Development of an Environment for Specification and Execution of Active Objects on Parallel Machines

Parallel Computing Action: Application No 4232

Jean-Pierre Briot, Philippe Gautron
Sylvie Lemarié, Loic Lescaudron, Hayssam Saleh

Joint Team RXF - LITP,
Université Pierre et Marie Curie,
4 place Jussieu, 75005 Paris, France
`[briot,gautron]@rxf.ibp.fr`

Report on the Project - October 1990

1 Summary of Objectives

The goal of our project is the design of an environment based on object-oriented programming to program parallel computers. We chose the object-oriented concurrent programming paradigm [OOCP 87] (based on active objects) as the foundation for expressing concurrent programs.

The system we propose includes two components:

- a specification and experimentation environment running on a workstation. This environment, named Actalk, extends the Smalltalk-80 system towards concurrent active objects (named *actors* [Agha 86]),

- an associated execution environment running on a parallel computer (a T-Node machine based on T800 transputers). This second component includes three sub-components:
  - the language, named GnuActalk, which is the Actalk system ported onto the GnuSmalltalk system (GnuSmalltalk is a C implementation of Smalltalk). There will be one GnuActalk system on each node, and they will be inter-connected and extended to support remote objects and communication.
  - a graphic interface software to configurate the distributed parallel computer.
  - a system layer to support basic object facilities, such as object localization, naming and migration. This layer is developped in C/C++ on top of Helios Operating System.

These two components are stand-alone, although they are complementary. Their combination will propose an integrated development and execution environment for concurrent object-oriented languages and programs. It will allow the programmer to design and experiment programs into a rich specification environment, and to run them onto the parallel machine through the support of the execution environment.

2 Motivations

Members of the team have large experience in specification and implementation of concurrent object-oriented programming languages and distributed systems. Like other researchers in the same field, we faced the almost non existence of appropriate tools to design such parallel programs and moreover experiment with them. Rather than focusing on a formal approach to design safe programs and reason about them, we advocate the needs for specific programming environments to help specification and to visualize experimentation.

We noted that there are now good execution systems for running programs on parallel computers, but their environment and development environments are usually very limited. There are also some specification and simulation environments for concurrent programs running on mono-processors, but they are limited to simulation and do not control true parallel machines.

We believe that realizing a system for both specification and execution on parallel computers is hard to complete at least in a reasonnable amount of time. Our proposal is to take the best of both components, namely simulation environment, and parallel execution system, and to design them to be fully compatible and interfaced. Programs specified in the development environment can be translated onto the execution environment to run them on the parallel machine. Although, as mentionned earlier, these two environments may be used independently, their combination will give the programmer a complete unified environment for specification, simulation and parallel execution of his programs.

3 Previous and Current Work

3.1 Development Environment: Actalk

The kernel and prototype environment of the first and higher-level component has already been partly designed and implemented as a stand-alone project. This system, named Actalk [Briot 89] (which stands for *act*ors in Small*talk* [Goldberg and Robson 83]), is an integrated environment for specification, classification and experiment with concurrent object-oriented languages, based on the notion of active objects and therefore also named actor-based languages.

We decided to base this laboratory for experiments on the object-oriented programming methodology in order to benefit its merits of uniformity, modularity, reusability, and easy interfacing. We chose the Smalltalk-80 programming language and environment as the unified foundation.

The design and implementation of the kernel of the platform has already been completed and it has been applied to simulate some of the most popular actor-based programming languages (the Act3 and ABCL/1 languages) [OOCP 87]. Actalk is being applied by another research team in order to describe various Distributed AI systems, by extending the concept of actor into the concept of *agent*. One such realization is the Mages multi-agent system [Bouron et al. 90].

The standard Smalltalk-80 programming environment has been extended in order to visualize and interactively monitor the activity of concurrent objects [Briot and Lescaudron 90]. We added some tools such as a generic preemptive scheduler, a scheduling monitor, and an automatic interface generator.

3.2 Parallel Execution Environment

This second component includes the execution system, based on the GnuSmalltalk system, the graphic configurator, and a system layer for object management.

3.2.1 Execution Environment: GnuSmalltalk

In order to get the best match with the Actalk environment, we chose the GnuSmalltalk system, because it is compatible with Smalltalk-80, implemented in C, and its sources are available. During the first part of the project we prospected another way, based on an extension of C++ towards actors, named Tact. A prototype of the Tact system has been specified and implemented, but we later decided to switch to the GnuSmalltalk system.

We started this new development by easily porting the Actalk kernel onto GnuSmalltalk. After a careful analysis of the GnuSmalltalk system and its implementation, we implemented a prototype of communication layer between several GnuSmalltalk systems on Sun workstations. We simulate the front-end / T-Node target architecture by having two kinds of communication: communication between GnuSmalltalk workers (simulated nodes) and communication with the main GnuSmalltalk (simulated front-end). The implementation relies on Unix sockets.

3.2.2 Graphical Configuration Tool

The network topology between transputers and the hardware resources must be specified by the user through a low level configuration file. We have designed and implemented a configurator to graphically specify this topology. This tool:

- automatically checks the validity of the configuration supplied by the user.

