INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team mimosa

Migration et Mobilité: Sémantique et Applications

Sophia Antipolis - Méditerranée

THEME COM

Activity Report

2007

Table of contents

1. Team

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

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

Administrative assistant
Sophie Honnorat [Inria]

Staff members Inria
Manuel Serrano [Research Director, Inria, HdR]
Olivier Tardieu [Research Scientist, Inria, till November 30]

Staff members CMA
Frédéric Boussinot [Research Director, CMA, HdR]

Reserch scientists (external)
Roberto Amadio [Professor, University of Paris 7, HdR]

Visting scientist
Erick Gallesio [Visiting Scientist, University of Nice Sophia-Antipolis, till October 1]

Ph. D. students
Marija Kolundzija [University of Torino]
Frederic Dabrowski [MENRT, till April 1]
Stéphane Epardaud [MENRT]
Florian Loitsch [MENRT]

2. Overall Objectives

2.1. Introduction

The overall objective of the MIMOSA project is to design and study models of concurrent, 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 have integrated migration primitives in this reactive approach. In the past we also intensively studied models of mobility, like the π-calculus and its distributed variants, and the calculus of Mobile Ambients. Our main research areas are the following:

- Security. We investigate security issues like confidentiality and resource consumption, using static analysis methods (for the verification of non-interference of programs with respect to given security policies, and of computational complexity), with an emphasis on the issues related to concurrent and mobile 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 study the integration of 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*. All our developments on the web rest on these implementations.
- Web programming. We design and implement a programming environment for the web 2.0. It relies on a new distributed programming architecture where a program executes simultaneously on a server and a client. We aim at providing a realistic implementation that we constantly validate by developing and using end-users web applications.

2.2. Highlights of the year

HOP, a software development kit for the Web, has won the software *open source contest* organized by the ACM Multimedia Conference 2007 (see http://mmc36.informatik.uni-augsburg.de/acmmm2007/).

3. Scientific Foundations

3.1. Semantics of mobility

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 have studied 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 designed and implemented a core programming language that integrates reactive programming and mobile code, in the context of classical functional and imperative programming.

3.2. Security of concurrent and mobile programs

We are studying security issues, especially those related to concurrent and mobile programming. In the past we have developed methods and tools for the verification of cryptographic protocols. We also work on secure information flow. This is motivated by the observation that access control is not enough to ensure confidentiality, since access control does not prevent authorized users to disclose confidential information. We use the language-based approach, developing static analyses, and especially type systems, to ensure that programs do not implement illegal flow of information. We work particularly on specific confidentiality issues arising with concurrent and mobile programming, but we also work on more general questions, like how to allow some pieces of code to declassify some information, while still ensuring some confidentiality policy.

We also use static analysis techniques, namely polynomial quasi-interpretations, to ensure that programs do not use computational resources beyond fixed limits. Again, a special effort is put here in finding methods that apply to reactive and/or mobile programs. This could also have applications to embedded code.

3.3. 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 [6]. 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).

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 programming

Along with games, multimedia applications, and email, the web has popularized computers in everybody's life. The revolution is engaged and we may be at the dawn of a new era of computing where the web is a central element.

Many of the computer programs we write, for professional purposes or for our own needs, are likely to extensively use the web. The web is a database. The web is an API. The web is a novel architecture. Therefore, it needs novel programming languages and novel programming environments.

In addition to allowing reactive and graphically pleasing interfaces, web applications are de facto distributed. Implementing an application with a web interface makes it instantly open to the world and accessible from much more than one computer. The web also partially solves the problem of platform compatibility because it physically separates the rendering engine from the computation engine. Therefore, the client does not have to make assumption on the server hardware configuration, and vice versa. Lastly, HTML is highly durable. While traditional graphical toolkits evolve continuously, making existing interfaces obsolete and breaking backward compatibility, modern web browsers that render on the edge web pages are still able to correctly display the web pages of the early 1990's.

For these reasons, the web is arguably ready to escape the beaten track of n-tiers applications, CGI scripting and interaction based on HTML forms. However, we think that it still lacks programming abstractions that minimize the overwhelming amount of technologies that need to be mastered when web programming is involved. Our experience on reactive and functional programming is used for bridging this gap.

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 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 has first be implemented in JAVA. It is 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 trade-offs 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: Stéphane Epardaud, Erick Gallesio, 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.

