lau, editors, *Proceedings of the CBD 2002 / ITCLS 2002 CoLogNet Joint Workshop*, pages 167–180, School of Computer Science, Technical University of Madrid, September 2002. Facultad de Informatica.

[37] J. Morales, M. Carro, G. Puebla, and M. Hermenegildo. A Generator of Efficient Abstract Machine Implementations and its Application to Emulator Minimization. In Maurizio Gabbrielli and Gopal Gupta, editors, *International Conference on Logic Programming*, number 3668 in LNCS, pages 21–36. Springer Verlag, October 2005.

[38] J. Navas, F. Bueno, and M. Hermenegildo. Efficient top-down set-sharing analysis using cliques. In *Eight International Symposium on Practical Aspects of Declarative Languages*, number 2819 in LNCS, pages 183–198. Springer-Verlag, January 2006.

[39] S. Mu noz, J.J. Moreno-Navarro, and M. Hermenegildo. Efficient Negation Using Abstract Interpretation. In *Proc. of the Eigth International Conference on Logic Programming and Automated Reasoning*, LNAI. Springer-Verlag, December 2001.

[40] A. Pineda and M. Hermenegildo. O'Ciao: An Object Oriented Programming Model for (Ciao) Prolog. Technical Report CLIP 5/99.0, Facultad de Informática, UPM, July 1999.

[41] G. Puebla, F. Bueno, and M. Hermenegildo. A Generic Preprocessor for Program Validation and Debugging. In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 63–107. Springer-Verlag, September 2000.

[42] G. Puebla, F. Bueno, and M. Hermenegildo. An Assertion Language for Constraint Logic Programs. In P. Deransart, M. Hermenegildo, and J. Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in LNCS, pages 23–61. Springer-Verlag, September 2000.

[43] G. Puebla, F. Bueno, and M. Hermenegildo. Combined Static and Dynamic Assertion-Based Debugging of Constraint Logic Programs. In *Logic-based Program Synthesis and Transformation (LOPSTR'99)*, number 1817 in LNCS, pages 273–292. Springer-Verlag, March 2000.

[44] G. Puebla and M. Hermenegildo. Abstract Multiple Specialization and its Application to Program Parallelization. *J. of Logic Programming. Special Issue on Synthesis, Transformation and Analysis of Logic Programs*, 41(2&3):279–316, November 1999.

[45] G. Puebla and M. Hermenegildo. Abstract Specialization and its Applications. In *ACM Partial Evaluation and Semantics based Program Manipulation (PEPM'03)*, pages 29–43. ACM Press, June 2003. Invited talk.