INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team mimosa

Migration et Mobilité: Sémantique et Applications

Sophia Antipolis

THEME 1C

Activity Report

2003

Table of contents

1. Team

MIMOSA *is a joint project of INRIA, the Centre for Applied Mathematics (CMA) of the Ecole des Mines de Paris, and the Laboratoire d'Informatique Fondamentale of CNRS and the Universities of Provence and Méditerranée.*

Head of project-team
Gérard Boudol [Research Director, Inria]

Vice-head of project team
Ilaria Castellani [Research Scientist, Inria]

Administrative assistant
Sophie Honnorat [Inria]
Dominique Micollier [Armines]

Staff members Inria
Manuel Serrano [Research Scientist, Inria]

Staff members CMA and CMI
Roberto Amadio [Professor, University of Provence]
Frédéric Boussinot [Research Director, CMA]
Silvano Dal-Zilio [Research Scientist, CNRS]

Ph. D. students
Raul Acosta [CMA, till October 17]
Christian Brunette [MENRT]
Damien Ciabrini [MENRT]
Frederic Dabrowski [MENRT, from October 1]
Ana Matos [Portuguese Gov.]
Charles Meyssonnier [ENS Lyon]
Vincent Vanackère [ENS Lyon]
Pascal Zimmer [ENS Lyon]

Post-doctoral fellow
Xudong Guan [Inria]

Software development staff
Yannis Bres [Inria, till November 30]

2. Overall Objectives

The MIMOSA project is a joint project with the Centre for Applied Mathematics of the *École Nationale Supérieure des Mines de Paris*, and the Laboratoire d'Informatique Fondamentale of CNRS and the University of Provence and Méditerranée. The overall objective of the project is to design and study models of distributed and mobile programming, to derive programming primitives from these models, and to develop methods and techniques for formal reasoning and verification, focusing on issues raised by the mobile code. More specifically, we develop a reactive approach, where concurrent components of a system react to broadcast events. We have implemented this approach in various programming languages, and we aim at integrating migration primitives in this reactive approach. Our main research areas are currently the following:

- Models of mobility. Here we study constructs for the migration of processes, especially in models based on the π-calculus and its distributed variants, and on the calculus of Mobile Ambients.
- Security. We develop methods and tools for the verification of cryptographic protocols, and we investigate some security issues related to the migration of code (static verification of non-interference of code with security policies, static restriction of the computational complexity of code).

- Models and languages for reactive programming. We develop several implementations of the reactive approach, in various languages. We have designed, and still develop, an alternative to standard thread systems, called FAIRTHREADS. We intend to integrate constructs for mobile code in the model of reactive programming.
- Functional languages. We develop several implementations of functional languages, mainly based on the SCHEME programming language. Our studies focus on designing and implementing a platform for a *distributed environment*. The FAIRTHREADS, which have been added to BIGLOO, our SCHEME implementation, are at the heart of our client/server architectures. SKRIBE, a functional language for authoring documents, is designed to be used by servers to satisfy client requests.

3. Scientific Foundations

3.1. Semantics of mobility and security

Mobility has become an important feature of computing systems and networks, and particularly of distributed systems. Our project is more specifically concerned with the notion of mobile code which is a logical rather than physical notion of mobility. A main task in this area is to understand the various constructs recently proposed to support this style of programming, and to design a corresponding programming model with a precise (that is, formal) semantics.

The models that we have investigated in the recent past are mainly the π-calculus of Milner and the Mobile Ambients calculus of Cardelli and Gordon. The first one is similar to the λ-calculus, which is recognized as a canonical model for sequential and functional computations. The π-calculus is a model for concurrent activity, and also, to some extent, a model of mobility: π-calculus processes exchange names of communication channels, thus allowing the communication topology to evolve dynamically. The π-calculus contains, up to continuation passing style transforms, the λ-calculus, and this fact establishes its universal computing power.

The Mobile Ambient model focuses on the migration concept. It is based on a very general notion of a domain – an Ambient –, in which computations take place. Domains are hierarchically organized, but the nesting of domains inside each other evolves dynamically. Indeed, the computational primitives consist in moving domains inside or outside other domains, and in dissolving domain boundaries. Although from a computational point of view, this model may look quite simple and limited, it has been shown to be Turing complete.

In the past we have studied type systems and reasoning techniques for these models. We are now using them to design a general framework for the notion of migration. We are also studying how to integrate the model of reactive programming, described below, into a "global computing" perspective. This model looks indeed appropriate for a global computing context, since it provides a notion of reaction and time-out, allowing a program to deal with the various kinds of failures (delays, disconnections, etc.) that arise in a global network.

Finally, we are using formal models and techniques to address security issues: we use models derived from the π-calculus for the formalization and verification of cryptographic protocols, we use type systems to statically ensure the properties of integrity and confidentiality of data manipulated by concurrent programs. We also intend to use static analysis techniques to ensure that the mobile code does not use computational resources beyond fixed limits.

3.2. Reactive and Functional programming

Reactive programming deals with systems of concurrent processes sharing a notion of time, or more precisely a notion of instant. At a given instant, the components of a reactive system have a consistent view of the events that have been, or have not been emitted at this instant. Reactive programming, which evolves from synchronous programming à la ESTEREL, provides means to react – for instance by launching or aborting some computation – to the presence or absence of events. This style of programming has a mathematical semantics, which provides a guide-line for the implementation, and allows one to clearly understand and reason about programs.

We have developed several implementations of reactive programming, integrating it into various programming languages. The first instance of these implementations was Reactive-C, which was the basis for several developments (networks of reactive processes, reactive objects), described in the book [5]. Then we developed the SUGARCUBES, which allow one to program with a reactive style in JAVA, see [4]. Reactive programming offers an alternative to standard thread programming, as (partly) offered by JAVA, for instance. Classical thread programming suffers from many drawbacks, which are largely due to a complicated semantics, which is most often implementation-dependent. We have designed, following the reactive approach, an alternative style for thread programming, called FAIRTHREADS, which relies on a cooperative semantics. Again, FAIRTHREADS has been integrated in various languages, and most notably into SCHEME via the BIGLOO compiler that we develop. One of our major objectives is to integrate the reactive programming style in functional languages, and more specifically SCHEME, and to further extend the resulting language to support migration primitives. This is a natural choice, since functional languages have a mathematical semantics, which is well suited to support formal technical developments (static analysis, type systems, formal reasoning).
We also designed a tool to graphically program in the reactive style, called ICOBJS. Programming in this case means to graphically combine predefined behaviours, represented by icons and to implement reactive code. Potential applications are in simulation, human-machine interfaces and games.

4. Application Domains

4.1. Simulation

Simulation of physical entities is used in many distinct areas, ranging from surgery training to games. The standard approach consists in discretization of time, followed by the integration using a stepwise method (e.g. Runge-Kutta algorithms). The use of threads to simulate separate and independent objects of the real world appears quite natural when the focus is put on object behaviours and interactions between them. However, using threads in this context is not so easy: for example, complex interactions between objects may demand complex thread synchronizations, and the number of components to simulate may exceed the number of available threads. Our approach based on FAIRTHREADS, or on the use of reactive instructions, can be helpful in several aspects:

- Simulation of large numbers of components is possible using automata. Automata do not need thread stacks, and the consumption of memory can thus stay low.
- Interactions are expressed by means of broadcast events, and can thus be dealt with in a highly modular way.
- Instants provide a common discrete time that can be used by the simulation.
- Interacting components can be naturally grouped into synchronized areas. This can be exploited in a multiprocessing context.

4.2. Embedded systems

Embedded systems with limited resources are a domain in which reactive programming can be useful. Indeed, reactive programming makes concurrent programming available in this context, even in the absence of a library of threads (as for example the `pthreads`). One objective is to build embedded systems from basic software components implementing the minimal functionalities of an operating system. In such an approach, the processor and the scheduler are considered as special resources. An essential component is a new specialized scheduler that should provide reactive engines with the functionalities they need.
This approach is useful for mobile telecom infrastructures. It could also be used in more applicative domains, as the one of gaming consoles. PDAs are also a target in which the proposed approach could be used. In this context, graphical approaches as ICOBJS could be considered to allow end-users to build some part of their applications.

