Development of an Environment for Specification and Execution of Active Objects on Parallel Machines PCA 4232

Jean-Pierre Briot, Philippe Gautron
Sylvie Lemarié, Loic Lescaudron, Hayssam Saleh
Rank Xerox France & LITP, Institut Blaise Pascal
Université Paris VI
4 place Jussieu, 75252 PARIS CEDEX 05, France
[briot,gautron]@rxf.ibp.fr

March 1991

Abstract

The goal of the project is the design and the implementation of an environment based on object-oriented techniques to program parallel applications.

The environment includes two components:

- a specification and experimentation environment running on a standard workstation. This environment, named Actalk, extends the Smalltalk-80 system towards concurrent active objects.
- an associated execution environment running on a parallel computer.

The system layer is developped in C/C++on top of the Helios operating system and addresses different issues:

- a system library to support basic object facilities, such as localization, remote and concurrent access, universal naming, and migration.
- a graphic interface software to configurate a parallel computer.
- the porting of GnuSmalltalk (a C implementation of Smalltalk-80) on a parallel computer.

In order to validate our approach, we plan to port Actalk on GnuSmalltalk [Lemarié 91]. There will be one GnuAltalk system on each node of the parallel machine and communications will rely on our system library.
This paper reports the current status of our project: deliverable and current work.

Current configuration

T-Node with 16 transputers 4M RAM, 2 hosts (Telmat STE-30 and Sun4), Helios Operating System.

Working Environment (systems and languages)
Unix, Helios, X-Windows, C, C++, Smalltalk-80.

1 Deliverable: System Configuration

Any application designed to support concurrent activities programming appears as a collection of communicating processes. But the connexions between processes is limited to the number of physical links on each processor. The graphical tool we have implemented allow users to cope with the following issues:

- configuration file generation with optional outputs: OCCAM, Helios.
- interactive checking of connexions (two OCCAM channels connected on the same link for instance).Once the user validates his interface, a correct configuration file is immediately generated.
- logical communication links. An application may need more processes than the number of available processors. In this case, several processes will be run on a same processor and remote communications will occur through logical links of this processor. Multiplexors and demultiplexors are wittingly introduced to manage messages between logical links in order to:
    1. make inter-processor communications as transparent as intra-processors communications are.
    2. avoid virtual deadlocks. Suicidary processes acts as buffers between the (de)multiplexor and the user process.
- Evaluation of performance on different network topologies. A same application can be performed on different topologies with the communication layer generated by the graphic tool.

The tool is written in C++, on top of X-Windows (X11-R4) and uses the `Xt` library of X11. A demonstration of this public domain software is planned for the Bonn meeting.

2 Current Work

2.1 Development Environment

Actalk is an extension to Smalltalk-80 introducing active objects and asynchronous message passing in a standard sequential environment. The graphic interface has been extended to provide visualization of active objects and the standard scheduler has been modified to support asynchronous events.

The current version of Actalk is available (domain public) for Smalltalk-80 version 2.5.

Futur work includes to implement a communication protocol between Actalk based on the host machine and different nodes of the underlying parallel machine running GnuSmalltalk.

2.2 Parallel Execution Environment

The current development of our execution environment is twofold:

- an extension to C++ to support concurrency and distribution. More precisely, we are designing a concurrency control machanism for C++ objects.
- a stub generator, including:
  1. an object finder. This object finder localizes objects through the network.
  2. a support for remote procedure calls. The stub generator packs and unpacks the arguments of a remote procedure call. In accordance with the object approach, any procedure call is applied on an object. A remote procedure call looks like a standard procedure call. Since the system can localize this object, it can generate either a remote or a local procedure call.
  3. asynchronous and synchronous procedure calls. Again, an asynchronous call looks like a synchronous call. But the declaration of the object on which this call is applied then differs.

Our system layer is based on C++: extension (concurrency), parsing (stub generator), library. Our library relies on C++techniques such parameterized types[1] to provide templates for asynchronous calls declarations.
A demonstration of the current status of the library is planned for the Bonn meeting.

References

[Lemarié 91] Sylvie Lemarié. Simulation d'une architecture multiprocesseur à mémoire répartie. Application à un système acteur réparti. In *AFUU 91 proceedings.* Paris, March 91.

[Saleh and Gautron 91] Hayssam Saleh and Philippe Gautron. A Concurrency Control Mecanism for C++ Objects. *Technical Report LITP 91-04 RXF , LITP, Université Paris VI - PARIS.*

[Stroustrup 88] Bjarne Stroustrup. Parameterized types for C++. In *Proc. C++ Conference. Also Published in Journal of Object-Oriented Programming, vol. 1, No 5, Jan 1989*, pages 1–18, Berkeley, CA (USA), October 1988. USENIX.

[1]Will be part of the next official release of C++ [Stroustrup 88]. Our domain public simulation is available.