INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team indes

Informatique Diffuse et Sécurisée

Sophia Antipolis - Méditerranée

Theme : Distributed Systems and Services

Activity Report

2009

Table of contents

1. Team

Research Scientist

Gérard Berry [Research Director, Inria, HdR]
Gérard Boudol [Research Director, Inria, HdR]
Frédéric Boussinot [Research Director, CMA, HdR]
Ilaria Castellani [Research Scientist, Inria]
Tamara Rezk [Research Scientist, Inria]
Bernard Serpette [Research Scientist, Inria]
Manuel Serrano [Team Leader, Research Director, Inria, HdR]

Faculty Member

Christian Queinnec [Professor, University Pierre et Marie Curie - Paris 6, HdR]

Technical Staff

Marcos Dione [Inria, from October 1]
Florian Loitsch [MENRT, from March 13, till December 31]
Cyprien Nicolas [Inria, from March 1]

PhD Student

Florian Loitsch [MENRT, till March 13]
Zhengqin Luo [MENRT]
Gustavo Petri [IP Mobius]

Administrative Assistant

Anais Cassino [Inria]

2. Overall Objectives

2.1. Overall Objectives

The goal of the Indes team is to study models for diffuse computing and develop languages for secure diffuse applications. Diffuse applications, of which Web 2.0 applications are a notable example, are the new applications emerging from the convergence of broad network accessibility, rich personal digital environment, and vast sources of information. Strong security guarantees are required for these applications, which intrinsically rely on sharing private information over networks of mutually distrustful nodes connected by unreliable media.

Diffuse computing requires an original combination of nearly all previous computing paradigms, ranging from classical sequential computing to parallel and concurrent computing in both their synchronous / reactive and asynchronous variants. It also benefits from the recent advances in mobile computing, since devices involved in diffuse applications are often mobile or portable.

The Indes team contributes to the whole chain of research on models and languages for diffuse computing, going from the study of foundational models and formal semantics to the design and implementation of new languages to be put to work on concrete applications. Emphasis is placed on correct-by-construction mechanisms to guarantee correct, efficient and secure implementation of high-level programs. The research are partly inspired by and built around Hop, the web programming model recently proposed by the former Mimosa team, which takes the web as its execution platform and targets interactive and multimedia applications.

3. Scientific Foundations

3.1. Parallelism, concurrency, and distribution

Concurrency magagement is at the heart of diffuse programming. Since the execution platforms are highly heterogeneous, many different concurrency principles and models may be at stake. Asynchronous concurrency is the basis of shared-memory process handling within multiprocessor or multicore computers, of direct or fifo-based message passing in distributed networks, and of fifo- or interrupt-based event handling in web-based human-machine interaction or sensor handling. Synchronous or quasi-synchronous concurrency is the basis of signal processing, of real-time control, and of safety-critical information acquisition and display. Interfacing existing devices based on these different concurrency principles within HOP or other diffuse programming languages will require better understanding of the underlying concurrency models and of the way they can nicely cooperate, a currently ill-resolved problem.

3.2. Web and functional programming

We are studying new paradigms for programming Web applications that rely on multi-tier functional programming [4]. This research has been initiated in the MIMOSA team, the predecessor of the INDES team, where we have created the new Web programming environment named HOP. It relies on a single formalism for programming the server-side and the client-side of the applications as well as for configuring the execution engine.

HOP is a functional language based on the SCHEME programming language. That is, it is a strict functional language, fully polymorphic, supporting side effects, and dynamically type-checked. HOP is implemented has an extension of the BIGLOO compiler that we develop [5]. In the past, we have extensively studied static analyses (type systems and inference, abstract interpretations, as well as classical compiler optimizations) to improve the efficiency of compilation in both space and time.

3.3. Security of diffuse programs

The main goal of our security research is to provide scalable and rigorous language-based techniques that can be integrated into multi-tier compilers to enforce security of diffuse programs. Research on language-based security has been carried on before in both the MIMOSA [2] and EVEREST [1] teams). In particular previous research has focused on controlling information flow to ensure confidentiality.

Typical language-based solutions to these problems are founded on static analysis, logics, provable cryptography, and compilers that generate correct code by construction [3]. Relying on the multi-tier programming language HOP that tames the complexity of writing and analysing secure diffuse applications, we are studying language-based solutions to prominent web security problems such as code injection and cross-site scripting, to name a few.

4. Application Domains

4.1. Web programming

Along with games, multimedia applications, electronic commerce, 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. The web constitutes an infrastructure more versatile, polymorphic, and open, in other words, more powerful, than any dedicated network previously invented. For this very reason, it is likely than most of the computer programs we will write in the future, for professional purposes as well as for our own needs, will extensively rely on the web.

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-tier 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.

