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 Scientist, Inria]

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

Visting scientist
Erick Gallesio [Visiting Scientist, University of Nice Sophia-Antipolis]

Ph. D. students
Lucia Acciai [MENRT]
Christian Brunette [MENRT]
Damien Ciabrini [MENRT, till January 1]
Frederic Dabrowski [MENRT]
Stephane Epardaud [MENRT]
Ana Matos [Portuguese Gov., till January 1]
Charles Meyssonnier [ENS Lyon]
Vincent Vanackère [ENS Lyon]
Pascal Zimmer [ENS Lyon]

Internship
Florian Loitsch [ESSI]

Post-doctoral fellow
Xudong Guan [Inria]

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 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. 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 (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, 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. 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.

5.2.3. 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. Functional programming

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

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.

We have distributed three major releases of Bigloo during 2004: versions 2.6c, 2.6d, and 2.6e.

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. 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 [11].

5.3.4. 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.

5.3.5. 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 in [19]. A reference manual (in french) can be found on the web page of the project, and is a chapter of Zimmer's thesis [11].

5.3.6. 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. 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 [**?bib ALV02**]). 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.

The work on the tree logic, showing the decidability of the model-checking and satisfiability problems, that we reported upon last year has been published at the POPL conference [22].
Elaborating on a deliverable of the MIKADO project, we have introduced a "membrane calculus", to deal with the problem of controlling the movement of computing entities with respect to the domains in which they run. The main idea of our model 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 into the domain, or the way in which they are allowed to exit from it. As regards the processes running in a domain, our calculus is generic, in the sense that it may be combined with a variety of programming models for processes, provided that such a model supports the dynamic creation of processes, and the sending of messages to the controlling membrane. In [17] we illustrate, by means of some examples, the expressive power of our model.

6.2. Security

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

6.2.1. Non-interference in reactive systems

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.
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. In particular, 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. We then have proposed in [24] a type system to enforce non-interference in a core reactive language. In this paper we establish the soundness of the type system with respect to the security property, which is formulated in terms of bisimulation.
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 started to work on 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. We address the second question by introducing in the programming language a construct to deal with local flow policies. Our preliminary results (not yet published) show that the classical notion of non-interference can be generalized, to deal with a dynamically evolving structure of the lattice of security levels, and that a type and effect system for an expressive core language, including a declassification construct, can be designed to ensure this generalized security property.

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.
The work by Roberto Amadio on quasi-interpretations which is presented in the report of last year is now accepted for publication in a journal [27]. A research direction that we are now exploring is the extension of this technique to the reactive code. In this setting, the problem is to ensure the termination of each "instant", and to

certify that the computations occurring during each instant are not too complex. This is the PhD programme of Frédéric Dabrowski. In [31], he addressed the first problem, showing that standard techniques used to establish termination of recursive programs can be extended and adapted to a first-order subset of the language ULM [18].
Amadio and Dal Zilio addressed the second question in [16]. There they develop new methods to statically bound the resources needed for the execution of systems of concurrent, interactive threads, that use the reactive model as regards scheduling. Their contribution is a system of compositional static analyses to guarantee that each instant terminates and to bound the size of the values computed by the system as a function of the size of its parameters at the beginning of the instant. The technique that is used generalises an approach designed for first-order functional languages that relies on a combination of standard termination techniques for term rewriting systems and an analysis of the size of the computed values based on the notion of quasi-interpretation. They show that these two methods can be combined to obtain an explicit polynomial bound on the space needed for the execution of the system during an instant. They also study the expressivity of their reactive programming model, and describe a bytecode for a related virtual machine.
In [32] Dabrowski also adresses the question of bounding the complexity of programs, focusing on the extension of the classical framework of recursive definitions over inductive data types to a core language with imperative features. Dabrowski first introduces a direct characterization of programs computing in polynomial space by means of explicit polynomial bounds that provide a stratification of programs, in the style of Bellantoni and Cook's approach. He then proves that this stratification leads to programs that can be run in space bound by a polynomial of the size of their inputs. Finally, he shows that this stratification of programs can be achieved by a type and effect system.
Finally, in [16], Amadio et al. study the complexity of programs from the point of view of a low level semantics, close to an actual implementation. The purpose is to check that a program which is of a given complexity class from the point of view of an abstract semantics indeed executes within the specified space and time bounds. To this end, they define a simple stack machine for a first-order functional language and show how to perform type, size, and termination verifications at the level of the bytecode of the machine. In particular, they show that a combination of size verification based on quasi-interpretations and of termination verification based on lexicographic path orders leads to an explicit bound on the space required for the execution.