4.3. Web servers and Web proxies

FAIRTHREADS, as created by the MIMOSA team enables simple and efficient implementations of client/server applications. Hence, one of the most obvious application fields for FAIRTHREADS is the implementation of Web servers. It has not been demonstrated yet, that FAIRTHREADS can be implemented efficiently. Therefore, in the current state of our knowledge, it would be premature to state that FAIRTHREADS can be successfully deployed in high-speed servers such as Apache or Zeus.

Currently, we are considering using FAIRTHREADS for a Web server where performance is not dramatically demanded: *user-land proxies*. These should be uncluttering Web proxies, spawned by users. They should be easy to start, easy to stop, and highly customizable and *scriptable*. These Web proxies could be used to access all kinds of local textual information. For instance, standard LINUX distributions contain numerous documentations written in different formats (Docbook, man pages, ascii documentations, HTML, PDF, etc.). It is always puzzling to try to remember where these files are stored and how to visualize them conveniently. Our proxy could help with that task. It could be configured by users to extend the special syntax used by the Web browser to serve the local requests. For instance, the proxy could be configured so that HTTP requests starting with `http://doc:` are intercepted and handled locally by a program exploring the locations known to contain documentations. When the requested document is localized, the same program could select the appropriate translator or plug-in in order to visualize it in the browser. Each user could use a different configuration of the proxy.

5. Software

5.1. Mimosa Softwares

Most MIMOSA softwares, even the older stable ones that are not described in the following sections are freely available on the Web. In particular, some are available directly from the INRIA Web site: http://www.inria.fr/valorisation/logiciels/langages.fr.html. Most other softwares can be downloaded from the MIMOSA Web site: http://www-sop.inria.fr/mimosa.

5.2. Reactive Programming

Participants: Raul Acosta, Frédéric Boussinot, Christian Brunette.

5.2.1. Reactive-C

The basic idea of Reactive-C is to propose a programming style close to C, in which program behaviours are defined in terms of reactions to activations. Reactive-C programs can react differently when activated for the first time, for the second time, and so on. Thus a new dimension appears for the programmer: the logical time induced by the sequence of activations, each pair of activation/reaction defining one instant. Actually, Reactive-C rapidly turned out to be a kind of *reactive assembly language* that could be used to implement higher level formalisms based on the notion of instant.

5.2.2. SugarCubes

SUGARCUBES is a set of JAVA classes that implements the reactive approach in JAVA. JUNIOR is a kernel model issued from SUGARCUBES which basically defines concurrent reactive instructions communicating via broadcast events. SUGARCUBES and JUNIOR do not implement immediate reactions to absence, and this is one of the major differences with synchronous formalisms. SUGARCUBES and JUNIOR have efficient implementations able to deal with large numbers of parallel components and events.

5.2.3. Icobjs

ICOBJS programming is a simple and fully graphical programming method, using powerful means to combine behaviours. This style of programming is based on the notion of an *icobj* which has a behavioural aspect (object part), and a graphical aspect (icon part), and which can be animated on the screen. ICOBJS programming evolves from the reactive approach and provides parallelism, broadcast event communication and migration through the network. A new ICOBJS system is currently under development in Java. It unifies icobjs and workspaces in which icobjs are created, and uses a specialized reactive engine. Simulations in physics and the mobile Ambient calculus have been ported to this new system.

5.2.4. Rejo-Ros

REJO is an extension of JAVA that defines reactive objects having their own data and set of reactive instructions. These objects might also be considered as mobile agents which can migrate using a platform, named ROS, based on JAVARMI. REJO and ROS are built over JUNIOR as a set of JAVA classes developed in the team. The programming model of REJO is close to that of JUNIOR and SUGARCUBES and produces reactive instructions that are executed in a reactive machine. REJO and ROS are described in detail in the thesis of Raul Acosta-Bermejo [10].

5.2.5. FairThreads in Java and C

FAIRTHREADS is implemented in JAVA and usable through an API. The implementation is based on standard JAVA threads, but it is independent of the actual JVM and OS, and is thus fully portable. There exists a way to embed non-cooperative code in FAIRTHREADS through the notion of a fair process. FAIRTHREADS in C introduces the notion of unlinked threads, which are executed in a preemptive way by the OS. The implementation in C is based on the pthreads library. Several fair schedulers, executed by distinct pthreads, can be used simultaneously in the same program. Using several schedulers and unlinked threads, programmers can take advantage of multiprocessor machines (basically, SMP architectures).

5.3. The Bigloo compiler

Participants: Yannis Bres, Damien Ciabrini, Bernard Serpette [Project Oasis], Manuel Serrano.

The programming environment for the Bigloo compiler [9] is available on the INRIA Web site at the following URL: http://www-sop.inria.fr/mimosa/fp/Bigloo. The distribution contains an optimizing compiler that delivers native code, JVM bytecode, and .NET CLR bytecode. It contains a debugger, a profiler, and various Bigloo development tools. The distribution also contains several user libraries that enable the implementation of realistic applications.

BIGLOO was initially designed for implementing compact stand-alone applications under Unix. The BIGLOO JVM back-end has enabled a new set of applications: Web services, Web browser plug-ins, cross platform development, etc. More recently we have started an integration on the Microsoft Windows platform. In our mind, this is not a plain port (that is yet operational by means of the `cygwin` library), but an integration. In particular, we are currently investigating if it is feasible to make BIGLOO accessible from traditional Windows integrated development environments (IDE).

We have distributed two major releases of Bigloo during 2003: versions 2.6a and 2.6b.

5.4. Skribe

Participants: Erick Gallesio [University of Nice Sophia-Antipolis], Manuel Serrano.

SKRIBE [32] is a functional programming language designed for authoring documents, such as Web pages or technical reports. It is built on top of the SCHEME programming language. Its concrete syntax is simple and looks familiar to anyone used to markup languages. Authoring a document with SKRIBE is as simple as with HTML or LaTeX. It is even possible to use it without noticing that it is a programming language because of the conciseness of its original syntax: the ratio *markup/text* is smaller than with the other markup systems we have tested.

Executing a SKRIBE program with a SKRIBE evaluator produces a target document. It can be HTML files for Web browsers, a LaTeX file for high-quality printed documents, or a set of *info* pages for on-line documentation. Building purely static texts, that is texts avoiding any kind of computation, is generally not sufficient for elaborated documents. Frequently one needs to automatically produce parts of the text. This ranges from very simple operations such as inserting the date of the document's last update or the number of its last revision, to operations that work on the document itself. For instance, one may wish to embed inside a text some statistics about the document, such as the number of words, paragraphs or sections it contains. SKRIBE is highly suitable for these computations. A program is made of *static texts* (that is, *constants* in the programming jargon) and various functions that dynamically compute (when the SKRIBE program runs) new texts. These functions are defined in the SCHEME programming language. The SKRIBE syntax enables a smooth harmony between the static and dynamic components of a program.

SKRIBE is the continuation of the project formerly known as SCRIBE. SKRIBE can be downloaded at http://www-sop.inria.fr/mimosa/fp/Skribe.

SKRIBE is used by the MIMOSA project for authoring its Web page and... this document. Hence, we do not depend on any external tools for providing a LaTeX and a XML version of our activity report.

5.5. Verification of cryptographic protocols

Participant: Vincent Vanackère.

The TRUST tool, designed for the verification of cryptographic protocols, is an optimized OCAML implementation of the algorithm designed and proved by Amadio, Lugiez and Vanackère (see [13]). It is available via the url http://www.cmi.univ-mrs.fr/~vvanacke/trust.html.

6. New Results

6.1. Semantics of mobility

Participants: Gérard Boudol, Silvano Dal-Zilio, Xudong Guan, Ana Matos.

6.1.1. Work on the π-calculus

The work of Amadio, Boudol and Lhoussaine on the receptive distributed π-calculus has been published in two journal papers, [11] and [12].

