INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team mimosa

Migration et Mobilité: Sémantique et Applications

Sophia Antipolis

THEME COM

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]

Staff members Inria
Manuel Serrano [Research Director, Inria]

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

Visiting scientist
Erick Gallesio [Visiting Scientist, University of Nice Sophia-Antipolis]
Maria-Grazia Vigliotti [Visiting Scientist, from October 1]

Ph. D. students
Lucia Acciai [MENRT]
Damien Ciabrini [MENRT]
Frederic Dabrowski [MENRT]
Daoudou Maoulida [Inria, from October 1]
Stéphane Epardaud [MENRT]
Florian Loitsch [MENRT, from October 1]
Ana Matos [Portuguese Gov.]
Charles Meyssonnier [ENS Lyon]

Internship
Florian Loitsch [Master University of Nice, till October 1]
Daoudou Maoulida [Summer internship, from July 15]
Celio Trois [Master University of Nice, till October 1]

2. Overall Objectives

2.1. 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 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 a mobile code, a logical rather than physical notion of mobility. An important task in this area has been to understand the various constructs that have been 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 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 focusses 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 this model may look, from a computational point of view, 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 have, in particular, used models derived from the π-calculus for the formalization and verification of cryptographic protocols.

We are now 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 time-out and reaction, allowing a program to deal with the various kinds of failures (delays, disconnections, etc.) that arise in a global network. We have started the design and implementation of a core programming language that integrates reactive programming and mobile code, in the context of classical functional and imperative programming. In this setting, we use standard techniques to address security issues: for instance, we use type and effect systems to statically ensure the properties of integrity and confidentiality of data manipulated by concurrent programs. We also 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. Scripting

Because functional languages offer a high level of abstraction, they generally enable compact implementations. So, they enable fast prototyping and fast implementing. In consequence, they are generally convenient when used as scripting languages. For this reason, many famous end-user applications (such as Emacs, Gimp, Auto-Cad, ...) embed interpreters of functional languages. The compilation of functional languages, at least for the representatives that use strict evaluation order, is now well understood. Hence, programming in a functional language does not forbid to produce fast applications that do not clutter the computers they run on. With some of the modern implementations, it is possible to blend compiled code, for fast execution, and interpreted code, for scripting. The combination of both execution modes brings expressiveness *and* efficiency. Few other languages offer this capability. Exploiting this specificity we have conceived an email synchronizer that is implemented in Scheme and that also uses this language for supporting user scripting.

4.4. Web servers and Web proxies

Since a couple of years programming the Web is a hot topic. It involves network, distributed, and concurrent programming. In addition, it also requires scripting facilities. Naturally , the techniques conceived and the languages designed in the Mimosa project could naturally be applied to this programming area. In particular, we are considering applying the FAIRTHREADS to the design and implementation of a *user-land proxy*. This proxy should be uncluttering and lean, spawned by users. It should be easy to start, easy to stop, and highly customizable and *scriptable*. This Web proxy 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. This user-land 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 found, 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 (such as the SugarCubes and Rejo-Ros) 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: Frédéric Boussinot, Stéphane Epardaud.

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. 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).

5.2.3. LURC

LURC is a Reactive threading library in C. It is based on the reactive model of ULM (*Un langage pour la mobilité*) and the desynchronization feature of FAIRTHREADS in C. It provides several types of thread models, each with different performance tradeoffs at run-time, under a single deterministic semantics. Its main features as taken from ULM are preemption, suspension, cooperation and signal emission and waiting. On top of that, threads can switch from asynchronous to synchronous at will. Event-loop programming has been integrated in a reactive style under the form of a Reactive Event Loop. The main difference with the syntax of LOFT, another threads library developed in the team, is that LURC is a pure C library, on top of which a pseudo-language layer can be added in the form of C macros in order to make reactive primitives look and behave like language primitives. LURC is available on the INRIA website at the following URL: http://www-sop.inria.fr/mimosa/Stephane.Epardaud/lurc.

5.3. Functional programming

Participants: Damien Ciabrini, Stéphane Epardaud, Erick Gallesio, Bernard Serpette [Project Oasis], Manuel Serrano.

5.3.1. The Bigloo compiler

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. Nowadays, it runs harmoniously under Linux and MacOSX. The effort initiated in 2002 for porting to Microsoft Windows is pursued by external contributors. In addition to the native back-ends, the BIGLOO JVM back-end has enabled a new set of applications: Web services, Web browser plug-ins, cross platform development, etc. The new BIGLOO .NET CLR back-end that is fully operational since release 2.6e enables a smooth integration of Bigloo programs under the Microsoft .NET environment.