5.3.2. ULM

ULM is a new language for the mobility that is developed in the team. The ULM Scheme implementation is an embedding of the ULM primitives in the Scheme language. The bytecode compiler is available on PCs only but there are two ULM Virtual Machines: one for PCs and one for embedded devices supporting Java 2 Mobile Edition (J2ME) such as most mobile phones. The current version has preliminary support for a mixin object model, mobility over TCP/IP or Bluetooth Serial Line, 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. Web programming

Participants: Florian Loitsch, Manuel Serrano.

5.4.1. The HOP web programming environment

HOP is a new higher-order language designed for programming interactive web applications such as web agendas, web galleries, music players, etc. It exposes a programming model based on two computation levels. The first one is in charge of executing the logic of an application while the second one is in charge of executing the graphical user interface. HOP separates the logic and the graphical user interface but it packages them together and it supports strong collaboration between the two engines. The two execution flows communicate through function calls and event loops. Both ends can initiate communications.

The HOP programming environment consists in a web *broker* that intuitively combines in a single architecture a web server and a web proxy. That broker embeds a HOP interpreter for executing server-side code and a HOP client-side compiler for generating the code that will get executed by the client.

An important effort is devoted to providing HOP with a realistic and efficient implementation. The HOP implementation is *validated* against web applications that are used on daily-basis. In particular, we have developed HOP applications for authoring and projecting slides, editing calendars, reading RSS stream, or managing blogs.

HOP has won software *open source contest* organized by the ACM Multimedia Conference 2007 (http://mmc36.informatik.uni-augsburg.de/acmmm2007/). It is released under the GPL license. It is available at http://hop.inria.fr.

5.4.2. Scheme2JS

Scm2JS is a Scheme to JavaScript compiler distributed under the GPL license. Even though much effort has been spent on being as close as possible to R 5 RS, we concentrated mainly on efficiency and interoperability. Usually Scm2JS produces JavaScript code that is comparable (in speed) to hand-written code. In order to achieve this performance, Scm2JS is not completely R 5 RS compliant. In particular it lacks exact numbers.

Interoperability with existing JavaScript code is ensured a JavaScript-like dot-notation to access JavaScript objects and by a flexible symbol-resolution implementation.

Scm2JS is used on a daily basis within HOP, where it generates the code which is sent to the clients (web-browsers).

Scm2JS can be found here (http://www-sop.inria.fr/mimosa/personnel/Florian.Loitsch/scheme2js/)

5.5. Old softwares

5.5.1. Bugloo

BUGLOO, is 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.

5.5.2. 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.

5.5.3. Icobjs

ICOBJS programming is a simple and fully graphical programming method, using powerful means to combine behaviours. This style of programming is based on the notion of an *icobj* which has a behavioural aspect (object part), and a graphical aspect (icon part), and which can be animated on the screen. ICOBJS programming evolves from the reactive approach and provides parallelism, broadcast event communication and migration through the network. 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.

5.5.4. 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 .

5.5.5. 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.

5.5.6. 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

6. New Results

6.1. Security

Participants: Roberto Amadio, Gérard Boudol, Ilaria Castellani, Frédéric Dabrowski, Marija Kolundzija.

6.1.1. 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, multithreaded code, in order to ensure that migrating agents and, more generally, concurrent threads do not corrupt protected data, and that the behaviour of such agents or threads 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. More specifically, we have introduced a block-structured programming contruct that allows the programmer to extend the current information flow policy by new rules for legal flow, in order to declassify information in a sub-program. The use of this construct is however unrestricted, and no way is provided to control how declassification is used. In [15] we have addressed this problem. The idea is to use standard access control mechanisms, to constrain a program to declassify only information that is is allowed to read, according to the access control policy. This is implemented by considering a high level programming language, extending the one we used for dealing with declassification, that is, a higher-order imperative language à la ML, enriched with contructs for dynamically managing the use of security policies, namely a mechanism for locally extending the flow relation (as in the work of Boudol and Matos on declassification), and constructs for extending, restricting and testing the access rights assigned to a piece of code, à la JAVA. We then define a type system for this language, extending the one of Boudol and Matos, and we show that typable programs are secure, not only from the information flow point of view, but also from the access control point of view. That is, a typable program never attempts, at any point of execution, to access information for which the appropriate access rigth is not currently granted. As a consequence, in particular, a typable program never attempts to declassify information that is does not have the right to read. Moreover, no dynamic access control check is needed in running a typable program.

