INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team mimosa

Migration et Mobilité: Sémantique et Applications

Sophia Antipolis - Méditerranée

THEME COM

Table of contents

1. Team

Research Scientist

Gérard Boudol [Research Director, Inria, HdR]
Frédéric Boussinot [Research Director, CMA, HdR]
Ilaria Castellani [Research Scientist, Inria]
Tamara Rezk [Research Scientist, Inria, from September 1]
Manuel Serrano [Research Director, Inria, HdR]

Faculty Member

Roberto Amadio [Professor, University of Paris 7, HdR]

Technical Staff

Anne-Marie Burns [Inria, from January 1, till November 20]

PhD Student

Stéphane Epardaud [MENRT, till February 28]
Marija Kolundzija [University of Torino and Inria]
Florian Loitsch [MENRT]
Zhengqin Luo [MENRT, from October 1]
Gustavo Petri [IP Mobius, from October 1]

Administrative Assistant

Anais Cassino [Inria]
Sophie Honnorat [Inria]

2. Overall Objectives

2.1. Overall Objectives

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 code mobility. 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.

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. Actually, 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 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 have 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, AutoCad, ...) 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 assumptions 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. Introduction

Most MIMOSA softwares, even the older stable ones that are not described in the following sections are freely available on the Web. In particular, some are available directly from the INRIA Web site:
http://www.inria.fr/valorisation/logiciels/langages.fr.html
Most other softwares can be downloaded from the MIMOSA Web site:
http://www-sop.inria.fr/teams/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 was first 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 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: 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/teams/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 (*Un langage pour la mobilité*) is a new language for 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/teams/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. The 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 the 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 R5RS, 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 R5RS compliant. In particular it lacks exact numbers.
Interoperability with existing JavaScript code is ensured by 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/scheme2js/)

5.5. Old software

5.5.1. LURC

LURC is a Reactive threading library in C. It is based on the reactive model of ULM (see Section 5.3.2)) 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 thread 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/teams/mimosa/Stephane.Epardaud/lurc.

5.5.2. 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 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.

5.5.3. Skribe

SKRIBE is a functional programming language designed for authoring documents, such as Web pages or technical reports. It is built on top of the SCHEME programming language. Its concrete syntax is simple and looks familiar to anyone used to markup languages. Authoring a document with SKRIBE is as simple as with HTML or LaTeX. It is even possible to use it without noticing that it is a programming language because of the conciseness of its original syntax: the ratio *markup/text* is smaller than with the other markup systems we have tested.
Executing a SKRIBE program with a SKRIBE evaluator produces a target document. It can be HTML files for Web browsers, a LaTeX file for high-quality printed documents, or a set of *info* pages for on-line documentation.

5.5.4. 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.

6. New Results

6.1. Security

Participants: Roberto Amadio, Gérard Boudol, Ilaria Castellani, Tamara Rezk.

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's 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, multi-threaded 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 in [1] 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, and we have extended the security analysis to this setting. However, the security property which is ensured is not easy to understand nor to deal with technically, and it was not clear how to extend it to other security-minded programming constructs.

In [14] we argue that, in the perspective of developing security-minded programming languages, the secure information flow property should be defined as a standard *safety* property, based on a notion of a security error, namely that one should not put in a public location a value elaborated using confidential information. This is formalized by means of a monitored operational semantics, where one records at each step the confidentiality level that has been acquired during the computation. This knowledge level is compared to the one of locations in which the program attempts to put the results of its computations, thus preventing security violations. We show that the resulting safety property is guaranteed by a standard security type system, and that, for a simple language, it is strictly stronger than non-interference. Moreover, we show that this notion of secure information flow allows us to give natural semantics to various security-minded programming constructs, including declassification.

6.1.2. Security properties and type systems for concurrent computations

The issue of secure information flow in concurrent computations has been intensively studied in the last decade, both for multi-threaded programming languages of various kinds (imperative, functional, synchronous, or combinations of them) and, to a minor extent, for process calculi (mainly variations of CCS and the π-calculus). Most studies focused on confidentiality properties such as non-interference, and provided type systems or other forms of analysis to statically ensure these properties. The relation among these different proposals has not been sufficiently investigated, however. In particular, the connection between the non-interference properties proposed for multi-threaded algebraic languages and the security properties proposed for process calculi needs to be further clarified.