We proposed ([25]) an extended, distributed π-calculus, taking advantage of the nested structure of ambients. While sticking to an asynchronous π-calculus core, channels in this calculus are nested and the nesting structure is explicitly maintained. The novelties of the calculus are the unification of channels and locations, and the transparency of addressing remote resources in mobile code.

In another the paper ([26]), we define a lexically scoped, asynchronous and distributed π-calculus, with local communication and process migration. This calculus adopts the network-awareness principle for distributed programming and follows a simple model of distribution for mobile calculi: a lexical scope discipline combines static scoping with dynamic linking, associating channels to a fixed site throughout computation. This discipline handles both remote invocation and process migration. A simple type system is provided, which is a straightforward extension of that of the π-calculus, adapted to take into account the lexical scope of channels. An equivalence law captures the essence of this model: a process behavior depends on the channels it uses, not on where it runs.

6.1.2. Work on the Ambient calculus, and related models

The work by Pascal Zimmer on the expressiveness of the model of pure Ambients has appeared in a journal [18]. The paper by Pascal Zimmer, David Teller and Daniel Hirschkoff on controlling resources in the Ambient model (presented at CONCUR'02) has been published in a journal [17].

In a deliverable of the MIKADO project [29], we have introduced a migration calculus, dealing with the problem of controlling the movement of computing entities with respect to the domains in which they run. More specifically, the main idea of our migration calculus is to provide the "membrane" of a domain with some computing power, so that we can program the way in which computing entities are accepted to enter the domain, or the way in which they are allowed to leave from it. This is a preliminary proposal. We illustrate, by means of examples, the expressive power of this model, but the deliverable does not contain any technical result.

The work by Silvano Dal-Zilio and Denis Lugiez on a tree logic based on the Ambient logic and extending XML Schemas, described in last year's report, has been published in the proceedings of the Conference on Rewriting Techniques and Applications [23]. This work received a "best paper award" at this conference. The results obtained in this work have been applied to decision problems in Ambient logic. Ambient logic is a modal logic used to describe the structural and behavioural properties of mobile Ambients. In the paper [24], we only consider the spatial fragment of the logic and work with finite, static processes. This static fragment, also called *tree logic* (TL), is essentially a logic on finite edge-labeled trees. The study of TL is motivated by a connection with type systems and query languages for semistructured data. In this setting, we are interested by two problems: *model-checking*, to test whether a given information tree satisfies a formula; and *satisfiability*, to test if there exists a tree that satisfies a formula. Given the similarity between TL and query languages, model-checking turns out to be similar to computing the result of a query, while satisfiability is useful for query optimizations or to check query inclusion. We prove the decidability of the model-checking and satisfiability problems for TL, as well as the decidability for the logic enriched with a limited form of fixed points on the vertical structure of a tree, akin to path expressions, and show that these two kinds of recursion are indeed orthogonal. The essence of our decidability results is a surprising connection between formulas of the ambient logic and counting constraints on (nested) vectors of integers.

6.2. Security

Participants: Roberto Amadio, Gérard Boudol, Ilaria Castellani, Ana Matos, Vincent Vanackère.

6.2.1. Cryptographic protocols

The work by Amadio, Lugiez and Vanackère on the symbolic reduction of processes with cryptographic functions appeared in a journal [13].

In a paper accepted for a conference affiliated with POPL [27], Vincent Vanackère presents *history-dependent scheduling*, a new technique for reducing the search space for the verification of cryptographic protocols. This technique allows the detection of some "non-minimal" interleavings during a depth-first search, and has been implemented in TRUST, our cryptographic protocol verifier. We give some experimental results showing that our method can signigicantly increase the efficiency of the verification procedure.

6.2.2. Non-interference in reactive systems

Non-interference is a program property asserting that a piece of code does not implement an information flow from classified or secret data to public results. In the past we have followed Volpano and Smith approach, using type systems, to statically check this property for concurrent programs. Motivation for this work is that one should find formal techniques that could be applied to mobile code, in order to ensure that migrating agents do not corrupt protected data, and that the behaviour of such agents does not actually depend on the value of secret information.

Ana Matos, in her thesis work under the supervision of Gérard Boudol and Ilaria Castellani, has adapted the results previously obtained in the MIMOSA project to the reactive programming approach that we develop. The reactive primitives offer new expressive power with reflections on the forms of security leaks that are to be controlled. On one hand we are able to code interference examples analogous to those we find in concurrent imperative languages, and to use analogous reasonings when designing the typing rules. On the other hand, the deterministic nature of the reactive concurrency calls for more restrictions on the set of accepted programs. In addition to the impact on the type system itself, the deterministic nature of the language allows for an alternative definition of "secure programs", which is usually based on a bisimulation. Inspired by the "Determinacy ⇒ (Observation equivalence = Trace equivalence)" result by Engelfriet we establish an equivalence between the bisimulation definition and an alternative one using trace-like reasoning. This simplifies the task of proving the soundness of the type system.

6.2.3. Controlling the complexity of code

This is a new research activity that started this year, and is now supported by the ACI "*Sécurité Informatique*" (CRISS Project). The objective is to design, study and implement static analysis techniques by which one can ensure that the computational complexity of a piece of code is restricted to some known classes. The motivation is primarily in the mobile code, to ensure that migrating agents are not using local resources beyond some fixed limits, but this could also apply to embedded systems, where a program can only use limited resources.

Quasi-interpretations are a way to bound the size of the values computed by a first-order functional program (or a term rewriting system) and thus a means to extract bounds on its computational complexity. We study in the paper [19] the synthesis of quasi-interpretations selected in the space of polynomials over the *max-plus* algebra determined by the non-negative rationals extended with -∞ and equipped with binary operations for maximum and addition. We prove that in this case the synthesis problem is NP-hard, and in NP for the particular case of *multi-linear* quasi-interpretations when programs are specified by rules of bounded size. The relevance of multi-linear quasi-interpretations is discussed by comparison to certain syntactic and type theoretic conditions proposed in the literature to control time and space complexity.

In ongoing work in the framework of the CRISS project, we have defined a simple stack machine for a related first-order functional language and shown how to perform type, size, and termination verifications at the level of the bytecode of the machine. We are also beginning to investigate whether the formal techniques for statically analyzing the complexity of programs can be adapted to a new proposal we make for mobile code, based on the reactive approach (this is the PhD programme of Frédéric Dabrowski).

6.3. Reactive programming

Participants: Raul Acosta-Bermejo, Frédéric Boussinot, Christian Brunette, Manuel Serrano.

6.3.1. Loft

LOFT is a thread-based language for concurrent programming in C. LOFT is closely related to FAIRTHREADS (actually, LOFT stands for *Language Over Fair Threads*). Its objectives are:

- to make concurrent programming simpler and safer by providing a framework with a clear and sound semantics.
- to get efficient implementations which can deal with large numbers of concurrent components.
- to allow lightweight implementations that can be used in the context of embedded systems with limited resources.
- to be able to benefit from parallelism provided by SMP multiprocessor machines.

A first implementation of LOFT consists in a rather direct translation of FAIRTHREADS in C. Standard modules are translated into automata while native modules, having the capacity to stay unlinked from any scheduler, are mapped to pthreads. A second implementation does not use Posix Threads anymore and thus, of course, cannot deal with unlinked threads. However this implementation is suited for embedded systems with limited resources and has been used for a little prey-predator demo running on the GBA platform of Nintendo. Efficient algorithms are used by the implementations of LOFT. For example, direct access to the next thread to execute is used by schedulers. The implementation adapts a proposal made by Olivier Para during a DEA internship. A first draft describing LOFT in details is available at http://www-sop.inria.fr/mimosa/rp/LOFT.

6.3.2. FairThreads in C

FAIRTHREADS offers a simple API for concurrent and parallel programming in C. Basically, it defines *schedulers* which are synchronization servers, to which threads can dynamically link or unlink. All threads linked to the same scheduler are executed in a cooperative way, at the same pace, and they can synchronize and communicate using broadcast events. Threads which are not linked to any scheduler are executed by the OS in a preemptive way, at their own pace. FAIRTHREADS is fully compatible with the standard Pthreads library and has a precise and clear semantics for its cooperative part. In particular, systems exclusively made of threads linked to one unique scheduler are actually completely deterministic. Special threads, called *automata*, are provided for short-lived small tasks or when a large number of tasks is needed. Automata do not need the full power of a native thread to execute and thus consume less resources.