4.2. Multimedia

Electronic equipments are less and less expensive and more and more widely spread out. Nowadays, in industrial countries, computers are almost as popular as TV sets. Today, almost everybody owns a mobile phone. Many are equipped with a GPS or a PDA. Modem, routers, NASes and other network appliances are also commonly used, although they are sometimes sealed under proprietary packaging such as the Livebox or the Freebox. Most of us evolve in an electronic environment which is rich but which is also populated with mostly isolated devices.

The first multimedia applications on the web have appeared with the Web 2.0. The most famous ones are Flickr, YouTube, or Deezer. All these applications rely on the same principle: they allow roaming users to access the various multimedia resources available all over the Internet via their web browser. The convergence between our new electronic environment and the multimedia facilities offered by the web will allow engineers to create new applications. However, since these applications are complex to implement this will not happen until appropriate languages and tools are available. In the Indes team, we develop compilers, systems, and libraries that address this problem.

4.3. House Automation

The web is the de facto standard of communication for heterogeneous devices. The number of devices able to access the web is permanently increasing. Nowadays, even our mobile phones can access the web. Tomorrow it could even be the turn of our wristwatches! The web hence constitutes a compelling architecture for developing applications relying on the "ambient" computing facilities. However, since current programming languages do not allow us to develop easily these applications, ambient computing is currently based on ad-hoc solutions. Programming ambient computing via the web is still to be explored. The tools developed in the Indes team allow us to build prototypes of a web-based home automation platform. For instance, we experiment with controlling heaters, air-conditioners, and electronic shutters with our mobile phones using web GUIs.

5. Software

5.1. Introduction

Most INDES software packages, 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 software packages can be downloaded from the INDES Web site:
http://www-sop.inria.fr/teams/indes

5.2. Functional programming

Participants: Frédéric Boussinot, Florian Loitsch, Zhengqin Luo, Bernard Serpette, Manuel Serrano.

5.2.1. The Bigloo compiler

The programming environment for the Bigloo compiler [5] is available on the INRIA Web site at the following URL: http://www-sop.inria.fr/teams/indes/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 it 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.

5.2.2. The FunLoft language

FunLoft (described in http://www-sop.inria.fr/indes/rp/FunLoft/) is a new programming language in which the focus is put on safety and multicore.

FunLoft is built on the model of FairThreads which makes concurrent programming simpler than usual preemptive-based techniques by providing a framework with a clear and sound semantics. FunLoft is designed with the following objectives:

- provide a safe language, in which, for example, data-races are impossible.
- control the use of resources (CPU and memory), for example, memory leaks cannot occur in FunLoft programs, which always react in finite time.
- have an efficient implementation which can deal with large numbers of concurrent components.
- benefit from the real parallelism offered by multicore machines.

A first experimental version of the compiler is available on the Reactive Programming site http://www-sop.inria.fr/indes/rp/). Several benchmarks are given, including cellular automata and simulation of colliding particles.

5.2.3. ULM

ULM (*Un langage pour la mobilité*) is a language for mobility that was developed in the MIMOSA 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.

5.3. Web programming

Participants: Florian Loitsch, Manuel Serrano.

5.3.1. 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 streams, 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.

5.3.2. 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

5.4. Old software

5.4.1. 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.

6. New Results

6.1. Security

Participants: Gérard Boudol, Ilaria Castellani, Zhengqin Luo, Tamara Rezk.

6.1.1. Secure information flow in ULM

We have pursued and extended our study of secure information flow as a safety property (paper by Boudol at FAST'08). We have applied this approach [20] to a significative fragment of the ULM language, namely the functional, imperative and concurrent fragment. This language follows the reactive style developed by Boussinot, where cooperative threads communicate and synchronize by means of broadcast events. Since these events determine for a part the flow of control of an ULM program, they convey some knowledge of the security level the program has at the point where events are emitted or received. Then this knowledge has to be controlled in order to ensure that the flow of information along the execution of a program is secure. This is done by means of a run-time monitoring mechanism. Moreover, a static analysis (type and effect system) of programs is designed, and proved to guarantee that the monitoring always succeeds. In other words, typable programs are secure, in the sense that their monitored execution does not encounter any security violation. The monitoring mechanism has been implemented, and some experimentations with scenarios provided by France Télécom R&D are planned.

6.1.2. A Security-Preserving Compiler for Distributed Programs

In language-based security, confidentiality and integrity policies conveniently specify the permitted flows of information between different parts of a program with diverse levels of trust. These policies enable a simple treatment of security, and they can often be verified by typing. However, their enforcement in concrete systems involves delicate compilation issues.

We consider cryptographic enforcement mechanisms for distributed programs with untrusted components. In source programs, security depends on an abstract information-flow policy for accessing the shared memory. In their implementations, shared memory is unprotected and security depends instead on encryption and signing.

In [3] we develop a cryptographic type system for a target probabilistic language. Our typing rules enforce the correct usage of cryptographic primitives against active adversaries; from an information-flow viewpoint, they capture controlled forms of robust declassification and endorsement.