6.1.2. Relation between language-based security and process calculi security

As regards security properties and related type systems, we pursued our work on the relation between noninterference for parallel programming languages and security notions for process calculi. A first outcome of this work is presented in the papers [[16],[25]] (the second being the full version of the first). In these papers we addressed the question of typing noninterference (NI) in Milner's Calculus of Communicating Systems (CCS), in such a way that Milner's translation of a standard parallel imperative language into CCS preserves both an existing NI property and the associated type system. In doing so, we generalized previous work by Focardi, Rossi and Sabelfeld, who showed that a variant of Milner's translation, restricted to the sequential fragment of the language, maps a time-sensitive NI property to that of Persistent Bisimulation-based Non Deducibility on Compositions (PBNDC) on CCS. We extended Focardi, Rossi and Sabelfeld's result by showing that a simpler variant of Milner's translation preserves a time-insensitive NI property on the full parallel language, by mapping it again to PBNDC. As a by-product, we formalised a folklore result, namely that Milner's translation preserves a behavioural equivalence on programs. We also presented a security type system for CCS, inspired from existing type systems for the pi-calculus. We proved that this type system ensures the PBNDC-property. We also showed that, as it stands, this type system is too restrictive to reflect any of the proposed type systems for the imperative language. We outlined a solution to overcome this problem, by slightly relaxing the type system for CCS.

6.2. Reactive programming

Participants: Roberto Amadio, Gérard Boudol, Frédéric Boussinot, Ilaria Castelani, Frédéric Dabrowski, Stéphane Epardaud, Olivier Tardieu.

6.2.1. The reactive model

6.2.2. Safe concurrent programming

We are designing a new safe language for reactive programming, called FunLoft. The language is based on FairThreads [5]. In FairThreads, a thread may either be linked to a scheduler and run cooperatively, or be unlinked and run preemptively. Threads communicate using both broadcast events and shared memory. Each event is associated with a unique scheduler which defines global instants for the threads linked to it and for its associated events. In FunLoft, each scheduler and each thread owns a distinct portion of the memory such that:

- The memory of a scheduler can only be accessed by the threads which are linked to it.
- The memory of a thread can only be accessed by it.

Non-intersection of distinct portions of the memory is statically checked using a type and effect system. This entails absence of thread interferences possibly leading to data-races. Termination of recursive functions is also controlled. In the present version of FunLoft, recursivity can only concern parameters of inductive types. To control the size of the global memory used by a program, memory locations are stratified over a finite domain. We rely on a simple data flow analysis to ensure that no cyclic dependence is possible for memory locations carrying complex datatypes (simple datatypes such as integers do not have this restriction). Several other static controls are also made, for example to guarantee that the number of living threads remains bounded. Static checks insure three fundamental properties of FunLoft: that programs are free of memory leaks, that code between two cooperation points is atomic, and that rounds are fair and always terminate (no run-time error nor divergence). A formal semantics exists for FunLoft, which takes into account the cooperative aspects as well as the preemptive ones [23]. The semantics restricted to a unique scheduler has been presented in a previous paper; the complete semantics, when there are several schedulers, of a model very close to FunLoft is described in the thesis of F. Dabrowski [10]. We are currently developing an implementation prototype (version

0.2), available at `http://www.inria.fr/mimosa/rp/FunLoft`. The prototype is used to implement various examples of graphical simulations (for example, cellular automata made of several thousands of cells, each being a thread). These examples show that systems with large numbers of concurrent units can be efficiently implemented in our language.
In Section 6.3.1 we present another approach, still using a type and effect system, of the fairness problem for a higher-order, concurrent (cooperative) and imperative language. More precisely, in [14] it is shown how to restrict the use of higher-order memory, in order to ensure that concurrent threads are cooperative, and that the "instants" in a higher-order reactive model are finite.

6.2.3. Reactive and mobile programming: ULM

During the year 2007, we have developed a prototype application in ULM for France-Télécom. The goal of this prototype is to study the possible use of mobile agents for dynamic reconfiguration of software components. This led to various improvements of the ULM virtual machine, compiler and runtime libraries. The mixin object system has been rewritten for a smaller code, processing and memory footprint. An option has been added while migrating an agent which will notify of the migration's success or failure, with possible local recovery of the migrated agent in case of failure. Terminated agents and unused signals are now being garbage collected. We have added support for SSL certificates and encryption for the transportation of agents. A new program lets users disassemble compiled ULM bytecode.