The main effort of 2005 has been to re-implement the runtime system for supporting preemptive concurrent programming. That is, the new Bigloo library is now thread-safe and its standard distribution comes with

two multi-threading library. A library for programming with Fair Threads and a library for programming with Posix-like threads. The first library offers security because it makes explicit locking and explicit mutual exclusion useless. The second library offers performance because it takes benefit from parallel architectures.
We have distributed one major releases of Bigloo during 2005, the version 2.7a.

5.3.2. Bugloo

Every programmer is frequently faced with the problem of debugging programs. Paradoxically, debuggers are hardly used in practice and have not evolved that much in the last decades. We believe these tools can be made more attractive by following some rules. Debuggers must be easily accessible from the programming environment. When using them, the performance slowndown must keep reasonable. At last, they have to match the specificities of the language of debugged programs.
These ideas have driven the design and implementation of BUGLOO, a source level debugger for Scheme programs compiled into JVM bytecode. It focuses on providing debugging support for the Scheme language specificities, such as the automatic memory management, high order functions, multi-threading, or the runtime code interpreter. The JVM is an appealing platform because it provides facilities to make debuggers, and helps us to meet the requirements previously exposed.

5.3.3. Skribe

SKRIBE 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.

5.3.4. ULM

The ULM Scheme implementation is an embedding of the ULM primitives in the Scheme language. This implementation provides a compiler and a virtual machine to execute ULM/Scheme programs. The current version has preliminary support for a mixin object model, reactive event loops, and native procedure calls with virtual machine reentry. The current version is available at http://www-sop.inria.fr/mimosa/Stephane.Epardaud/ulm.

5.4. Old softwares

5.4.1. 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. The Java version of ICOBJS 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.

5.4.2. TypI

TypI is a type inference interpreter for the intersection types discipline. It implements, in CAML, the algorithm designed and proved correct by Boudol and Zimmer. A reference manual (in french) can be found on the web page of the project, and is a chapter of Zimmer's thesis .

5.4.3. MLObj

MlObj is an interpreter for a prototype language composed of a functional core, objects, mixins and degree types, written in CAML. It implements Boudol's theory of objects as recursive records. A reference manual (in french) can be found on the web page of the project, and is a chapter of Zimmer's thesis.

5.4.4. Trust

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. It is available via the url http://www.cmi.univ-mrs.fr/~vvanacke/trust.html.

6. New Results

6.1. Semantics of mobility

Participant: Gérard Boudol.

The work on a "membrane calculus" done in the MIKADO project has been published [18].

6.2. Security

Participants: Roberto Amadio, Gérard Boudol, Ilaria Castellani, Frédéric Dabrowski, Silvano Dal Zilio, Ana Matos.

6.2.1. Controlling information flow

Non-interference is a property of programs asserting that a piece of code does not implement a flow of information 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. The motivation 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.

The non-interference property is very often questioned, on the basis that it cannot be used in practice because it rules out, by its very definition, programs that intentionally declassify information from a confidential level to a public one, like a password checking procedure for instance. We have addressed this problem, of how to combine declassification with a security analysis of programs, like typing the information flow. Our standpoint is that there are two different issues to be considered, namely *what* information is released and *how* information is declassified. To address this second question, we have introduced, in a Core ML-like language with concurrent threads, a declassification mechanism that takes the form of a local flow policy declaration.

The computation in the scope of such a declaration is allowed to implement information flow according to the local policy. This dynamic view of information flow policies is supported by a concrete presentation of the security lattice, where the confidentiality levels are sets of principals, similar to access control lists. To take into account declassification, and more generally dynamic flow policies, we introduce a generalization of non-interference, that we call the non-disclosure policy, and we design a type and effect system for our language that enforces this policy. Our workshop paper [16] has been selected for publication in the Journal of Computer Security.
In the paper [19], we further investigate the issue of typing confidentiality in a language-based information-flow security approach, aiming at improving some previously proposed type systems, especially for higher-order languages with mutable state à la ML. We show that the typing of terminations leaks can be largely improved, by particularizing the case where the alternatives in a conditional branching both terminate. Moreover, we also provide a quite precise way of approximating the confidentiality level of an expression, that ignores the level of values used for side-effects only.
Ana Matos has studied in [15] new forms of security leaks, called migration leaks, that are introduced in a mobile code scenario. The language used in [16] is extended with a notion of domain and a migration primitive. Then the non-interference property is generalized to networks of domains, and a type and effect system for enforcing it is presented and proved correct.
The full version of a paper of last year on non-interference for reactive programs has been submitted for publication in a journal [23].