In [13], we develop a uniform language-based model of security, ranging from computational non-interference for probabilistic programs down to standard cryptographic hypotheses. The theory has been implemented in a prototype called “CFlow”. It has been coded in F#. The code is under the terms of the CeCILL-B license.

6.1.3. Proof Carrying Code

Proof Carrying Code provides trust in mobile code by requiring certificates that ensure the code adherence to specific conditions. The prominent approach to generate certificates for compiled code is Certifying Compilation, that automatically generates certificates for simple safety properties.

In [10] we propose a security certifying compilation for multithreaded programs with secure information flow.

In [8] we present Certificate Translation, a novel extension for standard compilers that automatically transforms formal proofs for more expressive and complex properties of the source program to certificates for the compiled code. We outline the principles of certificate translation, instantiated for a non optimizing compiler and for standard compiler optimizations in the context of an intermediate RTL Language.

6.1.4. Secure session calculi

This work has been mainly carried out within the MATYSS project, whose goal was to study type systems for safe and secure sessions. A session is an abstraction for various forms of “structured communication” which may occur in a parallel and distributed computing environment. Examples of sessions are a client-service negotiation, a financial transaction, or a multiparty interaction among different services within a web application. Language-based support for sessions has now become the subject of active research. Primitives for enabling programmers to code sessions in a flexible way, as well as type systems (session types) ensuring the compliance of programs to session specifications, have been studied for various calculi and languages in the last decade. The key properties ensured by session types are communication safety, namely the consistency of the communication patterns exhibited by the various partners (implying the absence of communication errors at run time), and progress, assuring the absence of deadlock. On the other hand, security properties such as confidentiality have so far received little attention within the session type community. In collaboration with our MATYSS partners in Torino, we have addressed the question of incorporating access control and secure information flow requirements within session types, in the setting of a name-passing calculus akin to the π-calculus, with asynchronous communication and multiparty sessions.

In [19] we consider a calculus of multiparty sessions with delegation, enriched with security levels for both participants and data. A suitable type system ensures access control, namely that each participant can only receive data of security levels less than or equal to its own security level. For instance, in a well-typed session involving a Customer, a Seller and a Bank, the secret credit card number of the Customer is communicated to the Bank, but not to the Seller. This is obtained also by making delegation explicit in the typing of the delegated session channel. Moreover, the type system prevents undue flows of information via the selection and branching constructs of the language. This work reveals an interesting interplay between the constraints used in security types and those used in session types to ensure properties like communication safety and progress.

6.2. Models and semantics

Participants: Gérard Berry, Gérard Boudol, Gustavo Petri, Christian Queinnec, Manuel Serrano.

6.2.1. Semantics of concurrent programming

We are investigating the theory of multithreading. Multithreading is a form of concurrent programming where sequential programs (threads) run in parallel and communicate via a shared mutable store. Parallel execution is usually ruled according to some scheduling policy (in our work we consider preemptive, non-deterministic scheduling, that is the standard interleaving model), and controlled, from the threads, by means of synchronization constructs, like acquiring and releasing locks. It is well-known that this model is not correctly implemented in optimized execution platforms introducing some further parallelism in the computation, for instance in the accesses to the shared memory, like in weak memory models. Our work of last year on the write- buffering memory model has been published this year, see [12]. We have extended our approach to capture more relaxed memory models and, more generally, relaxed execution models where pieces of code can be executed in advance (or in parallel with the rest of the program), like with the write-buffers memory model where the updates to the memory are issued, and placed in buffers, while pursuing with the rest of the program. Such optimization mechanisms are known as *speculative computation*. We have proposed a formalization of a fully general notion of speculative computation, including branch prediction and value prediction. This is done in [17] for an expressive language (CoreML, or CoreScheme, that is the functional and imperative fragment of these languages, enriched with concurrent programming constructs) by means of Berry and Lévy's equivalence by permutation of computations. We have to define in particular a notion of *valid* speculation, since some speculative computation may be exploring the wrong branch in a conditional branching sub-program, the condition of which is not yet determined. We have shown that, for programs enjoying the property that their speculative execution does not create data-races (hence in particular for sequential programs), the speculative semantics correctly implements the standard interleaving semantics. We also propose in [18] a source programming language where a distinction is explicitly made between shared and private variables, and where we introduce a synchronization construct for temporarily having exclusive access to a shared variable. We show that, guided by a type and effect system guaranteeing that a private variable is indeed not shared, we can translate the (typable) programs of this language into programs that are correctly executed in the speculative semantics.

Regarding the latter core language for multithreaded programming, we have investigated another problem, namely the one of deadlocks. It is well-known that synchronization is needed to ensure exclusive access to shared data (like in updating a bank account, for instance), but that synchronization usually introduces deadlocks, a configuration where some threads cannot make any progress, being circularly blocked in waiting for a resource held by another thread. This is a kind of error that must be dealt with, and there are three techniques to do so: deadlock prevention (usually by means of static analysis or verification), deadlock detection and recovery (by means of a monitoring mechanism involving roll-back), and deadlock avoidance. Surprisingly enough, the last technique has not been explored at the programming language level. In [11], we design, for a language similar to the one of [18], a type and effect system where the locks that are anticipated to be acquired are recorded in the effect of a program. Then, guided by this analysis, we design a semantics where a lock is acquired only if no other lock that could be acquired in the future, that is, a lock in the effect of the program, is already held by another thread. We show that this provides, for typable programs, a deadlock-free semantics.