1. We have pursued the work initiated in 2007, exploring the relation between non-interference in a standard multi-threaded imperative language and the security notion of "Persistent Bisimulation-based Non Deducibility on Compositions" (PBNDC) in the process calculus CCS. More precisely, we aimed at sharpening the results of a previous paper by first establishing a "fully abstract" (rather than simply "correct") translation from the language to the process calculus, and then, by tuning the security type system for the process calculus to make it reflect (under the defined translation) a standard security type system for the source language. This work is currently under completion.
2. We have engaged a comparative study of different confidentiality properties (mainly variants of non-interference and "observational nondeterminism"), in the context of multi-threaded imperative languages. This work is under way.
3. A new thread of research, recently started together with Tamara Rezk in collaboration with Ana Almeida Matos and Cédric Fournet, aims at reducing the gap between security type systems and the semantic security properties they are meant to enforce. Although type systems for undecidable properties such as non-interference are necessarily incomplete, it is often the case that they overly approximate the properties they are designed for. One idea for narrowing the gap between typability and the property it ensures, is to allow the introduction of some (restricted) semantic tests in the hypotheses of the typing rules. These tests typically consist in comparing two subterms of a given program with respect to some semantic property which is simpler than the one which is seeked for the whole program. We have started examining particular cases of such relaxed type systems, with the aim of obtaining a general theory of type systems with semantic tests, possibly not restricted to security properties.

6.2. Concurrency and Reactive programming

Participants: Gérard Boudol, Frédéric Boussinot, Ilaria Castellani, Stéphane Epardaud.

6.2.1. Concurrency

In the paper [19], we revisit some significant results obtained within the European BRA project CEDISYS, which brought together researchers at the meeting point between true concurrency and process algebra (among which members of the ancestor team of Mimosa), in the period 1988-1991. The endeavour and results of the project are placed in their historical context and discussed in the light of their recent developments. The impact of this work on the international research community is emphasized.

Memory models define an interface between programs written in some language and their implementation, determining which behaviour the memory (and thus a program) is allowed to have in a given model. A minimal guarantee memory models should provide to the programmer is that well-synchronized, that is, data-race free code has a standard semantics. Traditionally, memory models are defined axiomatically, setting constraints on the order in which memory operations are allowed to occur, and the programming language semantics is implicit at determining some of these constraints. In [20] we propose a new approach to formalizing a memory model in which the model itself is part of a weak operational semantics for a (possibly concurrent) programming language. We formalize in this way a model that allows write operations to the store to be buffered. This enables us to derive the ordering constraints from the weak semantics of programs, and to prove, at the programming language level, that the weak semantics implements the usual interleaving semantics for data-race free programs, hence in particular that it implements the usual semantics for sequential code.

6.2.2. Programming of multicore machines: FunLoft

During the year, we have continued the work around FunLoft. We have continued to formalise the semantics of FunLoft including the multischeduler aspect. A journal paper is in preparation on this matter. We have also considered the issue of garbage collection in the context of a reactive framework. A new reactive garbage collecting technique (RGC) has been designed, with responsiveness as a major objective. RGC mixes reference counter based and tracing based techniques, in a generational approach. With tracing-based GCs, it shares the existence of specific garbage collection phases. These phases are devoted to the collection of short-living objects, and are extremely light-weight. Like in tracing-based GCs, some threads must be stopped during collection; however, the concerned threads are naturally those linked to a scheduler. RGC does not introduce these phases but simply uses them, as they result from the language definition. With reference-counter based GCs, RGC share the existence of counters which are manipulated by executing threads, in order to detect unreachable objects. The counters however do not count the references on memory location, but a more abstract notion of protection. These protection counters are only used for long- living objects that live during several instants, not for short- living ones that disappear at the end of their instant of creation.
Several developments have been made on the FunLoft compiler (present version is v0.3.1), including the introduction of RGC, and the delivery of readable error messages. Readability of error messages is an important issue in the context of a language based on type inference, especially when effects are also to be considered.

6.3. Functional programming

Participants: Gérard Boudol, Florian Loitsch, Manuel Serrano.