6.2.2. Controlling the complexity of code

The objective of this research activity 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.
In the paper [17], we propose a compositional static analysis for a language of cooperative threads which guarantees that the size of the values computed by a program is bounded by the size of the parameters of the system at the beginning of the computation. This improves previous results by Amadio and Dal Zilio (see below) where bounds were functions of the parameters of the system at the beginning of each instant. The existence of bounds for arbitrary many instants relied on the assumption of a dynamic check of the size of the parameters of the system at the beginning of each instant which is now useless. In the same paper, termination of the instants is revisited using a more general criterion. As before, simplification orders (a kind of well-found orders used for proving termination of rewriting systems) are used but bounds provided by the static analysis are used instead of the embedding relation to prove the existence of a reduction order. These two results lead to the termination of the instants in time polynomial in the size of the parameters at the beginning of the computation.
In [20] we define a method to statically bound the size of values computed during the execution of a program as a function of the size of its parameters. More precisely, we consider bytecode programs that are to be executed on a simple stack machine with support for algebraic data types, pattern-matching and tail recursion. Our size verification method is expressed as a static analysis performed at the level of the bytecode, that relies on machine-checkable certificates. We follow here the usual assumption that code and certificates may be forged and should be checked before execution. Our approach extends a system of static analysis based on the notion of quasi-interpretations that has already been used to enforce resources bounds on first-order functional programs. This paper makes two additional contributions. First, we are able to check optimized programs, containing instructions for unconditional jumps and tail recursive calls, and remove restrictions on the structure of the bytecode that were imposed in previous works. Second, we propose a direct algorithm that depends on solving a set of arithmetical constraints.
The work by Roberto Amadio on quasi-interpretations which was presented in a 2003 report has been published [10]. The full version of the paper on *Resource control for synchronous cooperative threads* by

Roberto Amadio and Silvano Dal Zilio has been accepted for publication in the journal Theoretical Computer Science.

6.3. Reactive programming

Participants: Gérard Boudol, Frédéric Boussinot, Frédéric Dabrowski, Stéphane Epardaud.

6.3.1. The reactive model

In the note [25] we revisit the so-called *reactive* programming style, which evolves from the synchronous programming model of the ESTEREL language by weakening the assumption that the absence of an event can be detected instantaneously. We review some research directions that have been explored since the emergence of the reactive model ten years ago. We also outline some questions that remain to be investigated.

In [24] we revisit the SL synchronous programming model introduced by Boussinot and De Simone *(IEEE, Trans. on Soft. Eng., 1996)*. We discuss an alternative design of the model including *thread spawning* and *recursive definitions* and we explore some basic properties of the revised model: determinism, reactivity, CPS translation to a tail recursive form, computational expressivity, and a compositional notion of program equivalence.

6.3.2. Safe Concurrent Programming

We have made an experiment to add concurrency to the Cyclone programming language, in order to get a safe concurrent language. The basic model considered is that of FairThreads in which synchronous and asynchronous aspects are mixed. The language Loft implements the FairThreads model in C. The experiment basically uses Cyclone instead of C in the implementation of Loft. Using the multi-threaded version of Boehm's GC, one gets an extension of Cyclone to concurrency which is as safe as Cyclone for sequential code, with some additional safety verifications for concurrent code. Several static analyses should be added in order to get a completely safe language (for example, verification that atoms executed by linked threads indeed terminate). The difficulty to add these verifications in Cyclone leads us to leave aside this experiment and to explore a new way based on a novel language inspired from ML and considered in the section "Cooperative Threads and Preemptive Computations".

We propose [29] a small language for programming cooperative threads in which one can also define preemptive computations, to deal with tasks that are not well-suited to a cooperative treatment but are useful in practice (e.g. blocking I/Os). These preemptive computations are executed in preemptive mode (in parallel with the cooperative system). We have introduced a type and effect system which ensures that preemptive computations do not interfere with the normal behavior of cooperative threads. This system is used to associate to each thread a subset of the memory locations created by it, namely unsharable memory locations, which are not shared with other threads. During a preemptive computation a thread has access only to these memory locations. This eliminates data-races with the cooperative threads or with the other preemptive computations, thus preserving the atomicity of the execution provided by a cooperative model. An important consequence of this is that it allows the cooperative model to safely benefit from multi-processor architectures.