6.2.2. A multi-tier semantics for Hop

HOP is a multi-tier programming language where a single program specifies servers and clients behaviors altogether. Hop adheres to the standard web programming style where servers elaborate HTML pages containing JavaScript code. This JavaScript code responds locally to user's interactions but also (following the so-called Ajax style) requests services from remote servers. These services bring back new HTML fragments containing additional JavaScript code replacing or modifying the state of the client.

We have proposed a continuation-based denotational semantics for a sequential subset of Hop. Though restricted to a single server and a single client, this semantics takes into account the nature of the web where the server elaborates some JavaScript code to be run in the client's browser. This new client-code dynamically requests services from the server which, again, elaborates new JavaScript code to be run in the client's browser.

This semantics details the programming model advocated by Hop and provides a sound basis for future studies such as web continuations and concurrency. The semantics has been presented at the event dedicated to Mitch Wand that took place in Boston in August.

6.3. Functional and concurrent programming

Participants: Frédéric Boussinot, Ilaria Castellani, Florian Loitsch, Cyprien Nicolas, Manuel Serrano.

6.3.1. Bigloo

Software distribution: The new Bigloo branch 3.2 has been released this year. This branch has introduced the following novelties:

Bigloo packaging: In order to spread Bigloo and Hop more easily, we are currently working on building the packages for several Linux distributions. We have thus accommodated the Bigloo install process to these distributions. So far, Bigloo is now compatible with the demanding Debian distribution model.

New FairThread support: The previous releases of the Bigloo compiler, prior to 3.2c, come with two multi-threading APIs, named pthread and fthread. The former is an implementation of the Posix's threading library, based on the primitives provided by the LibC, or constructed over Java or the .NET thread system, the latter relies on the same base, but offered cooperative scheduling of threads. The developer had the choice to use one of these APIs to write multi-threaded programs in Bigloo.

In the new Bigloo version the developers may use both APIs at the same time, which means that they are able to embed one fthread scheduler into a pthread. We also wanted to rewrite the fthread API in Scheme only, and make calls to the pthread API to manage fthread life cycle and scheduling (which means locks in practice). These changes have been included in release 3.2c, where fthreads are now thread-safe with pthreads.

A new fast multi-threaded DNS support: The traditional BSD implementation of sockets uses function that are not thread-safe (`gethostbyname`, `gethostbyaddr`, ...). Hence, the only way to use these functions in a multi-threaded context is to protect each call with mutexes. Unfortunately, this is intrinsically inefficient. If at some point of the execution a DNS is answering to one host, all the other running threads that are also needing information from the DNS will get blocked too. In the context of HOP this may reduce dramatically the overall performance of the whole server. To get rid of this problem, we have implemented a new cache DNS for BIGLOO. It is thread-safe and uses no lock. It is based on the POSIX-1 functions `getaddrinfo`. This new implementation has significantly increased the performance of the HOP web server.

Cryptography: We have implemented a cryptography library for BIGLOO. This library supports most of the common cryptographic primitives. Among the public-key cryptography systems it includes RSA, DSA, and ElGamal. It furthermore features the following block-ciphers: AES, CAST128, DES, and IDEA.

Floating-point: We have developed a fast binary-to-decimal floating-point conversion. It is up to 5 times faster than previously existing techniques. We have explored different use-cases and implemented different variants. The fastest and simplest solution produces decimal representations that might be unnecessarily long. Evolutions trim the output to produce shorter outputs. The algorithm can furthermore be adapted to produce the shortest possible output, but in this case it will reject a small percentage (about 0.6%) of all possible IEEE 754 floating-point numbers.

6.3.2. FunLoft

This year, the effort has been concentrated on developing several aspects of FunLoft, according to the objectives of the PARTOUT project.

Efficient Programming. The implementation in the FunLoft compiler of the garbage collection RGC (for reactive garbage collector) has been completed. RGC is adapted to the synchronous/reactive framework and is partially based on a reference counting approach. RGC is tested on a list of benchmarks, including a graphical simulation of colliding particles, using multicore. RGC is presently only described in an internal report.

Distributed Programming. Primitives for distributed programming have been introduced in FunLoft, and tested on a little "ping-pong" demo. This work is currently under development. We use HOP as a "proxy" for distribution and, in the course of the project PARTOUT, we plan to propose HOP as a general interface for SugarCubes and ReactiveML.

Safe Programming. FunLoft proposes ways of programming without data-races which preserve the natural atomicity of user programs within a multicore framework. Basically, FunLoft states a memory separation property which is statically checked. A paper considering a kernel in which the soundness of the approach could be formally proved is under work.