FAIRTHREADS in C make parallelism possible (for example, on SMP architectures). Indeed, when a fair thread unlinks from a scheduler, it becomes an autonomous native thread which can be run in parallel, on a distinct processor. Also, distinct schedulers can be run by distinct processors.

6.3.3. Icobjs

A new ICOBJS system is currently under development in JAVA. It has the form of an IDE for ICOBJS programming. The new system unifies icobjs and workspaces in which icobjs are created. In this way, a workspace can contain other workspaces, and workspaces can have associated behaviours. A specialized reactive engine has been developed for this specific system. The engine implements an efficient algorithm in which there is no inter-instant action performed by instructions awaiting absent events. A new algorithm to remove unused events has been introduced in order to deal with systems needing large numbers of icobjs. Simulations in Physics have been ported on the new system. Finally, means for migration and serialization have been included in the system.

6.3.4. Rejo and Ros

The ROS-REJO language and system have been completed and reported in the thesis of Raul Acosta-Bermejo [10]. Comparisons in terms of execution speed have been made between various reactive engines, as developed in our team, by France Telecom, and at the University of Paris 6, by Louis Mandel and Marc Pouzet. These comparisons are reported in the thesis of Raul Acosta.

6.4. Functional programming

Participants: Gérard Boudol, Yannis Bres, Damien Ciabrini, Erick Gallesio [University of Nice Sophia-Antipolis], Bernard Serpette [project Oasis], Manuel Serrano, Pascal Zimmer.

6.4.1. Types and recursion

We have pursued our work on the modelling of objects in an ML-like language enriched with extensible records. This motivated further investigations in two directions: type systems and implementation of recursion.

The most popular type system for the λ-calculus is the one of Hindley-Milner for which an algorithm inferring principal types exists. But other systems, more complex and more permissive, have been studied, for example *intersection types*. Different versions and inference algorithms have been proposed by Coppo, Dezani, Venneri, Ronchi della Rocca, and more recently by Kfoury and Wells. Their common point lies in their complexity and the difficulty to understand them. Motivated by some object constructions that are not typable in the Hindley-Milner approach, we have studied the intersection type discipline. We have designed a simple and concise algorithm for type inference that performs the same operations as Kfoury and Wells', with the advantage that the so-called "expansion variables" are no longer necessary. The corresponding results have been re-proved, either directly, or by giving the link with the previous systems: the inference algorithm gives a principal typing for strongly normalisable terms and gets decidable when restricted to a finite rank. Some variants of this algorithm and its extension to ML, references and recursion have been studied too. In parallel, we developed a prototype implementation of this algorithm; it helped us to check and experiment quickly our ideas during the design process. These results will appear in Zimmer's thesis, and a conference paper by Boudol and Zimmer is in preparation.

The underlying calculus that we use to prove results about type inference in the intersection type discipline is an extension of the λ-calculus, based on early ideas of Klop, where one never erases sub-computations that would be deleted by the ordinary β-reduction. We have shown in [21] that this calculus is also well-suited to study normalization: we give, by adapting and combining some standard techniques (most notably a conservation theorem, established by means of a finiteness of developments theorem, and weak normalization of typable terms) a new proof of the classical result that typability in the intersection type discipline implies strong normalization.

Our implementation of objects (see the previous reports) relies on a non-standard fixpoint construction for call-by-value programming languages. This recursion construct is generally unsafe, and its use in ML-like languages requires a specific static analysis (incorporated in the type system in our work). Recently, Derek Dreyer has proposed such a static analysis for a different implementation of the fixpoint construct. It is not clear however that Dreyer's type system supports type inference. We have shown, in an unpublished draft [30], that the notion of a "safeness degree" that we previously introduced can be used to design a type system (slightly less expressive than Dreyer's one) for Dreyer's fixpoint, that enjoys the standard properties of ML-like type systems (type safety and type inference of a principal type).

6.4.2. Bigloo

The studies and efforts of year 2003 on the BIGLOO system have focused on four points: (*i*) the design of a new debugger and a new memory profiler, (*ii*) improving the efficiency of bytecode generation for the JVM and .NET platforms, (*iii*) porting and integrating BIGLOO under Microsoft Windows, and (*iv*) the efficient implementation of FAIRTHREADS. We detail the first two points in the next sections. The last point is presented in Section 6.5.2.

6.4.2.1. Debugging

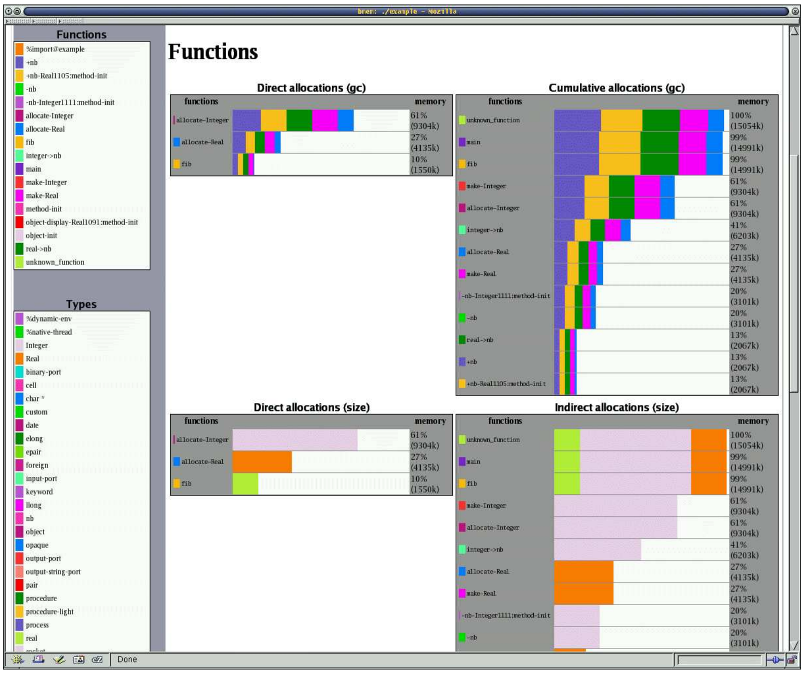
In practice, programmers hardly use debuggers. Obviously, they prefer *ad hoc* debugging techniques such as inserting `prints` in the source code. We believe debuggers can be made more convenient and thus more attractive. The JVM platform provides two standard APIs for implementing debuggers and profilers: JVMDI and JVMPI. One of the motivations for providing the BIGLOO compiler with a JVM back-end was to benefit from these two APIs to implement a debugger for SCHEME. We have developed BUGLOO, a source level debugger for SCHEME programs compiled with BIGLOO into JVM bytecode. It provides basic features found in classical C or Java debuggers, and also extended features like traces, debug sessions and memory debugging. It limits the loss of interactivity due to the compilation of programs by providing programmable breakpoints and interpreters embeddable into the debuggee. To make BUGLOO easily accessible, we have developed a complete user interface embedded into the Bee Emacs programming environment. We found that the JVM platform offers advantages over existing debugging platforms. Above all, thanks to the JIT compiler, the same version of a library can be used for both debugging and performance. This greatly eases the use of the debugger for programmers. Tests have shown that BUGLOO is usable in practice to debug large SCHEME programs such as the BIGLOO compiler itself, because the JIT allows to keep good a performance when programs are debugged. BUGLOO is available at (http://www-sop.inria.fr/mimosa/fp/Bugloo).