A first experimental implementation is under work. The compilation process consists in an type and effect inference algorithm, in several verifications (in particular, absence of instantaneous loops and termination of functions), and in a translation of FairThreads in C (via the Loft Language).

6.3.3. Reactive Programming: LURC

LURC, a lightweight reactive library based on ULM has been released this year. Originally a test project for studying new scheduling mechanisms in an optimized C library, it has grown into a full featured reactive multi-threading library with the addition of various thread implementations, all obeying a single semantics, a Reactive Event Loop and language-level features for reactive programming.

In LURC, there are four types of thread implementation, all scheduled similarly, but each with performance trade-offs at run-time. Two of them are purely cooperative and cannot become asynchronous: the first one minimises stack allocation with the cost of more memory copying when cooperating, the second one

minimises cooperation time at the cost of bigger memory usage. The last two types can switch from cooperative/synchronous to asynchronous: one minimises cooperation type when synchronous by sharing its executing native thread with that of purely synchronous LURC threads, at the cost of asynchonisation time, while the second minimises asynchronisation time at the cost of native thread switching when cooperative.
In addition to mixing several models of threads, LURC uses a new reactive scheduling mechanism, based on spreading the long lasting computations of the *end of instant* phase during the instant. Each thread attempts to schedule itself for the next instant at a new phase called the *end of action* (when cooperating in most cases). This effectively reduces the time delay required by most reactive schedulers between each instant, while increasing the delay of cooperation between threads, thus bringing a comparable time cost when scheduling is done between two threads within the instant and across instants.
Although LURC is written in POSIX C as a user-level library, it can take advantage of the GCC compiler (the most widely distributed and used C compiler) when present and offer language-level primitives for most reactive primitives, via a complex use of GCC C language features and macros. This proves to be very portable and integrates the reactive primitives in a program as if they were C language primitives.

6.4. Functional programming

Participants: Gérard Boudol, Damien Ciabrini, Erick Gallesio, Florian Loitsch, Bernard Serpette [Oasis project], Manuel Serrano.

6.4.1. Types and recursion

The work by Boudol and Zimmer on type inference in the intersection type discipline has been published [11].

6.4.2. Bigloo

For the year 2005, our efforts on Bigloo have mainly focused on preemptive multi-threading. We have re-implemented many components of the runtime system for supporting re-entrance. For this we have had to design and implement a new mechanism for handling errors. The new system uses exceptions. The importance of this implementation effort has slowed down the production of Bigloo releases. We have not been able to produce more than one version this year (the version 2.7a). However, the activity around Bigloo as continued as approximatively the same pace as the previous years. The Bigloo community is still committed to its evolution. This is demonstrated by the numerous mails that are sent to its mailing list: this year, approximatively 1000 mails have been sent.
In addition to multi-threading, we have developed new APIs for Bigloo. Even if they are not yet part of official distribution, we have nearly completed the implementation of libraries for:

- secure networking via SSL.
- IMAP mail management.
- Web programming. This involves facilities for parsing and producing XML documents, parsing HTTP requests, handling URLs, and decoding CGI arguments.
- multimedia programming with facilities for handling MP3 and playlist files, Jpeg Exif data, sound-card, etc.

All these libraries are meant to be integrated to the standard Bigloo distribution.

6.4.2.1. Stack Virtualisation for Source Level Debugging

The compilation of high-level languages to general-purpose execution platforms draws some concerns when it comes to debugging. Indeed, abstractions that are not naively supported by the execution platform are emulated with intermediate data structures and function calls. Unfortunately, the details of the emulation are visible in the execution stack, and this unwanted information greatly reduces the effectiveness of debuggers.

We have developed a novel and language-neutral technique for constructing a *virtual view* of the stack [28], in order to mask intermediate function calls that were generated to emulate high-level abstractions, or even to recover logical frame information that was lost during the compilation process. In particular, virtual views enable the visualization of two disjoint code representations (e.g., natively compiled code and dynamically interpreted bytecode) into a single unified stack.

We have designed a complete set of virtualization rules to hide all the details of the compilation of Bigloo programs into JVM bytecode. We have achieved to mask every emulated language features, such as high order functions, generic functions, exception handling, or runtime code interpretation. Other experiments have been conducted on the Rhino and the Jython languages, in order to show that this technique can be applied on a wide variety of languages.

The complete implementation of this work, along with examples of virtualization rules for various languages has been integrated into the Bugloo distribution available on-line.