Dynamic Aspects. A small language to program reactive scripts has been designed and a first experimental version of it has been implemented in FunLoft. The scripts actually do not define new functions for data manipulations, but rather just describe some kind of "orchestration" of calls of already existing functions. The production of SugarCubes code is also planned. A paper is in the course of being written on that matter.

6.4. Web programming

Participants: Marcos Dione, Florian Loitsch, Zhengqin Luo, Cyprien Nicolas, Tamara Rezk, Bernard Serpette, Manuel Serrano.

6.4.1. Hop

Software distribution: Two new major branches have been released this year: HOP 1.10.x and 1.11.x. Branch 1.10.x has carried new APIs and new features. HOP 1.11.x has focused on the implementation. In particular it has deployed a new version of the client-side compiler.

Multimedia Applications: We have completed HopAudio, the *ubiquitous home media center* started in 2008 and presented at the conference MMCN'09 [14]. 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.

Fast server events: *server events*, or *server push* is a central element of reactive Web applications. They enable servers to notify clients when some informations need to be updated. Server events simplify the programming of applications and they avoid inefficient client pulls that clutter network traffic. Unfortunately, HTTP is not well suited for server events. According to HTTP, only clients may initiate communications and the protocol is stateless. As a consequence, it is difficult to implement server push efficiently. HOP now contains three different implementations that are chosen dynamically according to the characteristics of the client the program executes on.

- if the client supports *multipart* `XMLHttpRequest` then when the program is installed on the client, a background request is spawned. This opens a tunnel (i.e., a socket) between the client and the server that lets the latter send data piece by piece. More precisely, each time the server needs to send a signal to a client, it uses that tunnel and sends a new information. This implementation is efficient because it uses only one communication channel and because the browser natively implements the wake up of the client. Unfortunately, it is not very portable.
- If the client supports *Flash*, then a new socket is opened when the program is installed on the client. This socket is used by the server to push data to the client. This method is reasonably efficient but creating the socket is slow and complex because it requires JavaScript and ActionScript to communicate and synchronize. Obviously, it also requires the browser to support Flash, which is a severe portability issue.
- If none of the previous techniques can be deployed, then HOP falls back to a generic method which consists in creating a background request per signal. The client spawns a request. When the server sends a signal, it writes the date on the connection created for the request and closes the request. Upon reception, the client parses the data received and spawns a new request. This method is slow because it keeps opening sockets, and unreliable because there is a time lag in which the client and the server are disconnected. However, it is portable because it is supported by all browsers.

Hop, a Fast Server for the Diffuse Web: Diffuse Web applications have similarities with Web 2.0 applications: first, they rely on fast bi-directional interactions between servers and clients, and second, they make extensive use of non-cachable dynamic contents. On the other hand, diffuse applications have also an important difference with respect to traditional Web applications: they generally do not need to deal with a huge number of simultaneous users. That is, diffuse Web applications are built on top of standard technologies but they use them differently. Therefore they demand different optimizations and tunings.
The HOP software development kit contains two compilers, one interpreter, and a bootstrapped Web server. That is, the HOP Web server is implemented in HOP. We have implemented a new strategy that allows HOP to dramatically outperform the popular mainstream Web servers for delivering dynamic contents. Contrary to most servers, HOP delivers static and dynamic contents at a comparable pace. This has been described in an invited paper presented at the COORDINATION'09 conference [9]
HSS css compiler: HSS is a compiler for CSS. It can be either used as a stand-alone HSS-to-CSS compiler in the goal of enriching CSS with user defined variables, functions, and element types. It can also be used with the Hop web development kit in which case, working hand in hand with the Hop programming language, it can be used to implement *skinning* or *theming* of web applications. HSS has been described in a paper currently submitted [22].

6.4.2. Scm2Js

We have continued the development of SCM2JS, our Scheme to JavaScript compiler. In particular we improved the code generation. A pretty-printing pass produces clean code, or, alternatively, obfuscated short code. We furthermore worked on error handling. Compared to previous versions, error messages contain much more useful information that helps developers debug their program. In a similar vein we have introduced runtime-checks (activated by the `-g` flag).

6.4.3. Ordered networks

A network is a graph of *nodes* where edges represent physical or logical connections. The main role of the network is to dispatch messages between nodes, a mechanism known as the *routing* process. To locate the destination of a message, each node has its own *identifiers*, e.g. IP addresses are the identifiers for the Internet. In logical networks, nodes also *manage* a set of identifiers. Thus, such a network does not resolve only queries like *send a message to the node whose identifier is id*, but also more general queries like *send a message to the node who manages the identifier (or key) id*. In the networks we have studied, we force, by construction, the uniqueness of identifiers: an identifier is managed by one and only one node. Moreover, we want to guarantee that messages reach their destinations, that is, that the network does not lose messages during the routing process. The approach we have followed is to consider a hamiltonian cycle over the network: a cycle which traverses all nodes of the graph once and only once.