6.3.1. Regions

In [15] we address the problem of proving, by static analysis means, that allocating and deallocating regions in the store provides a safe way to achieve memory management. That is, the goal is to provably ensure that a program does not use pointers into a deallocated region. A well-known approach to this problem is the one of Tofte and Talpin. Our first contribution is to provide a simple proof, by means of a subject reduction property, of the correctness of a type safety for their region calculus. Our second, main contribution is that we actually do this for an extension of Tofte-Talpin's calculus, featuring a primitive construct for deallocating regions, similar to C's `free`, that allows one to circumvent the strict stack-of-regions discipline enforced in Tofte-Talpin's calculus. Our static analysis consists in a novel type and effect system, extending the one of Tofte and Talpin, where we record deallocation effects.

6.3.2. Bigloo

The first version of the new Bigloo branch 3.1x has been released in May. In addition to fixing many minor problems, this major release added support for Big Numbers (i.e., exact integers). The three back-ends (namely, C, Java, and .NET) implement the new Bignum API.
The second version of that new branch (Bigloo 3.1b) has been released in September. This version contains several major novelties:

- New libraries for cryptography. It currently supports the symmetric AES encrypting/decrypting algorithm and SHA-1 hashing.
- A new XML parser that significantly improves the parsing of XML meta attributes.
- The multimedia library has been integrated into the official Bigloo branch.
- The I/O system has been fully re-implemented for efficiency. In particular, from the version 3.1b, outputs use a novel buffers, a new function, `send-file` has been added which avoids creating input ports for transferring files, and the sockets API has been extended with several facilities and configuration flags. These modifications have significantly improved the performance of the HOP web server [21].

The Bigloo branch 3.2x is in preparation. It is planned to release the version 3.2a in late December or early January. This version contains several new compiler optimizations and a faster runtime system.

6.4. Web programming

Participants: Anne-Marie Burns, Florian Loitsch, Manuel Serrano.

6.4.1. Hop

HOP (`http://hop.inria.fr`) [12], [13] a new Software Development Kit for the Web 2.0. It relies on a new higher-order language for programming interactive web applications such as multimedia applications (web galleries, music players, ...), office applications (web agendas, mail clients, ...), ubiquitous domotics, etc. HOP can be viewed as a replacement for traditional graphical toolkits. The HOP software development kit contains two compilers, one interpreter, and a Web server.

Software distribution: HOP 1.9.0, the first release of the new 1.9.x branch, has been released in September. This new branch brings so many fixes, optimizations, and novelties that it is not possible to list them all here. Let us only sum up the highlights of this new version:

- A new framework for server pipeline has been implemented. It allows system administrators to select a pre-existing pipeline implementation as well as to provide their own pipeline scheduler when starting HOP. This flexibility is useful for testing new strategies and for conducting experiments on the impact of the concurrency model on the overall HOP performance.
- The memory consumption of the server has been drastically reduced. The versions 1.9.x allocate about a tenth of the versions 1.8.x.
- A new weblet, `hzbuilder`, has been added to the set of standard weblets. It implements a wizard which assists users for creating new weblets.
- A new API and implementation for server events [17].

In addition to maintaining and improving HOP, we have started to study two application fields for the system, namely multimedia applications and domotics.

Multimedia Applications: Using HOP we have implemented a *ubiquitous home media center* named HopAudio. Taking advantage of the ubiquity of the Web, this application implements features that other programs used for playing music rarely propose. HopAudio can use many sources of music and radios and it can control several output speakers. All the electronic devices that can run a HOP broker (i.e., a dedicated Web server) can be used to *serve* musical content. All the devices that can run a stock web browser can be used to *control* the music being played back. HopAudio can be considered as a *realistic prototype*. It is operational and used on a daily basis although some implementation details must still be polished and some minor features must still be added.

Multimedia applications have specific demands on programming languages. For instance, the runtime systems of the languages must obviously provide means for playing music! This was missing in the previous versions of HOP and has been added to the system. In addition to these domain-specific extensions, we have found it useful to provide the language with a new communication channel between servers and clients that allows the former to *push* information to the latter. This feature is not specific to multimedia applications and can be used in many different contexts, for instance, when servers and clients must be synchronized. In HopAudio, it is used to synchronize the graphical user interfaces of all the devices. HopAudio will be presented at the conference MMCN'09 [17].