6.2.4. Synchronous Programming: Esterel

Instantaneous Transitions Esterel makes it possible to program hierarchical finite state machines in a safe and intuitive way. By contrast, specifying flat automata in Esterel is cumbersome and error prone. Many standards however explicitly prescribe unstructured state machines. For example, the link layer specification of the Serial ATA standard specifies a 31-state machine by listing transitions in a table. Previously, we extended Esterel with a primitive construct to encode non-instantaneous state transitions. This year, we further extended the language to cope with instantaneous transitions [21]. It benefits the programmer who can directly specify transition tables in Esterel if needed. It also benefits formal verification as, in contrast with previous techniques for encoding unstructured states machines in Esterel, it does not introduce spurious transitions.
Deterministic Semantics The deterministic semantics of Esterel we previously introduced defines at most one behavior per program and input sequence. It may define zero if the program is not logically correct. This year, we revised the deterministic semantics so as to generate error codes instead [13]. The new semantics formalizes error locations and causes. It enables formal reasoning about incorrect programs using only positive logic, that is, logic without negation.

6.2.5. Multicore and Embedded Systems: SHIM

Because of multicore and embedded systems, there is a growing need for languages and tools that help programmers exploit parallel processing units in software systems without requiring dramatic changes in development methodologies. Common concurrent models allow the behavior of a program to depend on the scheduling of operations. Unfortunately, reasoning about scheduling can be a new and difficult task for a programmer used to writing sequential code. SHIM is a novel concurrent programming language that avoids many of the pitfalls of traditional concurrent programming languages—e.g. data races—by adopting deterministic message passing as it sole communication mechanism.
Code Generation This year, we implemented a SHIM to C-plus-Pthreads compiler that can produce efficient code for shared-memory multiprocessors. We present a parallel JPEG decoder and FFT exhibiting 3.05$\times$ and 3.3$\times$ speedups on a four-core processor. This work will be presented in March 2008 at the Design, Automation, and Test in Europe conference.
Safety This year, we also formally established race freedom for a core language with local channels and recursive process definitions. This work is currently under review.

6.2.6. *Programming of multicore machines*

In FunLoft, schedulers and unlinked threads should be mapped onto native threads that are dispatched on available cores in a preemptive way. A submitted paper [24] describes the interest of FunLoft for the programming of multicore machines. A brief note [22] proposes a simulation of colliding particles as a benchmark for multicore machines.

6.2.7. *Logic Synthesis*

Optimizing sequential cycles is essential for many types of high-performance circuits, such as pipelines for packet processing. Retiming is usually applied to such circuits, which renders the length of purely combinational paths nearly irrelevant since retiming can divide such paths among multiple clock cycles to increase the clock rate. However, because retiming cannot change the number of registers on a sequential cycle, the depth of the combinational logic along sequential cycles becomes the bottleneck. Designers usually attack such cycles by manually combining Shannon decomposition with retiming. Shannon decomposition provides a way to restructure logic to hide the effects of late-arriving signals. This is done by duplicating a cone of logic, feeding constant 1s and 0s into the late-arriving signal and placing a two-input multiplexer on the output of the two cones. Following Shannon decomposition with retiming can greatly improve overall circuit performance. Since Shannon decomposition can move logic out of sequential loops, a subsequent retiming step can better balance the logic to reduce the minimum clock period, giving a faster circuit. In 2006, we proposed an efficient algorithm that simultaneously applies Shannon decomposition and retiming to optimize circuits with tight sequential cycles. This year, we extended this work with an in-depth analysis of optimality and convergence [12].

6.3. Functional programming

Participants: Gérard Boudol, Erick Gallesio, Florian Loitsch, Manuel Serrano.