Hamiltonian cycles are generally extracted from an existing graph, which is known to be an NP-complete problem. In our case, the cycle is established at the beginning, when the network is initiated with a first node, and preserved during insertion and deletion of nodes. As we consider oriented graphs, each node is only required to maintain its *successor* in the cycle. It is particularly convenient to consider a strict order $<$ over nodes. Given this order, a finite set of nodes can be sorted, thus giving an Hamiltonian path. The cycle is then achieved by artificially connecting the two bounds of the path.
We have experimented several orders,

1. **natural**: the basic order on natural numbers,
2. **gray code**: also based on natural numbers, it forms a Hamiltonian cycle on a hypercube, where each bit is seen as one dimension,
3. **space-filling curve**: based on 2D points with the Hilbert curve,
4. **alphabetic**: the basic order on strings.

To avoid a routing proportional to the number of nodes, each node has a set of *perfect* identifiers used to build shortcuts over the hamiltonian cycle. The general aim is, having a logarithmic number of perfect identifiers per node, to insure a logarithmic routing. We have experimented several functions computing the perfect identifiers,

1. **exponential**: associated to the natural order, it gives the well established Chord network.
2. **symmetrical**: with 2D points, perfects are chosen in symmetric sub-squares.
3. **hypercube**: all identifiers differ by only one bit.

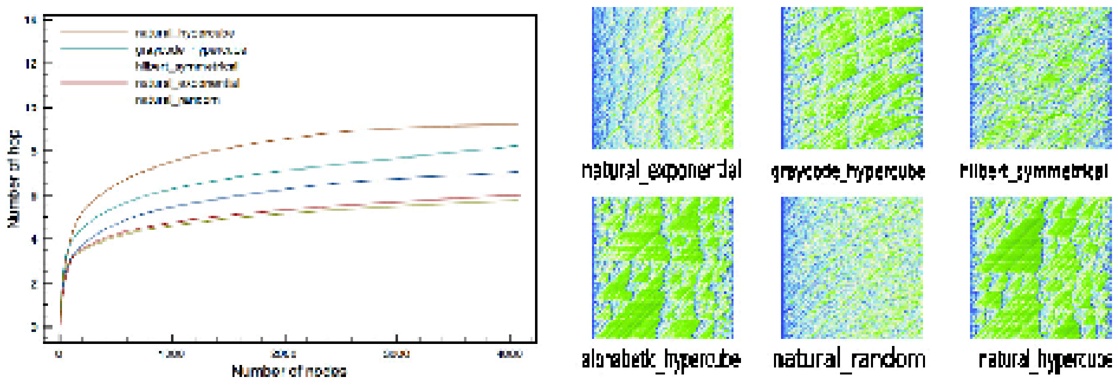

Figure 1. Performance of some ordered networks

The figure 1 shows the general performance of some ordered networks. All have the desired logarithmic behavior. Pictures on the right are snapshots of networks in stable state. The pixel at position *i,j* represents the number of nodes a message has to go through during the routing process (i.e. the number of hops), when this message starts from the *ith* node and wants to reach the *jth* successors of this node. Blue pixels represent a low number of hops, while green ones are used for a high number of hops. Thus the first column is in dark blue since a direct successor is always at one hop.
The attribution of a node for a perfect identifier, i.e. a shortcut, is done during the routing process. Some extra informations are added during message exchanges in order to share informations between neighbors in the networks. Th experiences we have made show that this strategy is relevant even in the case of a large amount of insertions and deletions of nodes.

Independently of the order and of the perfect identifier calculation, we have formalized the protocols for routing processes and for insertion and deletion of nodes. We have partially proven that messages reach their destination, as well as the absence of dead-locks and live-locks.

7. Contracts and Grants with Industry

7.1. Contracts and Grants with Industry

7.1.1. CRE France-Télécom R-D

A CRE (contract for external research) started in 2006, funded by France-Télécom R&D on “Analysis of security properties for global programming frameworks”. The initial duration of the project was 3 years. It has then been extended to 3.5 years. The total funding is 120 kEuros. The purpose of the project was initially to support the research done in the former MIMOSA team 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. The INDES team is now in charge of this project that will end in December 2009.

7.1.2. DGE SmartImmo

The SmartImmo DGE project of the world class ICT cluster SCS (Solutions Communicantes Sécurisées) started in 2009. It aimed at controlling and reducing the costs associated with building management. The duration of the project is 2 years, and the funding of 75 kEuros. In the context of this project, the INDES team will focus on making the Hop Web development kit compatible with industrial house automation systems.

8. Other Grants and Activities

8.1. National initiatives

8.1.1. STIC-AmSud FMCRYPTO

The FMCRYPTO project (for “Formal Methods for Cryptographically Secure Distributed Computations”) is funded by the STIC-AmSud programme for 2 years, starting January 2009. The partners of this project are the INDES team at INRIA (coordinator), Universidade Estadual de Campinas in Brazil, Universidad de Chile in Chile, and Universidad de la Republica in Uruguay.