6.3. Reactive programming

Participants: Frédéric Boussinot, Christian Brunette, Stéphane Epardaud, Manuel Serrano.

6.3.1. Reactive Programming of Cellular Automata

Cellular automata (CA) are used in various simulation contexts, for example, physical simulations, fire propagation, or artificial life. These simulations basically consider large numbers of small-sized identical components, called *cells*, with local interactions and a global synchronized evolution.
Using reactive programming for implementing CA has the following advantages:

- Modularity of programming. The behaviors of cells is rather opaque in usual CA implementations. This is however generally not felt as a big issue because cells behaviors are often very simple. In some contexts, for example artificial life, one may ask for more complex cell behaviors. In these cases, modularity is certainly a plus.
- Multiprocessing. Sequential programming of CA makes difficult the use of several processors because cells must be protected from concurrent accesses of their neighbors and because the global synchronization of cells has to be preserved. The proposed concurrent implementation of CA is compatible with multiprocessing.

One gets an efficient implementation in which cells are fair threads implemented as automata and created when needed. Cell behaviors are coded at a higher level than the one of look-up tables. The example of self-replicating loops is considered, following the work of Langton and Sayama in artificial life. A technical report [29] and the implementation are available at `http://www-sop.inria.fr/mimosa/rp/CellularAutomata`. For more details, see http://www-sop.inria.fr/mimosa/rp/CellularAutomata.

6.3.2. Reactive Programming: 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. 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 makes 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 real parallelism, on a distinct processor. Also, distinct schedulers can be run by distinct processors.
Current version of FairThreads in C is v1.2. See also: http://www-sop.inria.fr/mimosa/rp/FairThreads.

6.3.3. Reactive Programming: Icobjs

Icobj Programming is a simple and fully graphical programming method, using a powerful means to combine behaviors. This programming is based on the notion of an *icobj* which has a behavioral aspect (object part), a graphical aspect (icon part), and which can be animated at screen.
Icobj Programming comes from the reactive approach and provides parallelism, broadcast event communication, and migration through the network.
A new Icobjs system in Java is now available. It unifies icobjs and workspaces in which icobjs are created, and uses a specialized reactive engine named Reflex. Simulations in Physics and the Ambient calculus have been ported on this new system. The work on Icobjs is described in Christian Brunette's thesis [10]. The semantics of the reactive engine Reflex, including some optimisations, is also described in the thesis. See http://www-sop.inria.fr/mimosa/rp/Icobjs for additional information.

6.3.4. Reactive Programming: 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*). 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 for parallelism provided by SMP multiprocessor machines.

A first implementation of LOFT consists in a rather direct translation in 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 do not use Pthreads 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 implemented by schedulers.
LOFT has been used for implementing cellular automata spaces, in particular those devoted to artificial life simulations.
Ways to synchronize several independent schedulers and non deterministic schedulers have been introduced in the most recent version of LOFT.
Information and implementation of LOFT are available at http://www-sop.inria.fr/mimosa/rp/LOFT along with a draft book [30].