6.4.3. Skribe

During the year 2005, we have rewritten Skribe so that it can be embedded in a web browser. This rewriting was necessary because at its inception Skribe implementation has been designed as a batch document processing tool. We also took advantage of this rewriting to integrate some improvements based on our four years experience with Skribe development and usage.

The Skribe evaluator relies on three stages. In the first stage, the source document is parsed and a tree representing this document is built. The second stage is devoted to inter-document references resolution. Finally, the third phase is in charge of producing the final document. With this scheme, only the last stage should be dependent of the final document output format. However, this was not the case with the old implementation of Skribe: in order to produce documents with layouts highly adapted to the output media, it was possible to build a tree in the first phase which was dependent of the output format. In a batch approach, this is not a problem, since we need one execution of the Skribe evaluator per output (i.e. to produce a HTML version and a PDF version of the same document, the Skribe evaluator must be run twice).

With our work around Web servers, we think that it is interesting to embed Skribe in the server. In this approach, when a Skribe document is requested, it is parsed, a tree is built, the references are resolved and the final document is sent to the client. Since the web server is aware of the capacities of the requesting client, it can produce different documents for different browsers (with/without images, using/avoiding CSS, ...). In order to keep good performances, the two first phases can be done only once for a given document and cached by the server. So, when a previously served Skribe document is requested, the server only needs to produce the client dependent HTML. The model used in our previous implementation was a real hindrance to achieve the embedding of Skribe in a Web server. With our new implementation, a Skribe document could be represented on the server as a output-independent tree and an environment per client. This rewriting was necessary to evolve from the batch approach we had to an embedded one.

The new version of Skribe still needs to be polished before being officially distributed. It should be available by mid 2006. Once we have a stable version, we will be able to effectively start experimentations with embedded Skribe documents on the server.

6.4.4. Bimap

Low cost computers, ADSL, and wireless connections have made ubiquitous computing a reality. Because the Internet is now available nearly everywhere on the planet, most of us are nearly permanently connected. Many of us use various computers (maybe, one at home, one at work, and a roaming laptop). All these computers ideally use the same synchronized data. Enforcing this synchronization is not always so easy. Hopefully, some

dedicated tools such as Unison allow two replicas of a collection of files and directories to be stored on different hosts, modified separately, and then brought up to date by propagating the changes in each replica to the other. However, as convenient as these tools are for file and directory synchronization, they are of little help when considering email synchronization. We address the specific problem of synchronizing email in this study.
We have designed and developed Bimap [21], a tool for synchronizing email. It enables emails to be manipulated from different computers and localizations. A user can read, answer, and delete emails from various computers amongst which some can be momentarily disconnected. Bimap automatically propagates the changes to all these computers. Synchronizing email is a simple problem of synchronizing lists. Functional languages are therefore candidates of choice for implementing such algorithms. Bimap is implemented in one of them, namely Scheme, our favorite programming language. It benefits from the recent evolution of Bigloo. In addition to synchronizing mail, Bigloo is also able to filter and classify email. As such, Bimap could be a potential replacement for `procmail`. This is highly convenient because it enables email filtering with simple small Scheme scripts. Two such scripts have been presented: one for classifying emails that belong to mailing lists and a second one for implementing white-listing. Each of these scripts is no more that a few lines of Scheme code.

6.4.5. Jsigloo

Javascript is one of the most popular scripting languages available today. Interpreters are integrated into every dominant web-browser and Javascript hence benefits from a huge installation base. Despite their apparent syntactical differences Javascript and Scheme share many features: both are dynamically typed, they feature closures and allow for functions as first class citizens. A Javascript compiler should hence benefit from the research done on Scheme compilers. Instead of reimplementing a fully optimizing compiler we implemented Jsigloo [22], a Javascript to Scheme compiler. The produced Scheme code can then be compiled by already existing optimizing compilers like Bigloo. This approach takes advantage of the optimizations implemented in Bigloo, and benefits from the Bigloo's multiple backends (C, JVM and .Net).
Not all Javascript constructs can be directly mapped to Scheme expressions, and some constructs need special attention. In particular Javascript's `while`, `switch` and `with` statements needed special attention. These Scheme foreign constructs, and slightly different semantics for certain expressions make optimizations within Jsigloo necessary. Jsigloo features among others a typing pass and direct-call optimizations.
Some Javascript properties make important Bigloo optimizations ineffective, and the produced code is hence not as efficient as hand-written Scheme code. It is the goal of some of the previously mentioned optimization passes to prepare the code for further Bigloo optimizations, but Jsigloo does not yet achieve the same performance as hand-written Scheme code. With these optimizations Jsigloo is however able to compete with already existing Javascript compilers. Beside of *Rhino* it is now one of the most efficient Javascript compilers.