6.3.1. *Fair cooperative multithreading*

One of the main aim of our project is to integrate the constructs of reactive programming into expressive programming languages, like SCHEME or ML. However, there is in principle a difficulty in this integration, which is that in the reactive style of programming, and more generally in any multi-threading context where the scheduling is cooperative, any program must be cooperative, or fair, that is any program must release the control after a finite amount of time of running. Thread code may be written so as to be fair, using a “yield” instruction for instance, but it is not clear how to formally check the fairness property (which is undecidable in an expressive programming language). To address this issue, we have proposed in [14] a new operational model for shared variable concurrency, in the context of a concurrent, higher-order imperative language à la ML. The main novelty is the introduction of a “yield-and-loop” construct, to program intentionnally non-terminating programs, like servers for instance, the semantics of which is that a looping process suspends itself on each recursive call. In order to show the fairness property in this model – that is, any thread yields the scheduler after some finite computation –, we follow and adapt the classical method for proving termination in typed formalisms, namely the realizability technique. There is a specific difficulty with higher-order state, which is that one cannot define a realizability interpretation simply by induction on types, because applying a function may have side-effects at types not smaller than the type of the function. Moreover, such higher-order side-effects may give rise to computations that diverge without resorting to explicit recursion. We overcome these difficulties by introducing a type and effect system for our language that enforces a stratification of the memory. The stratification prevents the circularities in the memory that may cause divergence, and allows us to define a realizability interpretation of the types and effects, which we then use to prove the intended termination property. Our realizability interpretation also copes with dynamic thread creation, although the main difficulty lies in the sequential (imperative and functional) fragment of the language.

In the full version of [14] (available from the web page of the author, and submitted for publication in a technical journal), we have included a block-structured construct for declaring local regions in the memory, for effect masking purposes, as in the early work of Lucassen and Gifford. This construct is also used by Tofte and Talpin for memory management purposes, with a different semantics, namely that a local region is reclaimed at the end of its scope. The type safety property is difficult to establish in this case. In the course of proving the results presented in [14], we have discovered a new proof of type safety for Tofte and Talpin's calculus of regions. This new proof relies on a new type system for a language extended with an explicit memory deallocation construct. Then not only we have a simple proof of the safety of Tofte and Talpin's typing for the region calculus, but moreover we have a sound type discipline for explicit memory deallocation. This is exposed in a paper entitled “Typing safe deallocation” (by G. Boudol), which has been submitted for publication in a conference.

6.3.2. Bigloo

During the year 2007, we have designed and implemented a new package management system for the Scheme programming language named ScmPkg [19]. It is inspired by the *Comprehensive Perl Archive Network* (CPAN) and various GNU/Linux distributions. It downloads, installs, and prepares source codes for execution. It manages the dependencies between packages. The main characteristic of this system is its neutrality with respect to the various Scheme implementations. It is neutral with respect to the language extensions that each Scheme implementation proposes *and* with respect to the execution environment of these implementations. This allows the programmer to blend, within the same program, independent components which have been developed and tested within different Scheme implementations. We have successfully compiled applications blending Bigloo, Chicken, MzScheme, and STklos codes.
ScmPkg is a loose architecture that consists of a web server, a simple Interface Desciption Language (IDL) for describing packages, and ad-hoc package managers that are provided by the Scheme implementations (that we henceforth denote as *Scheme hosts*, *hosts*, or *dialects*) that support ScmPkg.

- The web server implements a graphical user interface for browsing the packages, their documentation, and their source code. It also supports two services that are used for installing the packages. The first one provides the whole list of packages and the second implements downloading of individual packages.
- An IDL describes the material exposed by a package along with some additional meta information.
- Each Scheme host that supports ScmPkg must provide a *package manager*. This is a simple program that is in charge of preparing a package for its host. This, in general, involves *downloading*, *preparing* and *installing* the code. Optionally it may also *compile* the code. The package manager is the equivalent to the `apt-get` command of GNU/Linux Debian distributions.

Before being compiled or interpreted a package may have to be *adapted* to a Scheme host. That is, some parts of its interface and its implementation may have to be rewritten. It is handled automatically by the host package manager. This process is at the heart of the whole system. The general idea of adaptation is that Scheme hosts are compatible to a large extent. They all support a common set of extensions even though they don't share the same interfaces. Sometimes, *reusing* a source code implemented for a host `H1` within a host `H2` requires us to provide glue that is made of additional functions and macros. It may also require us to override some definitions of the original source code (either because `H2` allows a faster implementation or because the `H1` implementation uses features that have no counterpart in `H2`). ScmPkg proposes a methodology for automatically processing *host adaptation*. This distinguishes ScmPkg from other packaging systems.
ScmPkg is available at:

http://hop.inria.fr/hop/scmpkg)
ScmPkg has been integrated into Bigloo in the version 3.0 which has been released during 2007. The activity on the Bigloo mailing list has been steady with approximatively new 800 messages posted, the same activity as in 2006.

6.4. Web programming

Participants: Florian Loitsch, Manuel Serrano.

6.4.1. Hop

During the year 2007, several major versions of HOP have been released. The last one, the version 1.8.0, released in November, contains several new APIs such as:

- SVG and MathML support.
- Canvas for interactive drawing.
- WebDAV client-side and server-side support.
- ...