8.1.2. COLOR project MATYSS

The one-year project MATYSS (Models And TYpes for Secure Sessions) is funded by the INRIA Sophia Antipolis COLOR programme. The partners of this project are the team INDES (coordinator) and the “Semantics and Logic of Computation” group at the Computer Science Department of the University of Torino.

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 INDES formerly 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.

8.1.4. ANR DEFIS ParTout

The PARTOUT project (PARTOUT = PARallélisme parTOUT) is funded by the ANR DEFIS programme for 4 years, starting January 2009 (ANR-08-EMER-010). The partners of this project are the teams INDES (coordinator), CNAM/Cédric, and LRI, Université d’Orsay.

8.1.5. ANR DEFIS PWD

The PWD project (for "Programmation du Web diffus") has been funded by the ANR Défis programme for 4 years, starting November 2009. The partners of this project are the teams INDES (coordinator), LIP6 at University Pierre et Marie Curie and PPS at University Denis Diderot.

9. Dissemination

9.1. Seminars and conferences

Gérard Berry participated in the Causality Workshop organized in Venezia by the Academia Europaea, of which he is a member. He participated and gave a talk in the ERCIM annual meeting, being a member of the ERCIM Advisory Committee. He gave several seminars in research or industrial conferences.

Gérard Boudol participated in the POPL'09 conference, where [12] was presented by Gustavo Petri. Gustavo also presented this work at the Journées INRIA Multicœur, which Boudol also attended. G. Boudol gave a three hours tutorial on Language Based Security at the first FMCrypto meeting in Campinas (Brazil). He also participated in the COORDINATION'09 conference. He presented [11] at a workshop of the PARSEC project and at the ICTAC'09 conference.

Ilaria Castellani presented her work on security for process calculi in the first MATYSS meeting at the University of Torino, on April 23. In September and December 2009 she spent two visits at the university of Torino, to pursue the MATYSS collaborative work [19].

Zhengqin Luo presented his work on cetified proof of anonymity protocols in the FMCrypto workshop, in Campinas (Brazil), April 2009. He also participated in the summer school “Logics and Languages for Reliability and Security” in Marktoberdorf (Germany), August 2009.

Gustavo Petri presented the work [12] in the journées “Informatique Massivement Multiprocesseur et Multicoeur” in Paris, February 2009. He was invited to present the same work at the Modeling and Verification group in May, at LIAFA, Paris. Later Gustavo was invited to the seminar “Design and Validation of Concurrent Systems” in Dagstuhl, September 2009, where he presented the work [17]

Christian Queinnec has been invited to the Symposium in Honor of Mitchell Wand (in cooperation with ACM SIGPLAN and co-located with Scheme and Functional Programming 2009) where he presented a semantics for Hop (a work done with Manuel Serrano) [21].

Tamara Rezk presented her work on the Cflow compiler in the ANR PARSEC meeting, in Paris. On February the 10th, she was invited by Bogdan Warinschi to spend a week at University of Bristol, where she gave a talk. On the 13th of the same month, she was invited by Kohei Honda to present her work at Queen Mary University, London. On April 23rd, Tamara participated in the first Matyss meeting in Torino, where she presented her work on “Security preserving compiler for multithreaded programs”. In march, she participated as track chair in ACM SAC 09. On April, Tamara was invited to attend the Spring School and French-Japanese collaboration workshop "Cosyproofs", on proofs for cryptography. This workshop was organized by Hubert Common-Lundh, Highashi Izu Peninsula in Japan. On April the 28th and the 30th, Tamara presented her work on anonymity protocols and the Cflow compiler in the FMCrypto workshop, organized in University of Campinas, Campinas, Brasil.

Manuel Serrano gave several presentations of the HOP system. In particular, he gave a keynote talk at the 11th international conference on Coordination Models and Languages about fast Web 2.0 server [9]. Manuel Serrano has been invited by professor J. Misra to spend a week at University of Texas where he gave a talk about Hop. He also gave another talk about Hop at IBM Watson, invited by O. Tardieu et J. Field. M. Serrano gave a presentation about multimedia applications in Hop at the MMCN'09 [14] conference in Jan Jose, USA. Manuel Serrano participated in the Inria-Alumni JAM session where he was involved in the panel about the Web.

9.2. Animation

Gérard Boudol is the coordinator of the ANR SETIN PARSEC project. He was a member of the Programme Committees of the conference COORDINATION'09 and the workshop SecCo'09. He was a member of the jury for the HDR (Habilitation à Diriger des Recherches) of Daniel Hirschkoff (ENS Lyon).

Frédéric Boussinot is the coordinator of the ANR DEFIS project PARTOUT.

Ilaria Castellani was the coordinator of the one-year COLOR project MATYSS.