7. Contracts and Grants with Industry

7.1. JSM

We have received a funding from Texas Instruments for studying efficient code generations on the JSM, a proprietary architecture. This platform is composed of a traditional Java Virtual Machine and a classical Risc architecture. The originality of the approach comes from the blending of the two instruction sets. In particular, the top of stack of the JVM is mapped to dedicated registers of the Risc instruction set. Well known compilation techniques don't apply well and generating efficient code on this platform is challenging! In a join work with the Oasis team we have adapted the Bigloo compiler for this new architecture. We have measured the impact of code generation optimizations embedded in Bigloo when applied to the JSM. Then, we have developed a new optimization based on an optimistic register allocations that mixes physical registers and stack allocation. Bernard Serpette (Oasis) and Manuel Serrano have been in charge of this contract.

8. Other Grants and Activities

8.1. National initiatives

8.1.1. ACI Sécurité Informatique ALIDECS

Frédéric Boussinot is participating in the ACI Sécurité Informatique ALIDECS whose coordinator is Marc Pouzet. The ACI started october 2004. Paricipants are Lip6 (Paris), Verimag (Grenoble), Pop-Art (Inria Rhône-Alpes), Mimosa (Inria Sophia) and CMOS (LaMI Évry). The objective is to study an integrated development environment for the construction and use of safe embedded components.

8.1.2. 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.

8.1.3. 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).

8.1.4. ACI Sécurité Informatique ROSSIGNOL

Roberto Amadio is 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. ARA Sécurité, Systèmes embarqués et Intelligence Ambiante, Cops

Silvano Dal Zilio is participating in this action that has started in September 2005. The other teams are IRIT (coordinator), MoVe from the LIF, and the LORIA projects CASSIS and ECOO.

8.1.6. ACI Masses de données TraLaLA

Silvano Dal Zilio is participating in this action. The other teams are the LIENS ENS Paris (coordinator), LRI Orsay, INRIA Futurs Projet GEMO, and LIFL and INRIA Futurs projet MOSTRARE.

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. This project ended in April 2005.

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. This project ended in April 2005.

9. Dissemination

9.1. Seminars and conferences

Roberto Amadio presented [25] at the meeting on Algebraic Process Calculi in Bertinoro. He participated in the workshops of the CRISS project, giving a talk on [24], and in the CONCUR Conference and the EXPRESS workshop.

Gérard Boudol in January, Gérard Boudol participated in a CRISS workshop in Marseille, where he gave a talk on [16], and in a meeting of the MIKADO project in Torino. He participated in the Dagstuhl Seminar on the Foundations of Global Computing in February, and gave a talk on [16]. He participated in the final review of the MIKADO project in Edinburgh, giving a talk on the research activities in the area of models in this project. He attended the ETAPS conference held in the same period (April). He gave a three hours course on Language-Based Security at the CIRM School on Security (Marseille, April). He gave an invited seminar on [16] at the DSL Workshop on Functional Programming and Verification (Nancy, May). He participated in a CRISS workshop in Paris (June), where he gave a talk on [19]. He participated in the 18th IEEE Computer Security Foundations Workshop (Aix-en-Provence, June), where the work [16] was presented by Ana Matos. He participated in a meeting on Algebraic Process Calculi in Bertinoro (August), where [25] was presented by Roberto Amadio. He attended the International Colloquium on Theoretical Aspects of Computing in Hanoi (October), where he presented the paper [19].

Ilaria Castellani participated to the 18th IEEE Computer Security Foundations Workshop (Aix-en-Provence, June 20-22, 2005) where she gave a short talk, and to the workshop *Algebraic Process Calculi: The First Twenty Five Years and Beyond* (Bertinoro, Italy, August 1-5, 2005), where the paper [25] was presented by Roberto Amadio. She took part in a meeting of the ACI project CRISS in Marseille (January 17-18, 2005). She visited the University of Aarhus (June 30, 2005) to attend Marco Carbone's PhD thesis defense.

Frédéric Dabrowski gave a talk on [17] at the LACL (University of Paris 12). He presented also this paper at the EXPRESS Workshop, and attended the CONCUR Conference. He participated in the workshops of the CRISS project.

