The Knowledge Engineering Review, Vol. 10:2, 1995, 205–208

Workshop on Integration of Declarative Paradigms

JUAN JOSÉ MORENO-NAVARRO

U. Politécnica, Facultad de Informática, Campus de Montegancedo, Boadilla del Monte, 28660 Madrid, Spain (e-mail: jjmoreno@fi.upm.es)

1 Introduction

The workshop in *Integration of Declarative Paradigms* was part of the post-conference workshops of the *International Conference on Logic Programming* held in Santa Margherita Ligure, Italy from June 14–18 1994.

The main topic of the workshop was to study recent advances in the integration of declarative programming languages, like functional and logic programming, as well as the integration of other paradigms, that are not strictly related with declarative languages, like object orientation, constraints and concurrency.

The integration of declarative paradigms and, in particular, the integration of functional and logic programming, was an active area of research some years ago. New models and languages were proposed, and it was shown that the new paradigm is feasible.

Recently, interest for this kind of languages has been spurred anew, especially for what concerns efficient implementation. Currently, there is active research in:

- Design of new programming languages accounting for a mature and up-to-date understanding of previously studied features meant to improve the expressive power of declarative languages (lazy evaluation, higher-order objects, types, object orientation, etc.).
- Development of efficient implementations. The proposals combine existing techniques from logic programming (e.g., graph reduction, strictness analysis, optimization of deterministic computations) as well as new techniques (e.g., combination of normalization and narrowing, residuation and coroutining, parallelization).
- Integration with other paradigms, like the inclusion of constraints, real-time programming, visual programming, etc.

The aim of this workshop was to bring together researchers from different communities as well as giving the logic programming audience an idea of recent advances in the area. So, the workshop tried to contribute to close the gap between the different communities and create a good opportunity for exciting discussions.

2 Overview of the workshop

The workshop was organized in four sessions, each with a different proposal for paradigm integration.

The first session was on *Logic Programming with Constraints*. The combination of constraint solving and logic programming is quite natural because both involve a search of values to variables under certain conditions, i.e., constraint solving generalises unification. The constraint logic programming paradigm has been very successful from the expressivity and the efficiency point of view. It has been also the basis for new concept integration, like object orientation. Rigotti et al. (LISI, Villeurbanne Cedex) have developed an integration of terminological constraints and Frame Logic (F-logic) programming. The approach supports two flavours of object-orientation: at

the term level and at the atom level. The paper "A Layer Architecture for the Integration of Rules, Inheritance, and Constraints", by Abecker and Wache (DFKI, Kaiserslautern), shows an integration of logic programming with terminological reasoning.

The most interesting paper of the session was "Semantics of Constraint Logic Programs with Optimization", by Marriot (Monash Univ., Melbourne) and Stuckey (Univ. Melbourne), a well-written work containing a precise definition of an optimization predicate for constraint logic programming. The work presents a clean declarative semantics for optimization in CLP languages; what is particularly interesting is if the corresponding operational semantics allows for an efficient implementation.

The second session was on *Extension of Type Systems for Declarative Languages*. It is widely accepted that type declarations and checking, while prevent program errors, are also useful to model the problem. However, it is not clear which is the limit of the expressivity of the type system so that it must be decidable and efficiently implemented. The paper "Logic Programming with Constructor-based Type Constraints", by Goltz (GMF-FIRST Berlin), proposes an ambitious type system based on constraints. A different approach appears in the work by Taver (Univ. Leeds), where a new functional language SEQUEL is presented based on sequent calculus.

The third session was on the *Implementation of Integrated Languages*. It was one of the most important sessions of the workshop, due to the significance and quality of the submissions. Chakravarty (TU Berlin) and Lock (INRIA Nancy) presented a detailed description of JUMP, a very efficient abstract machine designed for the implementation of a wide range of declarative multi-paradigm languages in a systematic way. It is composed of a minimal core and several orthogonal extensions. The core realises the basic operational abilities, such as a functional call mechanism, and provides the *glue* for integrating the extensions specific to a given paradigm, such as higher-order functions or logical variables.

Bert and Echahed (LGI-IMAG, Grenoble) gave the broad outlines of LPG, a programming language that integrates algebraic and logic programming paradigms. The paper shows how technical points of the integration have been tackled to define a denotational and operational semantics of the language. In particular, they discuss how to integrate disequations in the language.

Finally, Grivas (ETH Zürich) presented the language AlgBench, which integrates three paradigms in declarative programming: logic, functional and constraint-based programming, using a symbolic computation system as a general framework.

The last session was on *Declarative Programming plus Concurrency*, where we also found very interesting papers. Díaz *et al.*, (Univ. Málaga) proposed a Parlog extension with real-time characteristics. The implementation issues are not trivial, and they make a detailed description in the paper.

The paper "Compilation of Concurrent Declarative Systems", by Ariola et al. (Univ. Oregon), discusses interesting techniques for the implementation of concurrent declarative (functional and logic) languages. Non-strict functional languages and concurrent logic languages share a lot of common features; so the construction of a kernel language that benefits from just these features seems a good idea. They devote part of the work to study partitioning, a very interesting and difficult problem.

