Activity Report 2013

Project-Team INDES

Secure Diffuse Programming

RESEARCH CENTER
Sophia Antipolis - Méditerranée

THEME
Architecture, Languages and Compilation

Table of contents

1. Members **1**
2. Overall Objectives **1**
3. Research Program **2**
3.1. Parallelism, concurrency, and distribution 2
3.2. Web and functional programming 2
3.3. Security of diffuse programs 2
4. Application Domains **3**
4.1. Web programming 3
4.2. Multimedia 3
4.3. Home Automation 3
5. Software and Platforms **4**
5.1. Introduction 4
5.2. Functional programming 4
5.3. Language-based Security 4
5.4. Web programming 4
5.5. Old software 5
5.5.1. Camloo 5
5.5.2. Skribe 5
5.5.3. Scheme2JS 5
5.5.4. The FunLoft language 5
5.5.5. CFlow 6
5.5.6. FHE type-checker 6
5.5.7. Mashic compiler 6
6. New Results **6**
6.1. Security 6
6.1.1. Stateful Declassification Policies for Event-Driven Programs 6
6.1.2. A Monitor Inlining Compiler for Securing JavaScript Programs 7
6.1.3. Modular Extensions of Security Monitors for Web APIs: The DOM API Case Study 7
6.1.4. A Certified Lightweight Non-Interference Java Bytecode Verifier 7
6.1.5. Session types for liveness and security 7
6.1.6. Noninterference in reactive synchronous languages 8
6.2. Models, semantics, and languages 8
6.2.1. Formalization and Concretization of Ordered Networks 8
6.2.2. Absence Prediction in Esterel 8
6.2.3. Reactive Synchronous Languages 8
6.2.4. Locking Fast 9
6.2.5. JThread 9
6.3. Web programming 10
6.3.1. Colored λ-calculus 10
6.3.2. Multitier Debugging 10
6.3.3. Hop and HipHop : Multitier Web Orchestration 11
6.3.4. Hop Programming Environment 12
6.3.5. Web of Data 12
6.4. Web robotics 12
6.4.1. Cable driven robots 13
6.4.2. Web Robotics 13
7. Partnerships and Cooperations **15**
7.1. National initiatives 15
7.1.1. ANR DEFIS PWD 15

7.1.2. FUI X-Data 15
7.1.3. MEALS 15
7.2. European initiatives 15
7.2.1. FP7 Projects 15
7.2.2. Collaborations in European Programs, except FP7 15
8. Dissemination **16**
8.1. Seminars and conferences 16
8.2. Animation 16
8.3. Teaching - Supervision - Juries 17
8.3.1. Teaching 17
8.3.2. Supervision 17
8.3.3. Juries 17
8.4. Popularization 18
8.5. Transfer 19
9. Bibliography **19**

Project-Team INDES

Keywords: Programming Languages, Compiling, Security, Concurrency, Web

Creation of the Team: 2009 January 01*, updated into Project-Team:* 2010 July 01.

1. Members

Research Scientists

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

Faculty Member

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

Engineers

Vincent Prunet [Inria]
Marcela Rivera [Inria]
Ludovic Courtès [Inria]

PhD Students

Cyprien Nicolas [Inria]
Johan Grande [Inria]
Pejman Attar [Inria]
José Santos [Inria]
Yoann Couillec [Inria]

Administrative Assistant

Nathalie Bellesso [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 is partly inspired by and built around Hop, the web programming model proposed by the former Mimosa team, which takes the web as its execution platform and targets interactive and multimedia applications.

3. Research Program

3.1. Parallelism, concurrency, and distribution

Concurrency management is at the heart of diffuse programming. Since the execution platforms are highly heterogeneous, many different concurrency principles and models may be involved. 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 [6]. We have created a 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 as an extension of the BIGLOO compiler that we develop [7]. 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 the security of diffuse programs. Research on language-based security has been carried on before in former Inria teams [2], [1]. 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 [4]. 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 that 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. Home 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 and Platforms

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: Cyprien Nicolas, Bernard Serpette, Manuel Serrano [correspondant].

5.2.1. The Bigloo compiler

The programming environment for the Bigloo compiler [7] 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.

5.3. Language-based Security

Participants: Tamara Rezk [correspondant], José Santos.

5.3.1. IFJS compiler

The IFJS compiler is applied to JavaScript code. The compiler generates JavaScript code instrumented with checks to secure code. The compiler takes into account special features of JavaScript such as implicit type coercions and programs that actively try to bypass the inlined enforcement mechanisms. The compiler guarantees that third-party programs cannot (1) access the compiler internal state by randomizing the names of the resources through which it is accessed and (2) change the behaviour of native functions that are used by the enforcement mechanisms inlined in the compiled code.