Domotics Applications: X10 is an open protocol for personal domotics intended to be used for controlling the lights and appliances of a house. To be controlled by X10, the house lights and the other appliances need to be connected to X10 modules. These modules are either plugged into regular wall outlets or installed on the electrical house network in the same way regular switches and outlets would be. X10 modules are relatively cheap equipments, each costing about ten to twenty euros.

In order to control X10 modules by means of HOP programs, we have created a dedicated library: HopX10. The HopX10 library is composed of three threads and a database. The database is used to store the X10 network configuration in terms of modules disposition in the house and modules estimated status. Module status are estimated in function of the instructions sent to them, as mentioned earlier, X10 generally does not provide a way to confirm the effective module status. The three threads are responsible for:

1. sending X10 instructions on user or system automatic response request,
2. dealing with status change by updating the database and launching automatic responses when needed
3. listening for inputs or requests from the X10 controller.

The main motivation of this work was to demonstrate that by combining X10 and HOP it is easy to implement ubiquitous domotics applications where a user can control all sorts of appliances of his house using, for instance, his cell phone. Using the new HopX10 library we have built such an application that has been presented during the conference on the *Internet of things*.

6.4.2. Scm2JS

We continued the development of SCM2JS, our Scheme to JavaScript compiler. Much time and effort has been spent on increasing its speed and reducing its memory consumption. These improvements helped to reduce Hop's initial memory usage (from start to a stable state) of about 20MB. We achieved these important savings by switching from an ad-hoc implementation of prototype-based objects to Bigloo's native object system, and by lazily allocating containers for macros and runtime-variables.
We furthermore cleaned the module-system and introduced qualified names. The new module-system is based on Bigloo modules and thus integrates much better with Hop. These changes are however disruptive (i.e. they may break old code).
Finally we have continued improving our `call/cc` and tail-call system. Both subjects have become major parts of F. Loitsch's PhD thesis and have thus been explored in detail. In particular the thesis contains a simplified algorithm, called Replay-C, of the continuation technique, as well as a proof of its correctness.

7. Contracts and Grants with Industry

7.1. CRE France-Télécom R&D

A CRE (contract for external research) started last 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. 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.

8.1.2. 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.

8.1.3. Geocal

working group of the GDR Informatique Mathématique. The partners are IML, LIF (Marseilles), IRIT (Toulouse), LIP, ICJ (Lyon), LAMA (Chambéry), Calligrame, Carte (LORIA Nancy), PPS (Paris), LIPN (Villetaneuse), LSV (Cachan), CEA, Logical (Saclay), Sardes (INRIA Rhone-Alpes), UNSA, Mimosa (Sophia Antipolis).

9. Dissemination

9.1. Seminars and conferences

Frédéric Boussinot presented FunLoft at MULTICOMP’08 in Göteborg in conjunction with the 3rd International Conference on High-Performance Embedded Architectures and Compilers (HiPEAC). He took part in the meeting of the ANR project PARSEC in Rennes. FunLoft has also been presented at the Inria Working group on Massively Multiprocessor and Multicore Computers (animated by Marc Shapiro and Albert Cohen).

Gérard Boudol presented his work on typing termination in higher-order imperative languages (published at the CONCUR’07 conference) in a seminar at École Normale Supérieure, Lyon, and in a workshop of the GEOCAL-LAC groups at the University of Paris Nord. He participated in the ETAPS’08 Conferences in Budapest, where he gave a talk on [15]. He was invited by Lisbon University, and gave there two seminars, on [15] and [14]. He participated in the PARSEC meeting in Rennes, where the work [20] was presented by Gustavo Petri. He attended the colloquium in honour of Ugo Montanari (on the occasion of his 65th birthday) at the Science Faculty of Pisa University. He was invited for three months by the Fondation Science Mathématiques de Paris, visiting the PPS Lab, where he gave a talk on [15] and on [20]. He also presented this last work in a seminar given at the joint INRIA-Microsoft Laboratory. He participated in the conference ESORICS’08 and the workshop FAST’08 in Malaga, where he presented [14]. He participated in the conferences FSTTCS and APLAS held in Bangalore. He was invited for a workshop on Perspectives in Concurrency Theory in Chennai, where he gave a talk on [20].