In addition to adding new features, we have focused most of our efforts into improving the quality of the system.

- The reliability of the HOP Web server has been significantly improved. In consequence, we have been able to run continuously the HOP Web site http://hop.inria.fr, during several months without any problem.
- The portability of the HOP Web server has been improved. The last version can now be installed on any regular PC running any main-stream operating system. In addition, successful experiments for installing it on NASes and PDAs (Network Attached Storages) have been conducted. We plan to release official pre-compiled versions for theses architectures during the year 2008. The last portability issue we have addressed during the year is the
- The client-side execution of HOP programs has been improved. It is well-known that writing portable Web applications, that is, Web applications that can be executed by all the mainstream Web browsers (e.g., Firefox, Safari, Internet Explorer) is a difficult problem. Since HOP relies on a compiler for *generating* client-code on the fly, it has an opportunity to tune the produced code for the each browser. Hence, the burden of producing portable client-side code no longer rests on the shoulders of the programmer of the Web application but on the ones of the HOP implementor. We have made this effort this year.

After a whole year of improvements, we deem that HOP is now polished enough for being used for production-quality applications. Hence, we have started implementing our first HOP Web applications:

- A full-fledged mail client. This application is now used on a daily-basis by its author. It can be executed by regular PCs and by PDAs. The application automatically adapts its GUI to fit the size of its output device.
- A slide authoring system. This application allows slides to contain SVG images, MathML formulas, and other traditional multimedia resources. It supports graphical animations and video overlays for highlighting parts of a slide.
- An ubiquitous home media center. That system can use many sources of music and radios and it can control several output speakers. All the electronic devices that can run a Web browser can participate to the application. A mobile phone can turn on the music and select a podcast, a laptop can lower the volume, a PDA can switch off the speakers of the living room and switch on the speakers located in the kitchen, etc. We feel that without the Web it would have been impossible for us to develop such an application, or at it least, it would have been much harder. However, although the “standard” Web technologies are almost sufficient for building our targeted application, some mandatory features are still missing in current Web SDKs. In particular, the Web lacks portable means for playing music. HOP has bridged this gap. This application has been sketched in one of our publication [20].

6.4.2. Scm2Js

We continued the development of Scm2JS, our Scheme to JavaScript compiler. We experimented with different `call/cc` implementations (all based on exceptions) [17]. The final implementation contains several optimizations: it splits `call/cc` points into suspension and resume points, which allows to treat each separately. During restoration Scm2JS needs to emulate a jump to the resume-point, and this `goto` can be optimized, now that the jump-target can be manipulated independently from the suspension point.

We furthermore improved `call/cc` calls at tail-locations. These locations do not need any instrumentation anymore.

`Call/cc` support made it furthermore necessary to refactor Scm2JS's internals to work around JavaScript's missing `let` keyword. Whereas without `call/cc` most nested scopes can be easily merged into the surrounding function scope. Continuations however invalidate such simplifications.

Scm2JS has seen many improvements in other areas too. Scm2JS is now multithread-safe. We have added module-support. Scm2JS's documentation has been rewritten. Many bugs have been fixed.

The tool-chain around Scm2JS has been enhanced too. An `exporter` simplifies the creation of JavaScript interfaces. HOP has been modified to benefit from this tool.

7. Contracts and Grants with Industry

7.1. CRE France-Télécom R&D

A CRE (contract for external research) started this year, funded by France-Télécom R&D on "Analysis of security properties for global programming frameworks". The duration of the project is 3 years (may be extended to 4), and the total funding is 120 kEuros. The purpose of the project is to support the research done in MIMOSA on security issues and mobile code, and to study the applicability of our methods and results to concrete problems investigated at France-Télécom R&D.

8. Other Grants and Activities

8.1. National initiatives

8.1.1. APP registration for Hop

Hop has been registered to the APP on June 1fst, 2006 under the reference : `IDDN.FR.001.260002.000.S.P.2006.000.10400`.

8.1.2. ACI Sécurité Informatique ALIDECS

Frédéric Boussinot is participating to the ACI Sécurité Informatique ALIDECS whose coordinator is Marc Pouzet. The ACI started in 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.