6.4. Functional programming

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

6.4.1. Types and recursion

The work on type inference in the intersection type discipline for the λ-calculus has been finalized. Let us recall that intersection type systems provide more complex and more permissive systems than the classical Hindley-Milner's polymorphic types, for which an algorithm inferring principal types exists. Type inference has also been studied regarding intersection types, and various inference algorithms have been proposed by Coppo, Dezani, Venneri, Ronchi della Rocca, and more recently by Kfoury and Wells. They are however quite complex, and therefore difficult to understand and prove formally. We have designed in [19] 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 software implementing this algorithm; it helped us to check and experiment quickly our ideas during the design process. Most of the results obtained in the past few years on the topic of types and recursion can be found in Zimmer's thesis [11]. Moreover, the work of Boudol mentioned last year regarding the static analysis of Dreyer's fixpoint has appeared as a technical report [28].

6.4.2. Bigloo

For the year 2004, our effort on Bigloo have focused on two different areas. First, programming and debugging concurrent applications in Scheme (see [25]). Second, the efficient production of .NET bytecode.

6.4.2.1. Debugging Scheme FairThreads

There are two main policies for scheduling thread-based concurrent programs: preemptive scheduling and cooperative scheduling. The former is known to be difficult to debug, because it is usually non-deterministic and can lead to data races or difficult thread synchronization. We believe the latter is a better model when it comes to debugging programs.
We have extended BUGLOO to support the debugging of Scheme Fair Threads [25], that are based on cooperative scheduling and synchronous reactive programming. In this approach, thread communication and synchronization is achieved by means of special primitives called signals.
We showed that unlike the classic POSIX multi-threading approach, Fair Threads allow to provide the programmer with a strong debugging support. We have described three tools to deal with the type of concurrent bugs that can arise in the Fair Threads paradigm. First, an improved single-stepper. Second, a scheduler and

signal inspector to analyze the state the threads when the program is suspended. At last, a scheduler tracer to analyze the progression of the scheduling off-line.

6.4.2.2. .NET Bytecode generation

Introduced by Microsoft in 2001, the .NET framework has many similarities with the Sun Microsystems Java Platform. The execution engine, the Common Language Runtime (CLR), is a stack-based Virtual Machine (VM) which executes a portable bytecode: the Common Intermediate Language (CIL). The CLR enforces type safety through its bytecode verifier (BCV), it supports polymorphism, the memory is garbage collected and the bytecode is Just-In-Time compiled to native code.
Beyond these similarities, Microsoft has designed the CLR with language agnosticism in mind. Indeed, the CLR supports more language constructs than the JVM: the CLR supports enumerated types, structures and value types, contiguous multidimensional arrays, etc. The CLR supports tail calls, i.e. calls that do not consume stack space. The CLR supports closures through delegates. At last, pointers to functions can be used although doing so leads to unverifiable bytecode. The .NET framework has 4 publicly available implementations:

- From Microsoft, one commercial version and one whose sources are published under a shared source License: Rotor. Rotor was released for research and educational purposes.
- From DotGNU, the Portable.Net GPL project provides a quite complete runtime and many compilation tools. Unfortunately, it does not provide a full-fledged JIT.
- From Novell, the Mono Open Source project offers a quite complete runtime and good performances. In term of performance, Mono is the best implementation of .NET that runs on both Windows and Linux.

As for the JVM, the .NET framework is appealing for language implementors. The runtime offers a large set of libraries, the execution engine provides a lot of services and the produced binaries are expected to run on a wide range of platforms. Moreover, we wanted to explore what the “more language-agnostic” promise can really bring to functional language implementations as well as the possibilities for language interoperability.
This year, we have polished the Bigloo .NET back-end. Bigloo now delivers applications whose performance are similar to the C# applications compiled with regular compilers. It appears that in the current state of the art of the .NET implementations, most of the new functionalities of the .NET framework are still disappointing if we only consider performance as the ultimate objective. On the other hand, the support for tail calls in the CLR is very appealing for implementing languages that require proper tail-recursion. .NET performance is improving form version to version. Bigloo.NET programs still run significantly slower on the Mono implementation than programs compiled to native code of compiled to JVM bytecode. The same Bigloo.NET programs runs better on the Microsoft’s CLR. This work is described in the paper [13].