Silvano Dal Zilio gave a talk at NWPT'05, the 17th nordic workshop on programming theory [14], on a typed process calculus for querying distributed XML documents. He gave a talk on resource control for functional programs at CSL'04, the 18th International Conference on Computer Science Logic. A subsequent result [20], [27] that extend our work on resource control to the case of tail-recursive programs was presented at APLAS'05, the 3rd Asian Symposium on Programming Languages and Systems. Silvano Dal Zilio gave several talks on meetings of the ACI Masses de Données Tralala in Paris, Marseille and Lille.

Manuel Serrano gave a talk on *Ubiquitous Mail* during the 6th Scheme and Functional Programming Workshop that took place in Tallinn.

9.2. Animation

Roberto Amadio was the leader of the MOVE team (Modélisation et Vérification) of the Laboratoire d'Informatique Fondamentale of Marseilles (UMR-CNRS 6166) till the end of August. He was general Chair of the IEEE Computer Security Foundations Workshop (Aix en Provence), June 2005. He was a member of the programme committee of the following workshops and conferences: EXPRESS 2005, GEOCAL 2006, FOSSACS 2006, ICALP 2006, CONCUR 2006. He is co-chair of the programme committee of EXPRESS 2006. He is a member of the steering commitee of CONCUR, the Computer Security Foundations Workshop, and the Ecole de printemps d'Informatique Théorique. He was a referee in the HDR (Habilitation à Diriger des Recherches) of Jean-Marc Talbot (University of Lille 1).

Gérard Boudol acted as a referee for the PhD Theses of Daniele Gorla (University of Torino) and Fransisco Martins (University of Lisboa). He was the chairman of the jury in the defence of the PhD Thesis of Philippe Bidinger (University of Grenoble), and member of the jury of the PhD Thesis of Benjamin Leperchey (Univeristy of Paris 7). He was a member of the Program Committee of the FOSSACS'06 conference.

Frédéric Boussinot serves a reviewer of the thesis of Anne-Gwenn Bosser, PPS, University Paris 7. He was also examinator of the thesis of Philippe Bidinger, University of Grenoble. He is a member of the program committee for MSR'05 and SLAP'05. He is member of the editorial board of the revue TSI.

Ilaria Castellani acted as a referee for the PhD theses of Mauro Gattari (Dept. of Mathematics and Computer Science, University of Siena, January 2005) and Marco Carbone (BRICS, University of Aarhus, July 2005).

Silvano Dal Zilio was local organizer for the 18th IEEE Computer Security Foundations Workshop (see http://www.lif.univ-mrs.fr/CSFW18/) and the 2005 Spring School on Computer Seccurity – Marseille, April 25-29 (see http://www.lif.univ-mrs.fr/~secur05/). He is local coordinator for the ACI Masses de Donnees projet Tralala.

Manuel Serrano was a referee for the PhD thesis of Daniel Bonniot (Ecole des Mines and INRIA project CRISTAL). He was a member of the program committee for the Ecoop European Worskhop on Lisp and Scheme. He was a member of the ICFP'05 program committee. He is a member of the "SCHEME Strategy Group" that decides on the evolutions of this programming language. He is one of seven authors of the Revised 6 Report on the Scheme programming language. He participated to the annual meeting that took place this year in Boston.

9.3. Teaching

Roberto Amadio was responsible of the Master Informatique Fondamentale of the University of Provence. He is teaching on concurrency in the Master Parisien de Recherche en Informatique. In the Master d'Ingénierie Informatique (University of Provence), he was teaching on security, and on logical tools, and syntactic analysis and compilation in the Licence d'Informatique.

Frédéric Boussinot gave lectures at the "Master d'Informatique" of the University of Nice. He supervised the internship of Celio Troi (Master of the University of Nice).

Frédéric Dabrowski gave a course on "Introduction au parallélisme" at the Licence Professionnelle Systèmes Informatiques et Logiciels (LPSIL) of the university of Nice and a lecture at the "Master d'Informatique" of the university of Nice.

Silvano Dal Zilio gave a course in the Master Professionnel Informatique of Marseille universities on "XML: tools and documents" (see http://www.cmi.univ-mrs.fr/~dalzilio/MasterI2A/). He supervised the internship of Yann Barsamian (ENS Lyon) on the implementation of a core functional programming language for manipulating XML documents.

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

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", vol. 96, 1992.

[3] G. BOUDOL. *The π-calculus in direct style*, in "Higher-Order and Symbolic Computation", vol. 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", vol. 30, 2000.

[7] I. CASTELLANI. *Process Algebras with Localities*, in "Handbook of Process Algebra, Amsterdam", J. BERGSTRA, A. PONSE, S. SMOLKA (editors). , North-Holland, 2001, p. 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", vol. 10, n^o 2, May 2000, p. 1–43.