6.4.2.2. Bmem, a memory profiler

Automatic memory management is less and less controversially reckoned as beneficial: (*i*) it is a methodology of choice for drastically eliminating all nasty memory errors, and (*ii*) frequently it improves performance by enforcing locality of data. However, automatic memory management, generally implemented by means of garbage collectors (e.g. GC), has a dark side: by removing the burden of programmers that consists in handling carefully the memory it also makes these programmers less concerned by the allocations. This as consequences. For instance, library functions of languages that do not support garbage collection rarely allocate memory. Otherwise, it must be made explicit either in the documentation or using a naming convention, that the memory has to be deallocated. Nevertheless, mixing library-allocation with user-deallocation is error prone. A rule of thumb of programming is that the same piece of code is responsible for allocating and deallocating memory, unless automatic memory management is deployed. In fact, this drives the design of the `libc` (the C library) where no function allocates memory. Instead, these functions, take, as parameters, pointers to memory chunks, allocated and deallocated by the programs. For languages with GC (e.g. SCHEME, Caml, Java, JavaScript, C#, ...), this is another story. A library function may allocate since the deallocation is automatic. A library function does not even necessarily inform its user that it allocates some memory. In consequence, programs relying on GC frequently use too much memory. It is also difficult to figure out where and why the memory is allocated because, the memory allocation sites are not necessarily apparent in the programs. Hence, we need tools for solving this problem. BMEM, our memory profiler, is one of them.

BMEM is a profiler for SCHEME, for the BIGLOO system. It is easy to use because it does not require any special compilation flags. It can be used on all existing already compiled BIGLOO programs. This ease of use is obtained by means of the highly unsecure Unix `LD_PRELOAD` facility. This enables one to *preload* shared libraries before the ones loaded by the Unix loader. These preloaded libraries can override standard functions. BMEM uses a preloaded library that redefines the allocation functions of the BIGLOO GC library. This method is efficient and, as we have said, guarantees a comfortable ease of use, unfortunately, it is difficult to port it to other operating systems.

BMEM reports, at the end of the execution (or when the running program receives a `SIGINT` signal) the number of collections, and the heap size at each collection point. For each function *f*, it reports the memory allocated by *f* and the exact quantity of memory allocated by the functions called by *f*. In addition to the memory size, BMEM also reports on how this memory is organized (how many pairs, how many numbers, how many vectors, etc.). For each kind of objects used in the program, BMEM reports which functions allocate them and at which collection they have been allocated. Figure 1 presents a screenshot of Mozilla rendering the information collected during an execution of a SCHEME program.



Functions

- %import@example
- +nb
- +nb-Real1105:method-init
- -nb
- -nb-Integer1111:method-init
- allocate-Integer
- allocate-Real
- fib
- integer->nb
- main
- make-Integer
- make-Real
- method-init
- object-display-Real1091:method-init
- object-init
- real->nb
- unknown_function

Types

- %dynamic-env
- %native-thread
- Integer
- Real
- binary-port
- cell
- char *
- custom
- date
- elong
- epair
- foreign
- input-port
- keyword
- llong
- nb
- object
- opaque
- output-port
- output-string-port
- pair
- procedure
- procedure-light
- process
- real

Functions

Direct allocations (gc)

functions	memory
allocate-Integer	61% (9304k)
allocate-Real	27% (4135k)
fib	10% (1550k)

Cumulative allocations (gc)

functions	memory
unknown_function	100% (15054k)
main	99% (14991k)
fib	99% (14991k)
make-Integer	61% (9304k)
allocate-Integer	61% (9304k)
integer->nb	41% (6203k)
allocate-Real	27% (4135k)
make-Real	27% (4135k)
-nb-Integer1111:method-init	20% (3101k)
-nb	20% (3101k)
real->nb	13% (2067k)
+nb	13% (2067k)
+nb-Real1105:method-init	13% (2067k)

Direct allocations (size)

functions	memory
allocate-Integer	61% (9304k)
allocate-Real	27% (4135k)
fib	10% (1550k)

Indirect allocations (size)

functions	memory
unknown_function	100% (15054k)
main	99% (14991k)
fib	99% (14991k)
make-Integer	61% (9304k)
allocate-Integer	61% (9304k)
integer->nb	41% (6203k)
allocate-Real	27% (4135k)
make-Real	27% (4135k)
-nb-Integer1111:method-init	20% (3101k)
-nb	20% (3101k)

Figure 1. Memory profiling with Bmem

6.4.2.3. Bytecode generation

The .NET bytecode generation is still in progress. In a recent experiment we found that BIGLOO source code compiled into the .NET bytecode run about 2 to 4 times slower than the same source code compiled to the JVM. Two reasons explain this. First, the .NET platform is not as mature as the JVM, specifically under Linux. On this operating system, the two implementations currently available, PNET and Mono are still unstable and not very efficient. Second, we have learned from the BIGLOOJVM experiment that producing efficient bytecodes for a JIT-ed platform requires a lot of fine tuning in order to please the dynamic compiler. The BIGLOO .NET back-end is not yet mature enough for this fine tuning to take place. In particular, we first have to implement some static transformations (the most promising one seems to be a register allocation because the .NET JIT compilers are unable to allocate registers efficiently for functions using too many temporary variables).

During this year we have drastically cleaned the source code of the compiler and the runtime system. In particular, we have restructured the compiler so that the three different back-ends (C, JVM and .NET) now share about 90% of their source code while they were sharing no more than 20% in the previous version.

6.4.3. Skribe

The necessity of designing and implementing a new system has been raised during spring 2003 when we have received the visit of Phil Wadler (from Avaya Inc.). During our discussions, and comparisons with other systems and languages (mostly Xquery/XML on the one hand and SCRIBE on the other hand) we have identified several points that were either missing or poorly designed in SCRIBE.

SCRIBE has been deployed two years ago. Ever since, it is used on a daily-basis for producing all kinds of documents. The remarks collected from users during this period and our own experience enabled us to analyze the pros and cons of the authoring approach it promotes. SKRIBE, the successor of SCRIBE, has been designed based on this analysis.

6.4.3.1. Parenthetic syntax

The design of the SKRIBE syntax is editor-oriented: it is compact and easy to edit with traditional text editors. Definitely, the SKRIBE syntax based on parenthetic sc-expressions (an extension to MacCarthy's s-expressions) is off the beaten track for XML. We are aware that this decision might scare many people and prevent them from using the system. However, we are confident that for hand-written documents, the approach of SKRIBE makes sense.

6.4.3.2. Multi-targets

SKRIBE is multi-targets. That is, from one source document, SKRIBE can be used to produce several output files, such as HTML files, PostScript and PDF documents. Even if the HTML and PostScript back-ends are mainly used, it has been found useful to be provided with other output formats. For instance, we have found the ASCII output format surprisingly convenient for sending drafts by mail or submitting a textual version of the abstract of our papers. In addition to the multi-targets capacity, SKRIBE is highly customizable. A document may use the specificities of a given output format. For instance, an HTML output may use a layout that suits Web browsing (e.g. navigation bar for fast browsing), a PostScript output may contain headers and footers displayed on each page. In consequence, documents can have different shapes according to the selected target format. So, complex Web pages as well as high quality printed documents can be authored in SKRIBE. Our Web pages and many of our articles are now all written with SKRIBE.

6.4.3.3. User versus implementor

A SKRIBE document is represented by an abstract syntax tree (SC-AST). From the users point of view, the SC-AST is an opaque data type supporting three kinds of operation: *construction*, *introspection*, and *writing*. This model enforces a strict separation between the users point of view and the implementors point of view. This has proven to be inconvenient. The three main drawbacks of this approach are:

- Each writer contains a predefined set of possible customizations (for instance, the LaTeX back-end supports an option for choosing the printing style of the document). Because of the opacity of writers, users do not have the capacity to add new customizations.
- Users cannot add new constructor options. Customizing constructors in such a way would require to change the internal representation of a given node and the behaviour of the writer associated with each back-end.
- Users cannot add new kinds of nodes. For instance, one could wish to add a node for representing slides.

These drawbacks are not a hindrance for basic uses of SKRIBE. However, some users require sophisticated customizations that are incompatible with the limits imposed by the current model.

6.4.3.4. Introspection

Documents contain cyclic references. For instance, a table of contents is made of references to the chapters and sections composing a document. For handling such references SKRIBE proposes a library of introspection functions. These functions introduce forward references and thus, their invocations must be delayed. This can yield subtle errors. So, we think that SKRIBE introspection should be redesigned.