6.4.3. Skribe

During the year 2004, the necessity for a redesign of Skribe as been raised. This has been done this year. If the surface syntax has merely changed the internal of Skribe has been totally redesigned, more precisely, the programming language it relies on has been redesigned.
That is, in addition to being a markup language, Skribe is also a functional programming language. We have chosen to base Skribe on Scheme mainly because the Scheme syntax is genuinely close to traditional markup languages. Like XML, Scheme syntax is based on the representation of trees. Skribe’s modifications of the Scheme grammar are limited and simple, and merely improve Scheme to make it more suitable for representing literal text.
Like Scheme, Skribe relies on dynamic type checking. Unlike XML, it is not designed for supporting static verification of documents. The flexibility of dynamic types eases the production of dynamic texts. In particular, an Sk-expression can be a list whose elements are of different types. The first element of such a list could be a character string, and the next one a number. This generality enables compact representation of texts.

Skribe documents are programs. Executing each program produces a target document, such as HTML pages or PostScript texts. To avoid the confusion between the source document and the output documents, we refer to the former as source programs or programs, and we refer to the latter as output documents or documents.
Functions are ubiquitous in Skribe. They are the only means for defining the syntactic Skribe elements. For instance, functions are used to define convenience macros like the ones supported by most typesetting systems. In its simplest form, a macro is just a name that is expanded into, or replaced with, a text that is part of the produced document. Macros are implemented in Skribe by the means of functions that produce Sk-expressions. In addition to naturally implemeting markups and macros, Skribe functions have another strength: they expose a single formalism for specifying the static, (i.e., declared as-is) and dymamic, (i.e., computed) parts of the documents. This combination is the main originality and the main strength of Skribe in which it is particularly easy to merge texts and computations for authoring documents. Moreover, because Skribe is a full-fledged programming language, arbitrarily complex computations can be easily programmed.
The most significant achievement of this year has been the redesign of the Skribe evaluator. The new Skribe evaluator relies on three stages: The first stage parses the source program and builds an abstract syntax tree. This stage roughly corresponds to a Scheme evaluator augmented with additional library functions. The second stage is in charge of resolving self-references to the document itself. The third stage (Write) is in charge of producing the final output document. This last stage relies on a new programming style named *engine passing style* which is inspired from the *continuation passing style*. It is described in a paper currently submitted.
In addition to improving the language, we have started the development of an *integrated programming environment* (IDE) for Skribe. We have created an Eclipse-Plugin allowing to write Skribe-documents within Eclipse. Eclipse (http://www.eclipse.org) is a well known IDE initially developed by IBM and now available under the GPL. Written in Java some of its primary purposes were platform-independance and extensibility. As such it encourages independent developers to create their enhancements in the form of plugins. Skribe is basically an enhancement to the well-known programming language Scheme, so it becomes a powerful document markup-language that easily beats its competitors (HTML, Latex and others) in terms of flexibility. Even though Eclipse itself is written in Java, and expects its plugins to be written in Java too, we have decided to write large parts of the plugin in Scheme, and to compile the resultant code to Java-Bytecode using Bigloo. This makes sense, as Scheme is a subset of Skribe, and both (Bigloo and Skribe) are developed by the same person (Manuel Serrano). Bigloo's Java-Scheme interface allows to use Java code in Scheme and vice-versa. It is however not possible to extend Java-classes or implement Java-interfaces within Scheme. A small intermediate layer has hence been written that bridges the Eclipse-framework with the Skribe-plugin. In order to ease code-reuse, this layer and big parts of the plugin have been separated from the Skribe-specific code, so that other Eclipse plugins can be created within Scheme.
The developed plugin now provides syntax-highlighting, content assist, templates, automatic compilation and many other features for the Skribe language.