The compiler is written in JavaScript and can be found at http://www-sop.inria.fr/indes/ifJS.

5.4. Web programming

Participants: Gérard Berry, Cyprien Nicolas, Manuel Serrano [correspondant].

5.4.1. The HOP web programming environment

HOP is a 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 a 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. It is released under the GPL license. It is available at http://hop.inria.fr.

5.5. Old software

5.5.1. Camloo

Camloo is a caml-light to bigloo compiler, which was developed a few years ago to target bigloo 1.6c. New major releases 0.4.x of camloo have been done to support bigloo 3.4 and bigloo 3.5. Camloo make it possible for the user to develop seamlessly a multi-language project, where some files are written in caml-light, in C, and in bigloo. Unlike the previous versions of camloo, 0.4.x versions do not need a modified bigloo compiler to obtain good performance. Currently, the only supported backend for camloo is bigloo/C. We are currently rewriting the runtime of camloo in bigloo to get more portability and to be able to use HOP and camloo together.

5.5.2. Skribe

SKRIBE is a functional programming language designed for authoring documents, such as Web pages or technical reports. It is built on top of the SCHEME programming language. Its concrete syntax is simple and looks familiar to anyone used to markup languages. Authoring a document with SKRIBE is as simple as with HTML or LaTeX. It is even possible to use it without noticing that it is a programming language because of the conciseness of its original syntax: the ratio *tag/text* is smaller than with the other markup systems we have tested.

Executing a SKRIBE program with a SKRIBE evaluator produces a target document. It can be HTML files for Web browsers, a LaTeX file for high-quality printed documents, or a set of *info* pages for on-line documentation.

5.5.3. 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 at http://www-sop.inria.fr/indes/scheme2js.

5.5.4. The FunLoft language

FunLoft (described in http://www-sop.inria.fr/teams/indes/rp/FunLoft) is a 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/teams/indes/rp. Several benchmarks are given, including cellular automata and simulation of colliding particles.

5.5.5. CFlow

The prototype compiler "CFlow" takes as input code annotated with information flow security labels for integrity and confidentiality and compiles to F# code that implements cryptography and protocols that satisfy the given security specification.

Cflow has been coded in F#, developed mainly on Linux using mono (as a substitute to .NET), and partially tested under Windows (relying on .NET and Cygwin). The code is distributed under the terms of the CeCILL-B license.

5.5.6. FHE type-checker

We have developed a type checker for programs that feature modern cryptographic primitives such as fully homomorphic encryption. The type checker is thought as an extension of the "CFlow" compiler developed last year on the same project. It is implemented in F#. The code is distributed under the terms of the CeCILL-B license.

5.5.7. Mashic compiler

The Mashic compiler is applied to mashups with untrusted scripts. The compiler generates mashups with sandboxed scripts, secured by the same origin policy of the browsers. The compiler is written in Bigloo and can be found at http://www-sop.inria.fr/indes/mashic/.

6. New Results

6.1. Security

Participants: Ilaria Castellani, Bernard Serpette [correspondant], José Santos.

6.1.1. Stateful Declassification Policies for Event-Driven Programs

We propose a novel mechanism for enforcing information flow policies with support for declassification on event-driven programs. Declassification policies consist of two functions. First, a projection function specifies for each confidential event what information in the event can be declassified directly. Second, a stateful release function specifies the aggregate information about all confidential events seen so far that can be declassified. We provide evidence that such declassification policies are useful in the context of JavaScript web applications. An enforcement mechanism for our policies is presented and its soundness and precision are proven. Finally, we give evidence of practicality by implementing and evaluating the mechanism in a browser.
Report and mechanization can be found in http://people.cs.kuleuven.be/~mathy.vanhoef/declass.

6.1.2. A Monitor Inlining Compiler for Securing JavaScript Programs

JavaScript applications can include untrusted code dynamically loaded from third party code providers (such as online advertisements). This issue raises the need for enforcement mechanisms to ensure security properties for JavaScript programs. The dynamic nature of the JavaScript programming language makes it a hard target for static analysis. Hence, research on mechanisms for enforcing security properties for JavaScript programs has mostly focused on dynamic approaches, such as runtime monitoring and program instrumentation. We design and implement a novel compiler that inlines a security monitor and we formally prove it correct with respect to an information flow security property. To the best of our knowledge, it is the first proven correct information flow monitor inlining transformation for JavaScript programs.
Report can be found in http://www-sop.inria.fr/indes/ifJS. See also software section.