6.4.3.5. Dynamic typing

SKRIBE intrinsically needs dynamic type checking. Contrarily to XML, it is not designed for supporting static verification of documents. We have chosen to explore this path because it eases the production of dynamic texts. In particular, this type checking enables the embedding of SCHEME inside SKRIBE expressions and vice versa. The multi-targets aspect of SKRIBE impacts the way types are checked. The typing rules must reflect the capacity of the back-ends: (*i*) Some constructions are legal for some back-ends and illegal for other ones. (*ii*) Constructions options are back-end independent. However, certain of these options only make sense for some back-ends. future versions should be able to detect a situation where a source document is incompatible with a given back-end.

6.5. Combining Reactive and Functional programming

Participants: Gérard Boudol, Frédéric Boussinot, Christian Loitsch, Manuel Serrano.

6.5.1. Reactive and mobile programming: Ulm

We are beginning to investigate a combination of functional, reactive and mobile programming. The motivation for this is the fact that the reactive style of programming, providing a notion of reaction and time-out, looks appropriate for computing in a “global” context, where one has to deal with various kinds of failures – such as unpredictable delays, transient disconnections, congestion. We have designed a programming model that combines a core imperative and functional language with some primitives to deal with signals, directly inspired from the reactive style, and with constructs for strong mobility of threads. The main innovation in this model is the generalization of the notion of *suspension* – from reactive programming, where a thread may be suspended, waiting for a signal to be emitted: in our model, each operation involving the access to some resources (including pointer values or library programs for instance) is potentially suspensive, meaning that the code executing it may be waiting for the availability of the resource. In a draft paper (submitted to a conference) [31], we have investigated the expressive power of the proposed computing model, called ULM (for “*Un Langage pour la Mobilité*”). In particular, we have shown how to derive most of the standard constructs of reactive programming. We also introduce a static analysis to ensure that some pointer values are actually always available, thus allowing for a more efficient implementation.

6.5.2. FairThreads in Scheme

FAIRTHREADS offers a very simple framework for concurrent and parallel programming. Basically, it defines *schedulers* which are synchronization servers, to which fair threads are linked. All threads linked to the same scheduler are executed in a cooperative way, at the same pace, and they can synchronize and communicate using broadcast events. Threads which are not linked to any scheduler are executed by the OS in a preemptive way, at their own pace.

FAIRTHREADS is implemented in C, JAVA, and SCHEME. Implementations differ in the way unlinked threads are defined. In SCHEME, unlinked threads, called *service threads*, are not user-defined. In JAVA and C, unlinked threads are mapped to kernel threads and programming constructs are provided for dynamically linking and unlinking threads.

The main activity around FAIRTHREADS in SCHEME has been to polish the implementation. The BIGLOO system supports two execution platforms: (*i*) native executions on POSIX-1 compliant operating systems, (*ii*) Java executions on JVM. On both systems, user and service threads are mapped into native threads. POSIX threads and Java threads are close, so the implementation of FAIRTHREADS on both systems is very similar.

Naïve implementation: Conceptually, an execution token is passed through user threads and schedulers. In order to determine which threads can receive the token, the scheduler examines all user threads. It eliminates the ones that are explicitly blocked and the ones that are waiting for a non-present signal. When a thread yields explicitly or implicitly, it gives back the token to the scheduler. Service threads are implemented by means of native threads. That is, each time a service thread is spawned, the fair scheduler starts a new native thread that runs in parallel until completion and then broadcasts a signal.

Actual implementation: The actual implementation in the BIGLOO system, improves the naïve implementation in several directions. First, it avoids busy waiting. Second, it minimizes the creation of native threads. Third, it reduces the number of context switches.

– Avoiding busy-waiting: It is of extreme importance to efficiently implement signal broadcast because this operation is ubiquitous in the FAIRTHREADS programming style. Because the model does not propose point to point communication and because broadcasting is the only communication means between user and service threads. In the actual implementation, a thread does not consume the processor resource when waiting for a signal. The scheduler avoids active waiting by placing threads into wait queues associated with signals.

– Minimizing thread creations: Service threads are intensively used and in general they have a very short lifetime. So, the overhead imposed by spawning a service thread must be kept as minimal as possible. A naïve implementation that would start a new native thread each time a service thread is created would be inefficient. Using a pool of threads would improve the global performance because it would get rid of thread creation time. However, it would probably not perform sufficiently well because it would not reduce the context switches between user and service threads. The current implementation deploys an ad-hoc optimization that eliminates the thread creations and most of the thread context switches. It relies on the observation that the general pattern for spawning a service thread is: `(thread-await! (make-...-signal ...))`
  That is, the current user thread blocks until the service thread completes and broadcasts its signal. One may take advantage of this situation. It is possible not to allocate a native thread for implementing the service thread but instead to reuse the user thread. This optimization takes place inside the `thread-await!` function. When its argument is created by one of the `make-...-signal` library functions it *detaches* the user thread from the scheduler to make it execute the service required. In other words, the user thread becomes *temporarily* a service thread. On completion, this thread broadcasts a signal and gets *re-attached* to the scheduler.
  It might be possible that some service threads have been created but not awaited (for instance, when writing some data to a port, it is common not to check or not to wait for the result). So, at the end of the instant, when all user threads are blocked, the scheduler starts native threads for these pending service threads.

– Minimizing context switches: FAIRTHREADS context switches are expensive. A token is shared by all user threads. Allocating the CPU to a new user thread involves a lock or mutex acquisition and a notification. The less context switches, the more efficient the system.

One way to minimize context switches is to avoid, as much as possible, to switch back and forth to the scheduler. In particular, when a user thread *U1* cooperates, it is useless to switch back to the scheduler which, in turn, switches forth to the next runnable thread *U2*. *U1* can directly give the token to *U2*. The scheduling algorithm is thus actually split over all user threads. Each of them executes a small part of the scheduling task. In this framework, the thread running the scheduler gets the CPU only at the end of the instant. It might happen that all user threads are currently used to execute service threads. In such a situation there must exist another thread, waiting for the first broadcast signal. This is the role of the scheduler.

6.5.3. FairThreads based Web servers

We are conducting several experiments in order to understand if the FAIRTHREADS are suitable to implement efficient Web servers. Our preliminary results are rather negative. At this date, we have not been able to implement a server that is able to compete with top servers (Apache in particular). Currently, our best server lags far behind Apache. It is about five times slower. We have identified two main sources of inefficiency that we have not been able to fix yet:

- The parsing of HTTP requests needs a true expertise. HTTP syntax is loose and cluttered with many ambiguities. Traditional parsing techniques (those deployed in compilers) do not fit well with this problem. In addition, for this experiment we are using the SCHEME-based version of the FAIRTHREADS. As it is yet unclear if FAIRTHREADS compromise performance, it is unclear if SCHEME enables fast HTTP requests parsing. It seems that fastest HTTP server implementations play many tricks with C strings (pointers to characters). These tricks have no counterpart in SCHEME. So, using a safe high-level language such as SCHEME seems to be a handicap for high performance. However, we are bound to use SCHEME because it will ease the user-customization and user-scripting of the server. Ubiquitous customization and scripting being one of our prime goals for our Web server.
- The FairScheduler (the scheduler that schedules the fair threads) switches too frequently from one thread to another. That is, in addition to the context switches actually requested by the program, the scheduler, for its own purpose, switches from context to context. These context switches are tightly related to the semantics of FAIRTHREADS. However we have designed effective optimizations that get rid of all of them but one! We are confident that an additional effort, will also eliminate it.

7. Contracts and Grants with Industry

7.1. Rotor grant

We have received a grand from Microsoft to adapt the Bigloo compiler to the .NET platform. The task is twofold. On the one hand, it requires to add a new bytecode generator to the compiler. On the other hand, it requires to extend the runtime system for the integration in the Microsoft “Common Language Runtime” environment. The Bigloo version 2.6a, released in July, was the first version to contain the .NET bytecode generator. The second version, the version 2.6b released in December, improves the performance and enlarges the coverage of the .NET back-end. The port is now nearly complete. Manuel Serrano, Bernard Serpette (Oasis), and Yannis Bres are in charge of this contract.