6.5. Combining Reactive and Functional programming

Participants: Gérard Boudol, Frédéric Boussinot, Stéphane Epardaud, Manuel Serrano.

6.5.1. Reactive and mobile programming: Ulm

Last year we started 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, etc. We have designed a programming model, called ULM (for "*Un Langage pour la Mobilité*"), 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. This has been published in a conference [18], and a journal paper on this topic is in preparation.

ULM has been implemented in the form of an embedding in Scheme. For this Scheme augmented with ULM primitives, we provide a compiler and virtual machine written in Bigloo. This work has been presented at the Scheme Workshop 2004 [23] and is available at http://www-sop.inria.fr/mimosa/Stephane.Epardaud/ulm.
All ULM and Scheme primitives have been implemented and tested, including agent mobility to different virtual machines on possibly different hosts. However, this implementation is a prototype and virtually no attention has been spent on speed, security or a distributed Garbage Collector such as is needed in ULM.
In an attempt to improve the speed of the reactive scheduling, we have implemented LURC (Light ULM Reactive for C), which is a highly optimized library in C implementing the reactive primitives of ULM (that is, all of ULM except mobility). This library has been tested and used as an alternative for LOFT.
We have ported the reactive scheduling techniques of LURC in our Scheme embedding of ULM, and are currently investigating whether the mobility part of ULM combines well with highly optimized scheduling.
We have also introduced the concept of Reactive Event Loop in our implementation of ULM, by providing primitives that map common events of Event Loop programming (such as polling, idle functions and timeouts) in a reactive fashion. We have had success implementing traditional Event Loop example programs in our Reactive Event Loop model, and noticed how our model allows for simpler and smaller code by providing state to event handlers.
We have extended ULM to allow for native calls from the ULM Scheme embedding to either Bigloo or C layers. These native calls can also call ULM functions, thus reentering in the Virtual Machine. This implies that ULM threads can have some of their heap in a native form and cannot migrate. In order for ULM threads to call native functions, we used Bigloo FairThreads to delegate the calls while keeping the Virtual Machine heap clean. The technique used for native heap delegating is based on FairThread threads and signals and could be reused by any other Virtual Machine that supports custom threads in order to allow VM reentry.

7. Other Grants and Activities

7.1. National initiatives

7.1.1. 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.

7.1.2. 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.

7.1.3. 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.

7.1.4. 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).

7.1.5. 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.

7.2. European initiatives

7.2.1. 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.

7.2.2. 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. Dissemination

8.1. Seminars and conferences

Roberto Amadio gave an invited talk during the International Workshop on Security Analysis of Systems on *Towards control of resources for synchronous systems* that took place in Orléans.

Gérard Boudol organized the first meeting of the CRISS project, in Sophia Antipolis. He participated in a meeting of the MIKADO project in Grenoble, where he gave a talk on [18]. He gave an invited talk on the same topic at the COORDINATION Conference in Pisa. He participated in the Global Computing Workshop in Rovereto, where he gave a talk on [17], and in the ESOP'04 Conference in Barcelona, where he gave a talk on [18]. He gave a talk on the same topic at the meeting of the GEOCAL project in Lyon, and he participated of the joint workshop of the MIKADO, DART and MYTHS projects of the European Global Computing Initiative in Venice. He participated in the conferences ICALP and LICS in Turku, and in the satellite workshops ITRS and FCS, where the papers [**?bib BoudolZimmer:typ-inf,BoudolCastellaniMatos:non-interf-reactive**] were presented (by Pascal Zimmer and Ana Matos respectively).

Frédéric Dabrowski gave a talk on [31] at the second meeting of the CRISS project in Nancy. He participated in the CRISS-GEOCAL workshop in Villetaneuse, where he gave a talk on [32].

Frédéric Boussinot gave a talk at the first meeting of the ALIDECS project in Paris.