Ilaria Castellani was a PC member of the international conference CONCUR’08. In April 2008 she visited Ana Almeida Matos, in Cabanas, Portugal, to work on the subject of "type systems with semantic tests" together with Cédric Fournet and Tamara Rezk. She participated to the 4th meeting of the ANR project PARSEC at Irisa, Rennes, on June 9, 2008. On June 12, 2008, she attended the colloquium held in honour of Ugo Montanari (on the occasion of his 65th birthday) at the Science Faculty of Pisa University (for which she had coauthored the contribution [19]). She attended the conference FMCO’08, at INRIA Sophia Antipolis, in October 2008.

Manuel Serrano gave several presentations of the HOP system. In particular, invited by the Exalead company, he gave a presentation on Ubiquitous Multimedia on the Web with Hop. Manuel Serrano and Anne Marie Burns gave a demonstration on Ubiquitous Home Automation at the European 2008 conference on the *Internet of things*. Manuel Serrano gave a presentation on Multimedia applications on the Web with HOP in Tokyo, during the PUCC annual meeting. He gave a lightning talk about HOP at the W3C TPAC 2008 conference. He gave a presentation about multimedia on Web at the Microsoft/Inria join laboratory.

9.2. Animation

Gérard Boudol is the coordinator of the ANR SETIN project PARSEC. He was an examiner of the PhD Thesis of Alejandro Russo (Chalmers University, Göteborg).

Manuel Serrano wrote with Christian Queinnec from University Pierre et Marie Curie two scientific popularization papers published in the French magazine *Programmez!* [12], [13].

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 summer internship of Cyprien Nicolas, an undergraduate student, which during two months worked on developing a Multimedia application with HOP. Manuel Serrano supervised the Master internship of Olivier Plancade from February to August. O. Plancade studied the compilation of HOP to ActionScript, the language used in the Adobe Flash platform.

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° 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° 2, May 2000, p. 1–43.

Year Publications

Doctoral Dissertations and Habilitation Theses

[10] S. EPARDAUD. *ULM, un langage pour la mobilité*, Thèse de doctorat d'université, Université de Nice, Nice, Jan 2008.

Articles in International Peer-Reviewed Journal

[11] G. BOUDOL. *On strong normalization and type inference in the intersection type discipline*, in "Theoretical Computer Science", vol. 398, n^{o} 1, 2008, p. 63-81.

Articles in Non Peer-Reviewed Journal

[12] M. SERRANO, C. QUEINNEC. *Hop, un langage de programmation pour le Web*, in "Programmez!", n^{o} 104, Jan 2008.

[13] M. SERRANO, C. QUEINNEC. *Une galerie de photos sur le Web avec HOP*, in "Programmez!", n^{o} 105, Feb 2008.

International Peer-Reviewed Conference/Proceedings

[14] G. BOUDOL. *Secure information flow as a safety property*, in "FAST'08", 2008.

[15] G. BOUDOL. *Typing safe deallocation*, in "ESOP'08", Lecture Notes in Computer Science, vol. 4960, 2008, p. 116-130.

[16] F. BOUSSINOT, F. DABROWSKI. *Safe Reactive Programming: the FunLoft Proposal*, in "Proc. of MULTIPROG - First Workshop on Programmability Issues for Multi-Core Computers", Göteborg, January 2008.

[17] M. SERRANO. *Anatomy of a Ubiquitous Media Center*, in "Proceedings of the Sixteenth Annual Multimedia Computing and Networking (MMCN'09), San Jose, CA, USA", Jan 2009.

Scientific Books (or Scientific Book chapters)

[18] F. LOITSCH, M. SERRANO. *Hop Client-Side Compilation*, in "Trends in Functional Programming", vol. 8, n^{o} chap. 9, Moraz M. T. editor, Seton Hall University, Intellect Bristol, UK/Chicago, USA, 2008, p. 141–158.

Other Publications

[19] G. BOUDOL, I. CASTELLANI, M. HENNESSY, M. NIELSEN, G. WINSKEL. *Twenty Years on: Reflections on the CEDISYS Project. Combining True Concurrency with Process Algebra*, in "Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday", P. DEGANO, R. D. NICOLA, J. MESEGUER (editors), Lecture Notes in Computer Science, vol. 5065, Springer, 2008.

[20] G. BOUDOL, G. PETRI. *Relaxed memory models: an operational approach*, 2008, accepted for publication at POPL'09.

[21] M. SERRANO. *Hop, a Fast Server for the Diffuse Web*, in "paper submitted", 2008.