- optimizes the use of the hardware links.

After the processors have been connected, one needs to connect the different processes. Unfortunately, hardware links are limited on each processor. This leads the programmer to write another software layer to deal with multiplexing/demultiplexing of hardware channels.

To avoid such a burden for the programmer, we have designed and implemented a second tool to cope with:

- the placement of the processes on the network.
- the generation of multiplexors/demultiplexors when the processes channels cannot be mapped on the hardware links.

3.2.3 System Layer

C++ is our kernel language for system programming on Helios. We chose C++ because of its availability and because it is an efficient programming language well-suited to system implementation.

Our previous experience to implement disributed systems[Shapiro et al. 89] leads us to design a minimal system layer controlling resource management and system servers on top of this basic layer. System servers include objet shifting (migration with import/export semantics), object localization and object naming. We benefit from the support of C++ to implement generic facilities. Parameterized types and inheritance are the two basic techniques we use to implement system interface.

We have ported different softwares essential to our development on Helios:

- free software, like the GNU C pre-processor or the GNU make.
- C++ version AT&T-2.0. This implementation of C++ translates C++ code in C code. We use the Helios C compiler but the port of C++ on Helios required some modifications to the C++ compiler. Port of version 2.1 is planned as soon as it is available.
- cross-compiler (more precisely cross-translation). Our developments are first tested on Sun workstations under the usual Sun environment, with a Sun version of the Helios C++ compiler. The source codes are then recompiled on Helios.

The current prototype is in part implemented - we are able to remotely allocate objects and to move them -, and in part under specifications. Our design concerns a stub generator including remote pointers management and remote procedure calls. We rely on Helios both to install our system layer and to support the communication layer. This prototype aims at learning Helios programming and at evaluating the implementation of system facilities.

4 Work Planned

Most of the development of the Actalk simulation environment has already been achieved. The remaining work focuses on the programming environment aspects (visualization, control, debugging). This also includes the design and implementation of a higher level language level with bidirectional communication between actors, where the user does not have to take care of explicit continuations. A prototype compiler between such a higher layer and Actalk has already been implemented.

We will start porting GnuSmalltalk onto the transputers. Then we will change the current communication layer prototyped with sockets between work stations into the use of T-Node channels.

We plan to add a graphical interface to our channel multiplexor/demultiplexor.

We plan to design a high level execution layer for actors, including implicit allocation, migration, garbage collection. This will be designed on top of the basic execution layer which gives explicit control on migration. The GnuActalk will then be interfaced with this layer to provide a high level control on allocation/migration strategies.

The basis system layer will provide a library of C++ classes to support object facilities such as object migration and persistence. The new enhanced extension is planned as a pre-processor for the new version 2.1 of the C++ compiler.

About the system layer, future work is threefold:

- porting of environment tools.
- validate our prototype with GnuSmalltalk: our system support is then a system library.
- generalize our concept of object to active objects.

5 Deliverables

All public sources are deliverables (including Gnu porting). Public source is free. Sources based on licensed products needs appropriate licensing (including the AT&T-C++ porting).

The Actalk environment is portable in Smalltalk-80 2.4.

GnuSmalltalk is planned to run onto T-Node. The minimal object system layer and library tools object migration will be in C++ 2.0. The configurator is running in C++/X.

References

[Agha 86] G. Agha, Actors: a Model of Concurrent Computation in Distributed Systems, *Series in Artificial Intelligence*, MIT Press, 1986.

[Bouron et al. 90] T. Bouron, J. Ferber and F. Samuel, A Multiagent Testbed for Heterogeneous Agents, *2nd European Workshop on Modelizing Autonomous Agents and Multi-Agent Worlds (MAAMAW'90)*, published as a Laforia Research Report, No 20/90, July 1990.

[Briot 89] J.-P. Briot, Actalk: a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment, *European Conference on Object-Oriented Programming (ECOOP'89)*, *British Computer Society Workshop Series*, Cambridge University Press, pages 109-129, 1989.

[Briot and Lescaudron 90] J.-P Briot and L. Lescaudron, Building Unified Programming Environment for Object-Oriented Languages, *the Fifth International Symposium on Computer and Information Sciences (ISCIS V)*, also published as RXF-LITP Research Report, No 90-76, October 1990.

[Goldberg and Robson 83] A. Goldberg and D. Robson, Smalltalk-80: the Language and its Implementation, *Series in Computer Science*, Addison Wesley, 1983.

[OOCP 87] Object-Oriented Concurrent Programming, edited by A. Yonezawa and M. Tokoro, *Computer Systems Series*, MIT Press, 1987.

[Saleh and Gautron 90] H. Saleh and P. Gautron, Language Implementation Feedback on Design, *Technology of Object-Oriented Languages and Systems (TOOLS 2)*, editors: J. Bézivin, B. Meyer and J.-M. Nerson, Angkor (publisher), Paris, June 1990.

[Shapiro et al. 89] M. Shapiro, P. Gautron and L. Mosseiri, Persistence and Migration for C++ Objects, *European Conference on Object-Oriented Programming (ECOOP'89)*, BCS Workshop Series, Cambridge University Press, pages 191-204, July 1989.