Ilaria Castellani Ilaria Castellani participated to the conferences ICALP'04 and LICS'04 (Turku, Finland, July 12-16, 2004) and to the affiliated workshop on Foundations of Computer Security FCS'04. She was an active member of the Comité Local des Etats Généraux de la Recherche (for Nice Sophia Antipolis), inside which she animated a working group, and took part in the national meeting that concluded the Etats Géneraux (Assises de Grenoble, October 28-29, 2004). She taught a course on Calculi for Mobile Processes in the DEA of Mathematics of Nice University (12 hours, January-February 2004).

Damien Ciabrini gave a talk at the fifth Scheme Workshop located in Snowbird, Utah. He also gave a presentation of his work about the debugging of *Scheme Fair Threads*.

Silvano Dal-Zilio gave a talk on new complexity results for the ambient logic at POPL'04 [22], the 31st Symposium on Principles of Programming Languages. He gave a talk on resource control for functional programs at SPACE'04, the 2nd workshop on semantics, program analysis, and computing environments for memory management, co-located with POPL. He gave a talk on resource control for reactive programs [16] at CONCUR'04, the 15th International Conference on Concurrency Theory. He gave a talk on resource control for functional programs [15] at CSL'04, the 18th International Conference on Computer Science Logic. Silvano Dal Zilio gave a talk on a meeting of the ACI Sécurité Informatique CRISS in Nancy in Rennes.

Stéphane Epardaud Stéphane Épardaud gave a presentation of his work at the CRISS meeting in Nancy, the 23rd of June. He also published a paper and gave a talk about the ULM embedding in Scheme at Snowbird (USA) on the 22nd of September for the Scheme Workshop 2004.

Ana Matos took part in the first meeting of the CRISS project in INRIA Sophia-Antipolis, where she gave a talk on typing non-interference for reactive programs. She participated in LICS'04 and ICALP'04 in Turku, as well as in the FCS'04 workshop where she gave a talk on [24]. She attended PLID'04 in Verona. She spent one month at the Universiy of Chalmers working in the ProSec group, where she gave a talk on typing non-interference for reactive programs.

Manuel Serrano visited the IBM Watson laboratory where gave a talk on Skribe. He gave another talk on Skribe in the Gemo team in Inria-Futurs. Manuel Serrano and Erick Gallesio gave two tutorial of Skribe. The first one for opening the *Cafés logiciels* (http://www-sop.inria.fr/intech/cafes/skribe.html) seminar that took place at Sophia. The second one during the *Solutions Linux* (http://www.solutionslinux.fr/fr/index.php) event that took place in Paris. Manuel Serrano participated in a four day meeting in Snowbird (Utah) about the elaboration of the next specification of the Scheme programming language. He gave a talk on the Scheme FairThreads at the PPDP conference in Verona, Italia. He gave a talk on the compilation of Scheme to .NET in the second .NET workshop in Plzen, Czech Republic.

Pascal Zimmer attended the ETAPS'04 conference in Barcelona, and the ICALP-LICS'04 conference in Turku. He gave a talk at the ITRS'04 workshop on [19]. He also gave seminars on the topics of his thesis [11] at INRIA Futurs (seminar of the Comete project), the University of Chambery and the BRICS Laboratory in Aarhus, Denmark.

8.2. Animation

Roberto Amadio is the leading scientist of the *Modélisation and Vérification* (http://www.cmi.univ-mrs.fr/~amadio/ModVer/modver.html) (LIF, UMR-CNRS 6166). He was the invited editor of the special issue of *Journal of Logic and Algebraic Programming* on *Modelling and verification of cryptographic protocols*. He was a member of the program committee of the *3rd IFIP Conference on Theoretical Computer Science* and the *EXPRESS 2004* conference. He is member of the steering committee of the Concurrency Theory conference (CONCUR).