8. Other Grants and Activities

8.1. National initiatives

8.1.1. CNRS Specific Action Formal Methods for Mobility

Roberto Amadio was the coordinator of this Action, which ended in October 2003. The other participants were the CNRS laboratories LIENS, LIPN and PPS from Paris, and the LIP from Lyon. The action had three main directions of research, in logical foundations (game semantics for concurrent systems, linear logic and complexity), mobile code (security issues), and semi-structured data.

8.1.2. ACI Cryptologie VERNAM and AZURCRYPT

We participated in two projects of the “Action Concertée Incitative Cryptologie”, namely VERNAM and AZURCRYPT, which both ended this year. Roberto Amadio was the coordinator of the first one. The other partners of VERNAM were the PROTHEO project of INRIA Lorraine and the *École Normale Supérieure* of Cachan. The main topic of this action was the automatic verification of cryptographic protocols.

8.1.3. ACI Sécurité Informatique CRISS

This action started in July 2003. The participants are, besides MIMOSA and the *Laboratoire d'Informatique Fondamentale* of Marseilles, the LIPN from Paris (Villetaneuse) and the INRIA project CALLIGRAMME from LORIA in Nancy. Roberto Amadio is the coordinator of the CRISS action. Its main aim is to study security issues raised by mobile code.

8.1.4. ACI Sécurité Informatique ROSSIGNOL

Roberto Amadio and Vincent Vanackère are participating in this action (started in July 2003), the topic of which is the verification of cryptographic protocols. The action involves the participation of INRIA Futurs, LSV Cachan, and VERIMAG Grenoble.

8.1.5. ACI Nouvelles Interfaces des Mathématiques GEOCAL

Roberto Amadio and Gérard Boudol are participating in this action. The other teams are the LMD Marseilles Luminy (coordinator), PPS Paris, LCR Paris Nord, LSV Cachan, LIP Lyon (PLUME team), INRIA Futurs, IMM Montpellier, and LORIA Nancy (CALLIGRAMME project).

8.1.6. ARC Concert

Manuel Serrano is involved in this project aiming at producing a realistic certified compiler, with a proof of equivalence between source and generated codes, checked in Coq. The participants in this action are the INRIA projects CRISTAL, LEMME, MIMOSA, MIRO, OASIS, and the CPR team at CNAM Paris.

8.2. European initiatives

8.2.1. IST-FET Global Computing project MIKADO

The participants in the MIKADO project are INRIA (SARDES project, Rhône-Alpes, and MIMOSA), acting as the coordinator, France Télécom R&D Grenoble, and the universities of Sussex, Lisbon, Florence and Turin. The objectives of the MIKADO project are to study programming models for distributed and mobile applications, the study of related specification and analysis formalisms, and the design of relevant programming constructs and of corresponding prototypical virtual machines.

8.2.2. IST-FET Global Computing project PROFUNDIS

In this project our partners are KTH Stockholm (coordinator), the Scientific and Technological University of Lisbon, and the University of Pisa. Its objectives are the study of proof techniques (logics, behavioural equivalences, type systems) for mobile systems.

8.3. Visiting scientists

Giorgio Delzanno from the University of Genova visited Marseille for one month. Matthew Hennessy, from Sussex University, and António Ravara, from the Instituto Superior Técnico de Lisboa, both visited MIMOSA at Sophia Antipolis for one week. Rob van Glabbeek was an invited professor visiting our team for three months.

Kirill Kislovski and Dmitry Lizorkin from the Institute for System Programming of the Russian Academy of Science visited us for one week in March. Phil Wadler, from Avaya Inc., also visited us for one week.

9. Dissemination

9.1. Seminars and conferences

Roberto Amadio spent a sabbatical semester at the University of Munich (where he gave several seminars). He attended the Global Computing workshop held in Rovereto. He participated in the conference on Typed Lambda-Calculi and Applications in Valence, where he gave a talk on [19].

He also gave seminars in Paris, and at a meeting of the “Action Spécifique Sécurité” in Cachan. He participated in a meeting of the “Action Spécifique Modèles formels de la Mobilité” in Lyon, and he gave a talk on [19] at the workshop “Process Algebra: Open Problems and Future Directions” held in Bertinoro.

Gérard Boudol participated in the “Types in Global Computing” meeting in Paris. He gave a seminar at Sussex University on the recursive semantics of objects, and at a meeting of the “Action Spécifique Sécurité” in Cachan, on typing non-interference in concurrent and reactive programming languages. He attended the Global Computing workshop held in Rovereto. He participated in a meeting of the “Action Spécifique Modèles formels de la Mobilité” in Lyon, and in the MIKADO-MYTHS meeting in Brighton, were he gave a talk on [29]. He participated in the conference on Typed Lambda-Calculi and Applications in Valence, where he gave a talk on [21]. He was invited to give a talk, on programming models for mobility, at the Italian Conference on Theoretical Computer Science, held in Bertinoro. He participated in the first meeting of the “Action Concertée Incitative Sécurité Informatique” in Rennes.

Frédéric Boussinot presented the reactive approach in a meeting with a research group from Dassault Systèmes. He participated in the Synchron’03 workshop, where he presented his work.

Christian Brunette gave a talk at the Synchron’03 workshop.

Yannis Bres participated to the VSIP and Rotor Microsoft workshops organized by MSR in Cambridge. He also participated to the Rotor Microsoft workshop organized in Redmond. He gave a talk on the Bigloo .NET back-end at the Inria Intech seminar in Grenoble.

Ilaria Castellani gave a talk on the axiomatization of asynchronous semantics at the workshop “Process Algebra: Open Problems and Future Directions” held in Bertinoro. She attended the meeting of the “Action Spécifique Sécurité” in Cachan, and took part in the MIKADO-MYTHS meeting in Brighton, and to the Global Computing workshop in Rovereto. She participated in the workshop on “Language-Based Security” in Dagstuhl, and in the workshop “Sex and Gender in Scientific Work” in Paris.

Damien Ciabrini participated in the 3th International Lisp Conference, where he gave a talk on [22].

Silvano Dal-Zilio gave a talk on decision problems for spatial logic at the “Types in Global Computing” meeting in Paris. He attended the Global Computing workshop held in Rovereto. He gave a talk on [23] at the conference on Rewriting Techniques and Applications in Valence.

Ana Matos attended a meeting of the “Action Spécifique Sécurité” in Cachan. She participated in the Mini School Chambéry-Turin of Theoretical Computer Science in January 2003, as well as the Fields Institute Summer School in June 2003, Ottawa. She attended the Foundations of Global Computing workshop in Eindhoven, and gave a talk at the workshop on “Language-Based Security” in Dagstuhl, on typing non-interference for a reactive programming language.

Manuel Serrano participated in the VSIP Microsoft workshop organized by MSR in Cambridge. He wrote an article on C++ for the encyclopedia “Techniques de l’Ingénieur”

Vincent Vanackère visited Pisa University for one week, to work on the integration of his TRUST software in the tools developed withing the PROFUNDIS project.

9.2. Animation

Roberto Amadio is head of the team “Modélisation et Vérification” of the Laboratoire d’Informatique Fondamentale of the University of Provence and Méditerrannée. He was chair of the programme committee of the CONCUR’03 conference, and member of the programme committee for the workshop Fixed Points in Computer Science. He is a member of the programme committee for the conference COORDINATION’04. He participates in the steering committees of CONCUR, and of the French Spring School in Theoretical Computer Science. He was a referee for the PhD theses of Véronique Cortier (ENS Cachan) on the automatic verification of cryptographic protocols, of M. Turuani (LORIA) on decidability and complexity problems related to the analysis of cryptographic protocols, and of J.-Y. Moyen (LORIA) on the analysis of the complexity of programs. He was a coordinator of the CNRS Specific Action "Formal Methods for Mobility" and of the ACI Cryptology VERNAM. He is coordinator of the ACI Sécurité Informatique CRISS.