6.1.3. Modular Extensions of Security Monitors for Web APIs: The DOM API Case Study

JavaScript programs often interact with the web page on which they are included, as well as with the browser itself, through external APIs such as the DOM API, the XMLHttpRequest API, and the W3C Geolocation API. The continuous emergence and heterogeneity of different external APIs renders the problem of precisely reasoning about JavaScript security particularly challenging. To tackle this problem, we propose a methodology for extending arbitrary sound JavaScript monitors. The methodology allows us to prove noninterference for external APIs in a modular way. Thus, when considering new external APIs, the noninterference property of the security monitor still holds. We present two groups of DOM interfaces that illustrate how to extend a noninterferent monitor model with: (1) basic DOM methods, for which we have discovered new information leaks not explored in previous work; (2) live collections, which are special features of the DOM API with an unconventional semantics that can lead to several previously unknown information leaks. Finally, we inline an extensible noninterferent JavaScript monitor that handles (1) and (2), and we make it available online
Report can be found in http://www-sop.inria.fr/indes/ifJS.

6.1.4. A Certified Lightweight Non-Interference Java Bytecode Verifier

We propose a type system to verify the non-interference property in the Java Virtual Machine. We verify the system in the Coq theorem prover.

Noninterference guarantees the absence of illicit information flow throughout program execution. It can be enforced by appropriate information flow type systems. Much of the previous work on type systems for non-interference has focused on calculi or high-level programming languages, and existing type systems for low-level languages typically omit objects, exceptions and method calls. We define an information flow type system for a sequential JVM-like language that includes all these programming features, and we prove, in the Coq proof assistant, that it guarantees non-interference. An additional benefit of the formalisation is that we have extracted from our proof a certified lightweight bytecode verifier for information flow. Our work provides, to the best of our knowledge, the first sound and certified information flow type system for such an expressive fragment of the JVM.

This work appeared in the journal of Mathematical Structures in Computer Science [9].

6.1.5. Session types for liveness and security

Within the COST Action BETTY, we have started studying the interplay between liveness properties and secure information flow properties in session calculi, in collaboration with a colleague from Torino University. Recent developments in static analysis techniques have shown that behavioural types, and in particular session types, may be used to enforce liveness properties of communicating systems. Examples of such properties are deadlock freedom, eventual message delivery and session termination. Because secure information flow in communicating systems depends on the observation of messages, there is a clear connection between information flow analysis and the liveness properties of the systems under consideration. We have been examining the joint application of liveness enforcement and secure information flow analysis in session calculi. It appears that, by strengthening the assumptions on the liveness of systems, it is possible to relax the conditions under which a system satisfies secure information flow properties. This is ongoing work, which is expected to continue within the BETTY Action.

6.1.6. Noninterference in reactive synchronous languages

We defined two properties of Reactive Noninterference (RNI) for a core synchronous reactive language called CRL formalising secure information flow. Both properties are time-insensitive and termination-insensitive. Again, coarse-grained RNI is more abstract than fine-grained RNI.

Finally, a type system guaranteeing both security properties was presented. Thanks to a design choice of CRL, which offers two separate constructs for loops and iteration, and to refined typing rules, this type system allows for a precise treatment of termination leaks, which are an issue in parallel languages.

This work has been presented at the International Symposium on Trustworthy Global Computing (TGC 2013) [11]. It is also described in Attar's PhD thesis pejman:tel-00920152.

6.2. Models, semantics, and languages

Participants: Pejman Attar, Gérard Berry, Gérard Boudol, Ilaria Castellani, Johan Grande, Cyprien Nicolas, Tamara Rezk, Manuel Serrano [correspondant].

6.2.1. Formalization and Concretization of Ordered Networks

Overlay networks have been extensively studied as a solution to the dynamic nature, scale and heterogeneity of large computing platforms, and are a fundamental layers of most existing peer-to-peer networks. The basic mechanism offered by an overlay network, is routing, i.e., the mechanism enabling the delivery of messages from any node to any other node in the network. On top of routing are built crucial functionalities of peer-to-peer networks, such as networks maintenance (nodes joining and leaving the network) and information distribution and retrieval. Over the years, different topologies and routing mechanisms have been proposed in literature. However, there is a lack of formal works unifying these different designs and establishing their correctness. This paper proposes a formal common basis, partially validated with the Coq theorem prover, with the nice property of only requiring the definition of a total order on the nodes. We investigate how such a basic design can be used to build deadlock/livelock-free algorithms for routing, node insertion, and node deletion in the fault-free environment. The genericity of our design is then explored through the construction of orders on nodes corre- sponding to different topologies commonly encountered in the peer-to-peer domain. To validate the methodology proposed, a simulator tool was developed. This tool is able, given the definition of an order and the definition of shortcuts, to simulate the corresponding overlay network and to explore its performance.