The last paper was "Distributed Declarative Systems as Parts of Cooperating Software Environments", by Czajkowski *et al.* (Univ. Krakow). It describes a software architecture to glue different distributed declarative programming languages together. It presents a reasonable solution to the problem of the inter-operability between different programming paradigms.

3 Panel discussion: is there any need for the integration of paradigms?

The panel discussion brought together several interesting panellists, one for each kind of integration proposed: Mario Rodríguez-Artalejo argued for the inclusion of types and higher order features in the integrated languages. Evan Tick defended the point of view of concurrency, which can be used instead of lazy evaluation. Manuel Chakravarty gave several arguments for the

integration of functional and logic programming. Peter Stuckey was responsible for emphasising the importance of constraint-based languages. Michael Hanus and J. J. Moreno-Navarro moderated the panel.

The panel tried to discuss three points:

- The need for the integration of paradigms. Every panellist answered the initial question (hopefully in a positive way) by explaining the importance of a concrete paradigm. The answers focused on:

  —Expressivity of new languages

  Higher order features enables user defined control structures, in particular control structures (like map, fold and filter) tailored to recursive data structures (like lists and trees). Concurrency allows for managing new and interesting problems, namely those related with reactive and real-time systems. The integration of logic and functional programming is the basis for a lot of new features: functions and nested calls, lazy evaluation (which improves modularity and provides new programming techniques like the use of potential infinite data structures), default rules (or constructive function completion) and the combination of explicit and implicit negation, something very useful from the knowledge representation point of view. The inclusion of constraints in the model allows for a uniform and efficient treatment of primitive operations on concrete domains, like arithmetic operations on real numbers or finite domains.

  Some additional comments on these positions were made: Mario Rodríguez explained the merits of higher order features, mainly the huge amount of potential reusability, but he also pointed out the risk of producing more obscure programs.

  Peter Stuckey considered constraints as the unique needed extension because everything can be covered by this model: concurrency by *Concurrent Constraint Logic Programming* and functions using constraints over an equational theory. Mario Rodríguez had some doubts about this statement, because it is not very easy to model laziness in a constraint framework. Evan Tick reminded the audience that concurrency could be used to model laziness. Manuel Chakravarty concluded the discussion by remarking that the important point is not the execution model but how natural is the programming language for the programmer to solve specific problems.

  —Application fields and Implementation:

  Everybody agreed on the usefulness of integrated languages for certain applications. In particular, they are especially well suited for rapid prototyping and applications to such fields as symbolic computation, artificial intelligence and knowledge-based systems. Languages based on logic programming are very useful for problems which involve search, while constraint logic languages are used in the presence of combinational problems.

  It is worth mentioning that the current implementations are highly competitive with Prolog implementations. Moreover, the efficiency of functional-logic and constraint programs is often better than their Prolog equivalents.

- *Availability of integrated languages*

  The question posed to the audience was the following:

  Assuming we need integrated languages, how can we convince people to use them?

  Most people agreed with the fact that our usual role as researchers, i.e. to publish conference and journal papers is not enough.

  Of course, the first step is to provide an efficient implementation. But, how this implementation is to be made available was a major point of discussion. A part of the audience argued that an academic (often toy) implementation does not help too much, and it is necessary to provide a commercial product. As an example, they mentioned the success of Prolog (supported by several efficient commercial products) in contrast to the lesser use of functional languages. On the other hand, some other people established that, by now, a general-purpose programming language, no matter how efficient or convenient, is not a viable commercial product. What it is

usually sold perhaps is a super-duper programming environment for a language that is used. They assume that the best way to kill a new language is to try to sell it. Instead, they prefer to give it away for free. It may, at best, create a community of users. Ironically, someone added: "and this only provided your language has some absolutely marvellous feature, runs at light speed, and does not require more than a high-school degree to use." After the discussion, the only agreement was on the idea that, in any case, we need implementations with the same quality as well known commercial products, even if they are distributed in a public domain basis.

The second step was the software engineering point of view. Good propaganda, and at the same time a good bench test for this kind of language, is needed for industrial application. Furthermore, the language needs to provide the usual features demanded from the "programming-in-the-large" discipline of programming, including modularity, data abstraction, reusability support and separate compilation.

- *Future integrations*
  The panellists claimed that there are still more opportunities for integration: modularity facilities and, especially, the integration of object oriented features, must play an important role in integration. Also, integration with constraints needs further development. From the implementation point of view, new efforts must be invested in the parallel implementation of integrated languages.

The proceedings are available in the WWW at http://www.mpi-sb.mpg.de/guide/staff/michael/michael.html or by anonymous ftp from ftp.mpi-sb.mpg.de (139.19.1.1) in the directory pub/guide/staff/michael/publications. A special issue of the *Journal Computer Languages on Extensions of Logic Programming* is in progress, and it will contain selected papers from the workshop. As a consequence of the panel discussion a new initiative was started: cooperation in the design and implementation of a (unique) functional logic language in a similar framework to the Haskell experience. This will avoid the duplication of work while giving a broader projection to our work. The language should be a basic compromise to compare different implementations, join applications, etc., and should not contain all possible features, but only the basic ones which seem to be essential for functional logic programming. It should be the basis of and open for future extensions. Further information can be obtained by sending a message to fll-request@judith.ls.fi.upm.es.