8.1.3. ANR SETIN ParSec

The PARSEC project (for "Parallélisme et Sécurité") has been funded by the ANR Sécurité Informatique programme for 4 years, starting January 2007. The partners of this project are the teams MIMOSA (coordinator) and EVEREST at INRIA Sophia Antipolis, LANDE at IRISA, MOSCOVA at INRIA Rocquencourt and CONCURRENCY at the PPS Laboratory of Paris 7 University and CNRS.

9. Dissemination

9.1. Seminars and conferences

Gérard Boudol presented the research work of the MIMOSA team on security at the the first meeting of the ANR SETIN project PARSEC in Sophia Antipolis, February 2007. He participated in the second workshop at the MSR-INRIA Laboratory, Orsay, in June 2007 (where [15] was presented by Marija Kolundzija), and in the third workshop at the PPS Laboratory, Paris, December 2007, where he presented [14]. He attended the ETAPS'07 conference in Braga and the TLCA'07 conference in Paris. He participated in the CONCUR'07 conference in Lisbon, where he presented [14] and in the MMM-ACNS'07 workshop in St Petersbourg (where [15] was presented by Marija Kolundzija).

Ilaria Castellani took part in the three meetings of the ANR project PARSEC (held respectively at INRIA Sophia Antipolis, in February 2007, at the MSR-INRIA laboratory, Orsay, in June 2007, and at the CNRS-Paris7 laboratory PPS, in December 2007). At the second PARSEC meeting she presented the submitted work [16]. She participated to the conference CONCUR'07 and to its satellite workshop SecCo'07, where she presented the paper [16] (Lisbon, September 2007). She attended the workshop TGC'07 and its associated workshop on the Interplay of Programming Languages and Cryptography (INRIA Sophia Antipolis, November 2007).

Marija Kolundzija participated in the workshops of the ANR SETIN project PARSEC, and in the MMM-ACNS'07 workshop in St Petersbourg. She presented [15].

Florian Loitsch participated in the Dagstuhl seminar on *Programming paradigms for the Web*. He presented the paper [18] at Trends of Functional Programming in New York. Gave a talk on Scm2JS continuation techniques [17] at the *2007 Workshop on Scheme and Functional Languages*.

Manuel Serrano gave several presentations of the HOP system. In an attempt to establish collaboration with Orange he participated in two seminars at Orange R&D. The first one took place in Grenoble the second one in Sophia-Antipolis. He gave an invited presentation of HOP at the 2007 International Lisp Conference that took place in Cambridge, UK. He gave another invited presentation on programming multimedia application with HOP at the Montreal Scheme and Lisp User Group seminar. He participated to the ACM Multimedia 2007 event where he has received the price of the *best open source software*. He participated to the Dagstuhl seminar on *Programming paradigms for the Web*.

Olivier Tardieu took part in two meetings of the ANR project PARSEC (held respectively at INRIA Sophia Antipolis, in February 2007, at the MSR-INRIA laboratory, Orsay, in June 2007). Olivier Tardieu participated to the ETAPS'07 conference and to its SLA++P'07 satellite workshop where he presented the paper [21] (Braga, Portugal, April 2007). Olivier Tardieu participated to the POPL'07 conference (Nice, January 2007) and to the DATE'07 conference (Nice, March 2007) where he received a best-paper award for his 2006 paper "Optimizing Sequential Cycles through Shannon Decomposition and Retiming" written with Cristian Soviani and Prof. Stephen A. Edwards from Columbia University, New York.

9.2. Animation

Gérard Boudol is the coordinator of the ANR SETIN project PARSEC. He organized the first meeting of this project in Sophia Antipolis, February 2007. He was a member of the ESOP'07 programme committee. He was a reviewer of the PhD Theses of Gurvan Le Guernic (IRISA) and of Damien Pous (ENS Lyon). He was an examiner of the PhD Thesis of Frédéric Dabrowski.

Frédéric Boussinot was examiner of the PhD thesis of F. Dabrowski. F. Boussinot was invited speaker at the SYNCHRON'07 Workshop (Bamberg) and has participated to the Program Comitee of MSR'07 (Lyon).

Manuel Serrano was a member of the program committee of the *ACM SIGPLAN Commercial Users of Functional Programming*.

Olivier Tardieu is a member of the IEEE working group P1778 on Esterel v7 Standardization and particitaped to its monthly meetings. Olivier Tardieu visited Prof. Stephen A. Edwards in Columbia University, New York, during August 2007.

9.3. Teaching

Florian Loitsch is *moniteur* at the University of Nice Sophia-Antipolis. He participated to Scheme, C, and compilation courses.