Articles in refereed journals and book chapters

[10] R. AMADIO. *Synthesis of max-plus quasi-interpretations*, in "Fundamenta Informaticae", vol. 65, 2005, p. 29-60.

[11] G. BOUDOL, P. ZIMMER. *On type inference in the intersection type discipline*, in "Electronic Notes in Theoretical Computer Science", vol. 136, 2005, p. 23-42, http://www.inria.fr/mimosa/Gerard.Boudol/otiititd.html.

[12] F. BOUSSINOT. *FairThreads: mixing cooperative and preemptive threads in C*, in "Concurrency and Computation: Practice Experience", September 2005.

[13] E. GALLESIO, M. SERRANO. *Skribe: a Functional Authoring Language*, in "Journal of Functional Programming", 2005, http://www.inria.fr/mimosa/Manuel.Serrano/publi/jfp05/article.html.

Publications in Conferences and Workshops

[14] L. ACCIAI, M. BOREALE, S. DAL ZILIO. *A Typed Calculus for Querying Distributed XML Documents*, in "NWPT 2005 – 17th Nordic Workshop on Programming Theory", Oct 2005.

[15] A. ALMEIDA MATOS. *Non-disclosure for distributed mobile code*, in "FST-TCS'05", Lecture Notes in Computer Science, 2005.

[16] A. ALMEIDA MATOS, G. BOUDOL. *On declassification and the non-disclosure policy*, in "Computer Security Foundation Workshop", 2005, p. 226-240, http://www.inria.fr/mimosa/Gerard.Boudol/non-discl.html.

[17] R. AMADIO, F. DABROWSKY. *Feasible Reactivity for Synchronous Cooperative Threads*, in "12th workshop Expressiveness in Concurrency, San Francisco, USA", Electronic Notes in Theoretical Computer Science, Aug 2005.

[18] G. BOUDOL. *A generic membrane model*, in "Second Global Computing Workshop", Lecture Notes in Computer Science, vol. 3267, 2005, p. 209-222, http://www.inria.fr/mimosa/Gerard.Boudol/gmm.html.

[19] G. BOUDOL. *On typing information flow*, in "International Colloquium on Theoretical Aspects of Computing", Lecture Notes in Computer Science, vol. 3722, 2005, p. 366-380, http://www.inria.fr/mimosa/Gerard.Boudol/otif.html.

[20] S. DAL ZILIO, R. GASCON. *Resource Bound Certification for a Tail-Recursive Virtual Machine*, in "APLAS 2005 – 3rd Asian Symposium on Programming Languages and Systems", Lecture Notes in Computer Science, vol. 3780, Springer-Verlag, Nov 2005, p. 247–263.

[21] E. GALLESIO, M. SERRANO. *Ubiquitous Mail*, in "Proceedings of the Sixth Workshop on Scheme and Function Programming, Tallinn, Estonia", Sep 2005, p. 69–76, http://www.inria.fr/mimosa/Manuel.Serrano/publi/sfp05/article.html.

[22] F. LOITSCH. *Javascript Compilation*, in "Proceedings of the Sixth Workshop on Scheme and Function Programming, Tallinn, Estonia", Sep 2005, p. 101-111.

Internal Reports

[23] A. ALMEIDA MATOS, G. BOUDOL, I. CASTELLANI. *Typing Noninterference for Reactive Programs*, Submitted for publication., Report, n° 5594, INRIA, 2005, http://www.inria.fr/rrrt/rr-5594.html.

[24] R. AMADIO. *The SL synchronous language, revisited*, Technical report, Laboratoire PPS, Université de Paris 7, 2005.

[25] R. AMADIO, G. BOUDOL, F. BOUSSINOT, I. CASTELLANI. *Reactive concurrent programming revisited*, Technical report, n° NS-05-3, BRICS Notes Series, 2005, http://www-sop.inria.fr/mimosa/personnel/Ilaria.Castellani/abbc-APC25-abstract.html.

[26] F. BOUSSINOT. *Cyclone+Loft*, Technical report, September 2005.

[27] S. DAL ZILIO, R. GASCON. *Resource Bound Certification for a Tail-Recursive Virtual Machine*, Technical report, n° 26, LIF, Jun 2005.

Miscellaneous

[28] D. CIABRINI. *Stack virtualization for source level debugging*, 2005.

[29] F. DABROWSKY, F. BOUSSINOT. *Cooperative Threads and Preemptive Computations*, submitted to ESOP'2006, 2005.