Gérard Boudol was a referee for the PhD thesis of Vincent Simonet (University of Paris 7 and INRIA project CRISTAL). He was a member of the programme committee of the workshops Intersection Types and Related Systems (ITRS'04), a satellite event of ICALP-LICS, and Foundations of Global and Ubiquitous Computing, a satellite event of CONCUR. He was also a member of the programme committee of the conference Foundations of Software Technology and Theoretical Computer Science (FST-TCS'04).

Ilaria Castellani Elected member of the Comité de Centre (October 2001 - October 2004) and of the Comité Technique Paritaire (March 2003 - March 2006). Member of the Comité des Projets, as researcher representative (March 2003 - October 2004).

Silvano Dal-Zilio Silvano Dal-Zilio was a member of the programme committee for the workshop WOOD'04 – 2nd Workshop on Object-Oriented Developments – co-located with CONCUR. He is local coordinator of the ACI *Masses de Données* TRALALA.

Manuel Serrano was co-editor of two special issues of the journal "Higher-Order and Symbolic Computation" about the programming language SCHEME. He was a member of the ICFP'04 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.

Pascal Zimmer is moderator of the Mobile Calculi electronic forum.

8.3. Teaching

Roberto Amadio is charge of the Master of *Informatique Fondamentale* of the University of Provence. He supervised the internship of J. Radhakrishnan who is coming from the master of the Birla Institute of Technology and Science (India). He supervised the PhD thesis of Vincent Vanackere (with D. Lugiez), Charles Meyssonnier (with S. Dal Zilio), Frederic Dabrowski (with G. Boudol), and Mathieu Agopian (avec P. Niebert). He was a referee for the HDR Thesis of S. Coupet-Grimal. He teaches formal languages and theory at a graduate level.

Frédéric Boussinot gave lectures at the "DEA d'Informatique" of the University of Nice. He supervised the internship Stéphane Epardaud (DEA University of Nice).

Silvano Dal-Zilio Silvano Dal Zilio gave a course at the DEA Informatique of Marseille universities, on types for program verification. He supervised the internship of Pierre-Paul Tacher (Supelec Metz) on the implementation of a core functional programming language for manipulating XML documents. He also supervised the DEA internship of Regis Gascon on resource control at byte-code level for optimized programs.

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

Pascal Zimmer teaches functional programming (exercise groups, DEUG MI2), and programming in C and relevant tools (exercise groups, Licence Informatique), at the University of Nice Sophia Antipolis.

9. 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.

Doctoral dissertations and Habilitation theses

[10] C. BRUNETTE. *Construction et simulation graphiques de comportements: le modèle des Icobjs*, Thèse de doctorat, Université de Nice-Sophia Antipolis, Oct 2004.

[11] P. ZIMMER. *Récursion généralisée et inférence de types avec intersection*, Thèse de doctorat, Université de Nice Sophia Antipolis, France. INRIA Sophia Antipolis, 2004, http://www-sop.inria.fr/mimosa/Pascal.Zimmer/papers/these.ps.

Articles in referred journals and book chapters

[12] G. BOUDOL. *The Recursive Record Semantics of Objects Revisited*, in "Journal of Functional Programming", vol. 14, 2004, p. 263-315, http://www.inria.fr/mimosa/Gerard.Boudol/safe-rec-obj.html.

[13] Y. BRES, B. SERPETTE, M. SERRANO. *Bigloo.NET: compiling Scheme to .NET CLR*, in "Journal of Object Technology", vol. 3, n^o 9, Oct 2004, http://www.inria.fr/mimosa/Manuel.Serrano/publi/jot04/jot04.html.

[14] D. TELLER, P. ZIMMER, D. HIRSCHKOFF. *Using ambients to control resources*, in "International Journal of Information Security", vol. 2, 2004, p. 126–144, http://www-sop.inria.fr/mimosa/Pascal.Zimmer/papers/ijis_resources.pdf.

Publications in Conferences and Workshops

[15] R. M. AMADIO, S. COUPET-GRIMAL, S. DAL ZILIO, L. JAKUBIEC. *A Functional Scenario for Bytecode Verification of Resource Bounds*, in "CSL 2004 – 18th International Conference on Computer Science Logic", Lecture Notes in Computer Science, vol. 3210, Springer-Verlag, 2004, p. 265–279.