Manuel Serrano supervised the internship of three undergraduate students during the summer 2007. During two months, they worked on developing Web applications with HOP.

10. Bibliography

Major publications by the team in recent years

[1] A. ALMEIDA MATOS, G. BOUDOL. *On declassification and the non-disclosure policy*, in "Computer Security Foundation Workshop", 2005, p. 226-240.

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

[3] G. BERRY, G. BOUDOL. *The chemical abstract machine*, in "Theoretical Computer Science", vol. 96, 1992.

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

[5] F. BOUSSINOT. *FairThreads: mixing cooperative and preemptive threads in C*, in "Concurrency and Computation: Practice and Experience", vol. 18, n^{o} DOI: 10.1002/cppe.919, 2006, p. 445-469.

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

[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] M. SERRANO, E. GALLESIO, F. LOITSCH. *HOP, a language for programming the Web 2.0*, in "Proceedings of the First Dynamic Languages Symposium, Portland, Oregon, USA", October 2006.

[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.

Year Publications

Doctoral dissertations and Habilitation theses

[10] F. DABROWSKY. *Programmation Réactive Synchrone: langages et contrôle des ressources*, Ph. D. Thesis, Université Paris 7, 2007.

Articles in refereed journals and book chapters

[11] A. ALMEIDA MATOS, G. BOUDOL, I. CASTELLANI. *Typing Noninterference for Reactive Programs*, in "Journal of Logic and Algebraic Programming", vol. 72, 2007, p. 124-156.

[12] C. SOVIANI, O. TARDIEU, S. EDWARDS. *Optimizing Sequential Cycles through Shannon Decomposition and Retiming*, in "IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems", vol. 26, n^o 3, Mar 2007, p. 456–467.

[13] O. TARDIEU. *A deterministic logical semantics for pure Esterel*, in "ACM Transactions on Programming Languages and Systems", vol. 29, n^o 2, Apr 2007, 8.

Publications in Conferences and Workshops

[14] G. BOUDOL. *Fair cooperative multithreading, or: typing termination in a higher-order concurrent imperative language*, in "CONCUR'07", Lecture Notes in Computer Science, vol. 4703, 2007, p. 272-286.

[15] G. BOUDOL, M. KOLUNDZIJA. *Access control and declassification*, in "MMM-ACNS'07", Communications in Computer and Information Science, vol. 1, 2007.

[16] I. CASTELLANI. *State-oriented noninterference for CCS*, in "Proceedings SecCo'07", D. GORLA, C. PALAMIDESSI (editors), ENTCS, vol. 194, n^o 1, Elsevier Science Publishers, 2007, p. 39-60.

[17] F. LOITSCH. *Exceptional Continuations in JavaScript*, in "2007 Workshop on Scheme and Functional Programming", Sep 2007.

[18] F. LOITSCH, M. SERRANO. *Hop Client-Side Compilation*, in "Proceedings of the 8th Symposium on Trends on Functional Languages, New Yort, USA", Apr 2007.

[19] M. SERRANO, E. GALLESIO. *An Adaptive Package Management System for Scheme*, in "Proceedings of the Second Dynamic Languages Symposium, Montréal, Québec, Canada", Oct 2007.

[20] M. SERRANO. *Programming Web Multimedia Applications with Hop*, in "Proceedings of the ACM Sigmm and ACM Siggraph conference on Multimedia, Best Open Source Software, Augsburg, Germany", Sep 2007.

[21] O. TARDIEU, S. EDWARDS. *Instantaneous Transitions in Esterel*, in "Proceedings of the Workshop on Model-Driven High-Level Programming of Embedded Systems, Braga, Portugal", Mar 2007.

Internal Reports

[22] F. BOUSSINOT. *A Benchmark for Multicore Machines*, Technical report, n^o inria-00174843, 2007, http://hal.inria.fr/inria-00174843.

[23] F. BOUSSINOT, F. DABROWSKY. *Formalisation of FunLoft*, Technical report, n^o inria-00183242, 2007, http://hal.inria.fr/inria-00183242.

[24] F. BOUSSINOT, F. DABROWSKY. *Safe Reactive Programming: the FunLoft Prop osal*, Technical report, n^o inria-00184100, 2007, http://hal.inria.fr/inria-00184100.

[25] I. CASTELLANI. *State-oriented noninterference for CCS*, Research Report, n^{o} 6322, INRIA, 10 2007, http://hal.inria.fr/inria-00180168/fr/.