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

Parallel Computing Action: Application No 4232

RXF-LITP Research Team

RESEARCH PROPOSAL

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 concurrent oriented programming paradigm (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 top of the Smalltalk-80 system on a workstation, named Actalk,
- an execution environment running on top of the C++ environment on a parallel computer (a T-Node machine based on T800 transputers), named Tact.

These two components will be stand-alone, although they will be developed together in order to get a perfect combination and interface between them. The complete system Actalk/Tact will provide 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 transparently run them onto the parallel machine through an automatic translation to the execution environment.

2 Motivations, Previous and Current Work

2.1 Motivations

Members of the teams 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 environment 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. We rather chose to take the best of both components, namely specification and simulation environment, and parallel execution system, and to design them to be fully compatible and interfaced. Programs specified in the development environment will be transparently 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.

2.2 Previous and Current Work

2.2.1 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 (which stands for *act*ors in Small*talk*), 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) and is currently used as a tool for designing new ones (the Mering-IV programming language for distributed artificial intelligence which will be implemented on hypercube machine). Actalk is modular and flexible enough to allow the design of various concurrent object-oriented programming languages and the corresponding programs rather than being stuck to some specific language.

A sub-project of Actalk whose first prototype is now just completed currently designs some programming environment customized for actors. It is implemented as an extension (subclasses) of the standard Smalltalk-80 programming environment, and is able to visualize and control active and concurrent objects.

We think that, because our specification platform has already been validated, it is now time to start implementing its counterpart on the parallel machine.

2.2.2 Parallel Execution Layer: Tact

We chose C++ as the kernel language for programming on the transputers. C++ is the mostly used language for current distributed operating systems research projects, and we already have such experience. C++ will be ported on top of C compilers on transputers.

Therefore we started prospecting how to express the kernel semantics of

Actalk actors into C++. We call this layer Tact. We experienced interfacing Smalltalk-80 with C++ by adding C++ primitives to the Smalltalk-80 virtual machine. Our previous experience of using C++ to implement distributed systems lead us to design a minimal object system layer controlling execution and resource management (object allocation and migration) onto a distributed environment.

3 Work to be Done

3.1 Work Planned

We intend to develop concurrently both two components: the specification environment Actalk on Smalltalk-80, and the execution environment Tact on transputers. By developping them concurrently we expect an optimal symbiosis between them. The interface between Actalk and Tact will include a translator which transforms Actalk programs into their equivalent form in Tact. The resulting Tact programs may then be compiled into executable code on the transputers.

The resulting development chain includes four layers:

- Actalk on top of Smalltalk,
- Smalltalk interfaced with Tact,
- Tact interfaced with the C compiling chain,
- native transputer code resulting from C compiler.

3.2 Schedule

The project will be divided in 3 phases:

first phase: *(1 year)* **prototyping** Design of a first prototype of the Tact system on a workstation. Implementation of the C++ basic execution layer on the transputers. Design and implementation of a first prototype of the Actalk− >Tact translator.

The development of the Actalk system, focused on the programming environment aspects (visualization, control, debugging) will go on during

all the project. This also includes the design and implementation of a higher level language level with bidirectional communication between active objects, 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.

second phase: *(6 months)* **validation** We plan some experiments with several application fields.

third phase: *(1 year)* `implementation` The feedback of the previous validation will lead to enhance the first prototypes to achieve a complete system. A high level execution layer, including implicit allocation, migration, garbage collection will be designed on top of the basic execution layer and connected to the Tact system.

Three PhD students will work on this project, under our direction. One will work on the Actalk platform, another one on the Tact extension, and the last one on the basic execution layer on the T-Node.

4 Description

4.1 First Phase: Prototyping

- Design and implementation of a concurrent extension of C++ towards active objects, named Tact. The activity of active objects will rely on C++ tasks. This first prototype will be designed on a workstation.

  6 man/months.

- Design and implementation of a prototype translator plus interface from Actalk to Tact. Because the kernel of Actalk is minimal and modular, it could easily evolve in order to keep a perfect match between Actalk and Tact during the project development process.

  6 man/months.

- Design and implementation of a minimal object system layer to allocate and execute C++ tasks on the T-Node transputer processors.

  6 man/months.

- Extension of the C++ compiler in order to allow object migration and persistence. We already implemented a similar extension for the purpose of SOS, a distributed and object-oriented system (Esprit-1 project 367 SOMIW). The new enhanced extension is planned for the new version 2.0 of the C++ compiler.

  6 man/months.

- Further development of the visualization and control environment for Actalk execution whose prototype is already implemented. This includes the study of debugging facilities.

  12 man/months.

4.2 Second Phase: Validation

- We will test the various parts of our system with several application fields like: multi-agents reasonning systems (such a system has already been designed on top of Actalk), games, distributed simulation, multi-voice music simulation (we intend to reuse some musical application for jazz improvisation that we already developped at IRCAM in C++ multi-task system but on a mono processor.) ...

  12 man/months.

4.3 Third Phase: Implementation

- Design and implementation of a high level execution layer for active objects, including implicit allocation, migration, garbage collection. This will be designed on top of the basic execution layer which gives explicit control on migration. The Tact prototype will be adapted to run on top of this distributed execution layer.

  12 man/months.

- Connection of the Actalk environment with this distribution layer to provide a high level control on allocation/migration strategies.

  6 man/months.

5 Results Planned

The global result of the project will be a unified platform with two components: a high-level specification environment on workstation: Actalk, and a minimal and flexible execution environment on transputers: Tact. These two components are automous and may be used independently. But they are complementary and combine together into the system Actalk/Tact through the interface and translator.

6 Deliverables

All public sources are deliverables. Public source is free. Sources based on licensed products needs appropriate licensing.

The Actalk environment is portable in Smalltalk-80. Tact will include an extended C++ compiler, a minimal object system layer, and library tools for C++ object migration.

7 Dissemination

We intend to publish our ongoing work in related conferences and working groups. We will discuss the progress and results with other teams through the special workshops organized within the program.

8 Using Results

We will be the first users of our system to prospect and experiment both with concurrent programming methodology, evaluate algorithms for resource allocation, and study relation between object migration and routing topology. We will introduce our results and experience into some pedagogic courses at University. As already pointed, we have many application fields in mind.