Manuel Serrano is the coordinator of the ANR DEFIS project PWD. He serves the program committee of the IFL2009 conference.

9.3. Teaching

Gérard Berry is giving a course at Collège de France called "Penser, modéliser et maîtriser le calcul informatique" [15] (Thinking About, Modeling and Mastering computation), where he discusses numerous models of sequential, parallel or diffuse computation. Manuel Serrano will give a seminar about diffuse and HOP computation in the seminar series associated with this course.

Ilaria Castellani participated in the course "Secure diffuse computing" of the Ubinet master at Nice University (in October-November 2009), both as a lecturer and as the course responsible.

Christian Queinnec is teaching at UPMC two courses (20 hours each). The first one, a M1 course, is devoted to compilation. The second one, a M2 course, is centered on Web technologies, from database to client. Hop is presented within that course as an emerging new paradigm.

Tamara Rezk participated as a lecturer in the course "Secure Diffuse Computing" of the Ubinet master (October-November 2009), University of Nice.

Manuel Serrano supervised the summer internship of Pierre Karpman, an undergraduate student of Insa Lyon, who worked during two months on implementing an efficient interpreter for Hop. In October 2009, Manuel Serrano gave a 6 hour-course on multi-tier Web programming at the Master Ubinet of University of Nice.

10. Bibliography

Major publications by the team in recent years

[1] G. BARTHE, T. REZK, A. RUSSO, A. SABELFELD. *Security of Multithreaded Programs by Compilation*, in "ESORICS", 2007, p. 2-18.

[2] G. BOUDOL, I. CASTELLANI. *Noninterference for Concurrent Programs and Thread Systems*, in "Theoretical Computer Science", vol. 281, n^{o} 1, 2002, p. 109-130.

[3] C. FOURNET, T. REZK. *Cryptographically sound implementations for typed information-flow security*, in "Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008", 2008, p. 323-335.

[4] 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.

[5] 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

[6] M. KOLUNDZIJA. *Type Systems for Access Control and Information Flow in Programming Languages*, Universities of Nice - Sophia Antipolis and Torino, March 2009, Ph. D. Thesis.

[7] F. LOITSCH. *Scheme to JavaScript Compilation*, Universit de Nice - Sophia Antipolis, Mar 2009, Ph. D. Thesis.

Articles in International Peer-Reviewed Journal

[8] G. BARTHE, B. GRÉGOIRE, C. KUNZ, T. REZK. *Certificate translation for optimizing compilers*, in "ACM Trans. Program. Lang. Syst. (TOPLAS)", vol. 31, n^{o} 5, 2009.

Invited Conferences

[9] M. SERRANO. *HOP, a Fast Server for the Diffuse Web*, in "Invited paper of the 11th international conference on Coordination Models and Languages (COORDINATION'09), Lisbon, Portugal", Jun 2009.

International Peer-Reviewed Conference/Proceedings

[10] G. BARTHE, T. REZK, A. RUSSO, A. SABELFELD. *Security of Multithreaded Programs by Compilation*, in "ESORICS'07 Special Issue in ACM Transactions on Information and System Security (TISSEC)", 2009.

[11] G. BOUDOL. *A deadlock-free semantics for shared memory concurrency*, in "ICTAC'09", August 2009, p. 140-154, LNCS 5684.

[12] G. BOUDOL, G. PETRI. *Relaxed memory models: an operational approach*, in "POPL'09", 2009, p. 392-403.

[13] C. FOURNET, G. LE GUERNIC, T. REZK. *A Security-Preserving Compiler for Distributed Programs*, in "ACM Conference on Computer and Communications Security (CCS))", 2009.

[14] M. SERRANO. *Anatomy of a Ubiquitous Media Center*, in "Proceedings of the Sixteenth Annual Multimedia Computing and Networking (MMCN'09), San Jose, CA, USA", (taux d'acceptation 12/34), Jan 2009.

Scientific Books (or Scientific Book chapters)

[15] G. BERRY. *Penser, modéliser et maîtriser le calcul informatique*, 2009.

Other Publications

[16] L. ABBAS-TURKI, S. VIALLE. *European Option Princing on a GPU Cluster*, 2009, http://hal-supelec.archives-ouvertes.fr/hal-00364242/en/, Note bidon.

[17] G. BOUDOL, G. PETRI. *A theory of speculative computation*, October 2009, Submitted.

[18] G. BOUDOL, G. PETRI. *On speculative computation and thread safe programming*, November 2009, Draft.

[19] S. CAPECCHI, I. CASTELLANI, M. DEZANI-CIANCAGLINI, T. REZK. *Session types for access and information flow control*, Dec 2009, Draft.

[20] Z. LUO. *Secure information flow in ULM as a safety property*, April 2009, Draft.

[21] M. SERRANO, C. QUEINNEC. *A multi-tier semantics for Hop*, 2010, Submitted.

[22] M. SERRANO. *HSS: a compiler for Cascading Style Sheets*, 2010, Submitted.