BEAUTY — THE LANGUAGE FOR PARALLEL PROGRAMMING

P. Přikryl

TU of Brno, Fac. of EE & CS, Dept. of Computer Science & Engineering
Božetěchova 2, 612 66 Brno

Key words: parallel programming, object oriented, high-level language

Beauty is a working name for the object-oriented language for parallel programming. It was inspired by the mechanism called *generative communication* and by the language Linda which was its first implementation [1]. However, Beauty is not only "Linda in an object-oriented coat". The author believes that Beauty makes a step ahead toward theoretical purity and practical usability.

The Linda language is based on abstraction of separated data entities (tuples) and processes. They are enclosed in the abstract environment called *tuple space* (`TS`). The processes can access tuples associatively via operations `in()` and `rd()`, and they can generate new tuples using the `out()` and `eval()` operations. The `eval()` operation is also launching new processes. Abstract features of `TS` can be implemented in a parallel environment thus making parallel programming in Linda independent on hardware and architecture. Interaction of otherwise independent processes based on persistent and associatively accessed data often makes parallel programming easier. One of the reasons is that programmer can focus on programs for processes one by one — he/she need not to keep in mind a mutual dependency of explicit interaction in the context of all processes. Highly intuitive operations also help the beginners to learn the problems of parallel programming.

Despite nice features of Linda, deeper observation disclosed also some weak points. For example, the exclusively associative access to shared tuples leads sometimes to problems while working with graph data structures. The problem of *direct access* to tuples was partially solved in [3]. The introduction of *multiple tuple spaces* in [2] and in other works shows another general problem — the problem of *protection* of some groups of tuples. A different approach to solve the problem was used in the PIN language [4]. PIN introduces new operations for separation of the directly-only accessed data space. PIN also introduces the notion of objects and object space instead of tuples and tuple space. Beauty is direct and "nicer" ancestor of PIN.

An object can encapsulate both data and algorithms. This simple observation has nontrivial consequences in a parallel environment. New questions are: how many threads can be encapsulated in one object; how objects communicate amongst themselves (they run in parallel); if the object is to be chosen associatively, how it can be done; how the objects can interact in a distributed environment; etc. Besides the syntax and semantics (static properties), we also have to take care of the mutual *behaviour* of objects (dynamic qualities).

Beauty defines two basic kinds of objects: *object space* (`OSp`) and *the other objects running in parallel* — let's call them *internal* objects. The object space is an object that replaces tuple space in Linda. Internal objects replace tuples and processes. One `OSp` is therefore superior to many internal objects. The internal objects communicate directly with their `OSp` that acts as a communication agent between objects on the same hierarchical level and as an interface to the "outer world" (i.e. to the `OSp`'s lower or higher in the hierarchy). As all interaction passes through, the `OSp` knows everything about activities and qualities of the subordinate objects. The `OSp` has also direct access to the internal objects, therefore it can also implement an efficient associative searching.

Beauty defines the following kinds of `OSp`'s methods: create & destroy, call, and lock & unlock. The methods *create* and *destroy* are self-explaining. The method *call* replaces all the operations of Linda. The methods *lock* and *unlock* allow implementation of customized atomic operations. The methods vary according to the access to internal objects.

There are three basic variations considered: an access to one object via its identification, an access to one object randomly chosen from the candidates found associatively, an access to multiple (all) objects found associatively. The associative searching is based on *class* of the object and *m-method* of the object. The m-method (match method) is a non-blocking method of the internal object that decides if the object is a candidate for the search or not. The `OSp` calls the m-method of all its internal objects of the given class with given arguments. This way the set of potential candidates is reduced to those that are "willing" to be candidates. If the *multiple* version of the operation is called then the `OSp.`*method*`()` of all candidates is called (or of one chosen candidate, respectively). Another methods—the *e-methods* (evaluation methods)—are called only for the chosen candidates. They may block the activity of the object until required conditions for further computing are satisfied. The communication between `OSp` and the internal processes is always synchronous. An asynchronous communication may be implemented using an inserted object.

The OO mechanism of inheritance allows to make the searching for objects tailored to an application domain — unlike in variations of Linda where the searching mechanism is hard-coded inside the system. The embedding of Beauty into the base language is easy as the `OSp` has the usual-object interface. The operations, similar to Linda's, can be implemented syntactically as usual methods of `OSp`. In Linda, the operations have to be preprocessed into the sequence of other operations in the base language. And also, from the view point of the application design, objects are less restrictive than tuples.

References:

[1] GELERNTER, D.: *Generative communication in Linda.* ACM Transactions on Programming Languages and Systems, 7(1):80–112, January 1985.

[2] GELERNTER, D.: *Multiple tuple spaces in Linda.* Proceedings of PARLE '89: Parallel Architectures and Languages Europe, Vol. II: Parallel Languages, Lecture Notes in Computer Science no. 366, pp. 20–27, Springer-Verlag, June 1989.

[3] PŘIKRYL, P.: *Extension of Linda and the Discrete Simulation.* Proceedings of 6th Moravo-Silesian International Symposium on Modelling and Simulation of Systems (MOSIS '94), pp. 188–195, Zábřeh na Moravě, Czech Rep., May 31 – June 2, 1994.

[4] PŘIKRYL, P.: *PIN — the language for parallel/distributed environment.* Proceedings of 30th Spring International Conference on Modelling and Systems Simulation, MOSIS '96, Vol.1, Krnov, Czech Republic, April 23–25, 1996.

This research has been conducted at the Dept. of Computer Science and Engineering at the Technical University of Brno as a part of PhD work of the author reading "Means for parallel programming".