[16] R. M. AMADIO, S. DAL ZILIO. *Resource Control for Synchronous Cooperative Threads*, in "CONCUR 2004 – 15th International Conference on Concurrency Theory", Lecture Notes in Computer Science, vol. 3170, Springer-Verlag, 2004, p. 68–82.

[17] G. BOUDOL. *A generic membrane model*, in "Second Global Computing Workshop", To appear in Lecture Notes in Computer Science, 2004, http://www.inria.fr/mimosa/Gerard.Boudol/gmm.html.

[18] G. BOUDOL. *ULM, a core programming model for global computing*, in "ESOP'04", Lecture Notes in Computer Science, vol. 2986, 2004, p. 234-248, http://www.inria.fr/mimosa/Gerard.Boudol/ulm-v0.html.

[19] G. BOUDOL, P. ZIMMER. *On type inference in the intersection type discipline*, in "ITRS'04 Workshop", to appear in Electronic Notes in Computer Science, 2004, http://www.inria.fr/mimosa/Gerard.Boudol/otiititd.html.

[20] Y. BRES, B. SERPETTE, M. SERRANO. *Compiling Scheme programs to .NET Common Intermediate Language*, in "2nd International Workshop on .NET Technologies, Plzen, Czech Republic", (taux d'acceptation 11/31), May 2004, http://www.inria.fr/mimosa/Manuel.Serrano/publi/bss-dotnet04.pdf.

[21] D. CIABRINI. *Debugging Scheme Fair Threads*, in "Proceedings of the 5th ACM-SIGPLAN Workshop on Scheme and Functional Programming", September 2004, p. 75-88.

[22] 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", ACM Press, Jan 2004.

[23] S. EPARDAUD. *Mobile reactive programming in ULM*, in "Proceedings of the Fifth ACM SIGPLAN Workshop on Scheme and Functional Programming, Snowbird, Utah", O. SHIVERS, O. WADDELL (editors)., Sep 22, 2004, p. 87-98, http://www-sop.inria.fr/mimosa/Stephane.Epardaud/papers/ulm-scheme-2004.pdf.

[24] A. MATOS, G. BOUDOL, I. CASTELLANI. *Typing noninterference for reactive programs*, in "Foundations of Computer Security", 2004, http://www-sop.inria.fr/mimosa/personnel/Ana.Matos/fcs04.pdf.

[25] M. SERRANO, F. BOUSSINOT, B. SERPETTE. *Scheme Fair Threads*, in "6th Acm sigplan International Conference on Principles and Practice of Declarative Programming (PPDP), Verona, Italy", (taux d'acceptation 21/44), Aug 2004, p. 203–214, http://www.inria.fr/mimosa/Manuel.Serrano/publi/sbs-ppdp04.html.

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

Internal Reports

[27] R. AMADIO. *Synthesis of max-plus quasi-interpretations*, To appears in Fundamenta Informaticae, Research report, n° 18-2004, LIF, Marseille, France, January 2004.

[28] G. BOUDOL. *Safe recursive boxes*, Technical report, n° RR-5115, INRIA, February 2004, http://www.inria.fr/rrrt/rr-5115.html.

[29] F. BOUSSINOT. *Reactive Programming of Cellular Automata*, Technical report, n° RR-5183, Inria, May 2004, http://www.inria.fr/rrrt/rr-5183.html.

Miscellaneous

[30] F. BOUSSINOT. *Concurrent Programming with Fair Threads – The LOFT Language*, 2004, http://www-sop.inria.fr/mimosa/rp/LOFT/doc/book/book.html.

[31] F. DABROWSKI. *Reactivity in ULM*, 2004.

[32] F. DABROWSKI. *Side effects and polynomial bounds*, 2004.