Gérard Boudol was a referee for the PhD thesis of Tom Hirschowitz (University of Paris 7 and INRIA project CRISTAL). He was an examiner for the PhD thesis of Ludovic Henrio (UNSA and INRIA project OASIS), and for the "Habilitation à Diriger des Recherches" of Loïc Pottier (INRIA project LEMME).

Ilaria Castellani is a member of the programme committee and of the organization committee of the "Forum des Jeunes Mathématiciennes".

Silvano Dal-Zilio was the chair of the scientific and organization committee for the workshops affiliated with the CONCUR'03 conference. He was the chair of the programme committee of the workshop Formal Methods for Mobility (Marseilles, September 2003).

Manuel Serrano was co-editor of two special issues of the journal "Higher-Order and Symbolic Computation" about the programming language SCHEME. He is a member of the "SCHEME Strategy Group" that decides on the evolutions of this programming language.

Pascal Zimmer is a moderator of the Mobile Calculi electronic forum.

9.3. Teaching

Roberto Amadio gave a course at the DEA Informatique of Marseilles universities, on functional programming and resource control, and at a DESS on cryptology. He teaches formal languages and compilers at a graduate level. He supervised the internship of Eric Leviel (ENS Paris).

Frédéric Boussinot together with Raul Acosta, Christian Brunette, and Manuel Serrano, gave a course on "Objets réactifs en Java" at UNSA (DEA Informatique and RSD, and ESSI). He supervised the DEA internship of Olivier Para on the reactive implementation of embedded systems.

Silvano Dal-Zilio supervised the internship of Etienne Duchesne (ENS Paris) on type systems in the π-calculus.

Manuel Serrano gave lectures at the "DEA d'Informatique" of the University of Nice. He supervised the internship of Christian Loitsch (ESSI Nice).

Pascal Zimmer teaches functional programming (exercise groups, DEUG MI2), programming in C and relevant tools (exercise groups, Licence Informatique), and the semantics of programming languages (exercise groups, Maîtrise d'Informatique) at the university of Nice and Sophia-Antipolis.

10. Bibliography

Major publications by the team in recent years

[1] R. AMADIO, P.-L. CURIEN. *Domains and Lambda-Calculi.* Cambridge University Press, 1998.

[2] G. BERRY, G. BOUDOL. *The chemical abstract machine.* in « Theoretical Computer Science », volume 96, 1992.

[3] G. BOUDOL. *The π-calculus in direct style.* in « Higher-Order and Symbolic Computation », volume 11, 1998.

[4] F. BOUSSINOT. *Objets réactifs en Java.* Collection Scientifique et Technique des Telecommunications, PPUR, 2000.

[5] F. BOUSSINOT. *La programmation réactive.* Masson, 1996.

[6] F. BOUSSINOT, J.-F. SUSINI. *Java threads and SugarCubes.* in « Software Practice & Experience », volume 30, 2000.

[7] I. CASTELLANI. *Process Algebras with Localities.* J. BERGSTRA, A. PONSE, S. SMOLKA, editors, in « Handbook of Process Algebra », North-Holland, Amsterdam, 2001, pages 945-1045.

[8] D. SANGIORGI, D. WALKER. *The π-Calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.

[9] M. SERRANO. *Bee: an Integrated Development Environment for the Scheme Programming Language.* in « Journal of Functional Programming », number 2, volume 10, May, 2000, pages 1–43.

Doctoral dissertations and "Habilitation" theses

[10] R. ACOSTA-BERMEJO. *Rejo - Langage d'objets réactifs et d'agents.* Ph. D. Thesis, Thèse de doctorat de l'ENSMP, Oct, 2003.

Articles in referred journals and book chapters

[11] R. AMADIO, G. BOUDOL, C. LHOUSSAINE. *On message deliverability and non-uniform receptivity.* in « Fundamenta Informaticae », number 2, volume 53, Dec, 2002, pages 105–129.

[12] R. AMADIO, G. BOUDOL, C. LHOUSSAINE. *The distributed receptive pi-calculus.* in « ACM Transactions of Programming Languages and Systems (TOPLAS) », number 5, volume 25, 2003, pages 1–29.

[13] R. AMADIO, D. LUGIEZ, V. VANACKERE. *On the symbolic reduction of processes with cryptographic functions.* in « Theoretical Computer Science », number 1, volume 290, Dec, 2002, pages 695–740.

[14] W. CHARATONIK, S. DAL ZILIO, A. GORDON, S. MUKHOPADHYAY, J.-M. TALBOT. *Model Checking Mobile Ambients.* in « Theoretical Computer Science », number 1, volume 308, Nov, 2003, pages 277–331.

[15] E. GALLESIO, M. SERRANO. *Programming Graphical User Interfaces with Scheme.* in « Journal of Functional Programming », number 5, volume 13, Sep, 2003, pages 839–886.

[16] M. SERRANO. *Langage C++.* series Techniques de l'Ingenieur, volume H 3 078, Techniques de l'ingénieur, 249, rue de Crimée, 75925 Paris, Cedex 19, France, 2003, pages 1–19.

[17] D. TELLER, P. ZIMMER, D. HIRSCHKOFF. *Using ambients to control resources.* in « Schedae Informaticae », volume 12, 2003, pages 113–127, Special issue: Chambery-Krakow-Lyon Workshop on lambda calculus, type theory and mathematical logic.

[18] P. ZIMMER. *On the expressiveness of pure safe ambients.* in « Mathematical Structures in Computer Science », volume 13, 2003.

Publications in Conferences and Workshops

[19] R. AMADIO. *Max-plus quasi-interpretations.* in « Proc. Typed Lambda Calculi and Applications (TLCA 2003) », series Lecture Notes in Computer Science 2701, Valencia, Spain, 2003.

[20] R. AMADIO, D. LUGIEZ (EDITORS). *Proceedings of the International Conference on Concurrency Theory – CONCUR 2003.* series Lecture Notes in Computer Science, 2761, Marseille, France, Sep, 2003.

[21] G. BOUDOL. *On strong normalization in the intersection type discipline.* in « TLCA'03 », series Lecture Notes in Computer Science, number 2701, pages 60-74, 2003.

[22] D. CIABRINI, M. SERRANO. *Bugloo: A Source Level Debugger for Scheme Programs Compiled into JVM Bytecode.* in « 3th International Lisp Conference », New York, USA, Oct, 2003.

[23] S. DAL ZILIO, D. LUGIEZ. *XML Schema, Tree Logic and Sheaves Automata.* in « RTA 2003 – 14th International Conference on Rewriting Techniques and Applications », series Lecture Notes in Computer Science, volume 2706, Springer, pages 246–263, Jun, 2003.

[24] S. DAL ZILIO, D. LUGIEZ. *A Logic you Can Count On.* in « POPL 2004 – 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages », 2004.

[25] X. GUAN. *Towards a tree of channels.* in « Electronic Notes in Theoretical Computer Science », volume 85, Elsevier, V. SASSONE, editor, 2003.

[26] A. RAVARA, A. MATOS, V. VASCONCELOS, L. LOPES. *Lexically scoped distribution: what you see is what you get.* in « Workshop on Foundations of Global Computing », number 1, volume 85, 2003.

[27] V. VANACKERE. *History-Dependent Scheduling for Cryptographic Processes.* in « Proc. VMCAI' 2004 », series Lecture Notes in Computer Science, Venice, Italy, 2004.

[28] P. ZIMMER. *On the expressiveness of pure mobile ambients.* in « Electronic Notes in Theoretical Computer Science », volume 39, Elsevier, L. ACETO, B. VICTOR, editors, 2003.

Miscellaneous

[29] G. BOUDOL. *Core Programming Model V1: elements for a MIKADO programming model.* Sep, 2003, Deliverable D1.2.2, IST-FET Global Computing Project MIKADO.

[30] G. BOUDOL. *Safe recursive boxes.* Sep, 2003.

[31] G. BOUDOL. *ULM, a core programming model for global computing.* Oct, 2003.

[32] M. SERRANO, E. GALLESIO. *Skribe: a Functional Language for Programming Documents.* 2003.