6.2.2. Absence Prediction in Esterel

We have formally proved, with the Coq system, the correctness of an absence prediction of Esterel's signals. For this we have formalised in Coq the static analysis and the interpreter written in Scheme (see the previous activity report). With this formal specification, we prove the correctness of the analysis: if a signal is considered absent by the evaluator at one instant, then this signal will be not emitted during this instant. This work is described in a currently submitted paper.

6.2.3. Reactive Synchronous Languages

CRL: We have studied the security property of noninterference in a synchronous Core Reactive Language (CRL). In the synchronous reactive paradigm, programs communicate by means of broadcast events, and their parallel execution is regulated by a notion of instant.

We have first shown that CRL programs are indeed reactive, namely that they always converge to a state of termination or suspension ("end of instant") in a finite number of steps. This property is important as it also entails the reactivity of a program to its environment, namely its capacity to input events from the environment at the start of instants, and to output events to the environment at the end of instants. While classical in synchronous languages, this property required to be established afresh in CRL, since this language makes use of a new asymmetric parallel operator.

We defined two bisimulation equivalences on CRL programs, corresponding respectively to a fine-grained and to a coarse-grained observation of programs. We showed that coarse-grained bisimilarity is more abstract than fine-grained bisimilarity, as it is insensitive to the order of generation of events and to repeated emissions of the same event during an instant.

DSLM :

We have finalised our work on the language DSLM (Dynamic Synchronous Language with Memory), which is an extension of CRL with memory and distribution. There are now several sites, and agents may migrate between sites. Two main properties are established for DSLM: reactivity of each agent and absence of data-races between agents. Since DSLM uses the same asymmetric parallel operator as CRL, reactivity is proven in a similar way. Moreover, the language offers a way to benefit from multi-core and multi-processor architectures, by means of the notion of synchronized scheduler, which abstractly models a computing resource. Each site may be expanded and contracted dynamically by varying its number of synchronized schedulers. Moreover agents can be moved transparently from one scheduler to another one within the same site. In this way one can formally model the load-balancing of agents over a site. This work is part of Pejman Attar's PhD thesis, defended in December 2013.

6.2.4. Locking Fast

We have studied the integration of low-level locking mechanisms in programming language execution environments. We have shown that for a given low-level locking mechanism the performance of the applications may vary significantly according to decisions taken for integrating it in the runtime system. We have studied two different aspects. First, we have shown how to accelerate C IO locking by selecting at runtime the adequate implementation and by using spin locks instead of full-fledged mutexes. Second, we have presented a new schema for improving the slow path of Java-like synchronized blocks. It consists in lifting the exception handler that is installed on the stack and which is in charge of releasing a monitor up to the closest exception handler already installed on the stack. All these optimizations have been implemented in Hop, our Web programming language. We have conducted experiments that shows significant speed up (up to 30%) for applications using locks extensively.

The synchronization lifting technique could be generalized to all the exception handlers, not only the handlers of synchronized blocks. As lifting only modifies the interception of exceptions, not the way they are thrown, it is compatible with languages such as Java or JavaScript that store a description of the stack at the moment when the exception is thrown inside the exception handlers. The technique should thus be broadly applicable. Exploring this idea is left for future work.

This work is described in the paper that will be published in the proceedings of the SAC'14 conference [12].

6.2.5. JThread

The jthread library is a library for Hop offering threads and mutexes and whose main locking function implements deadlock avoidance. Our library offers structured locking (i.e., critical sections instead of explicit `lock`/`unlock` functions). It supports nested locking. Our library is implemented using the preexisting pthread library and is offered as an alternative to the latter.

Compared to usual locking functions, our primitive relies on the programmer to provide some supplementary information such as the set of mutexes that might be acquired while owning a first one. However, for this supplementary information we chose default values that limit the need for the programmer to actually write it to a minimum.

The syntax of our locking construct is as follows:

```
(synchronize* l [:prelock p]
   expr1
   expr2
   ... )
```

where `l` is the list of mutexes to lock and `p` is a list that contains (some of [1]) the mutexes that might be locked during the execution of the body of the construct.

The implementation of this function relies on the ability to lock n mutexes at once. We found an algorithm for this that is both deadlock-free and starvation-free. Our algorithm relies on a dynamic total ordering of threads; this is inspired by Lamport's bakery algorithm.

We wrote a starvation-freedom property that applies to our real-life language with dynamic thread creation and programs that run forever on purpose. To express the property we need to define the following relation over threads:

$t_1 prect_2$ iff. $existsm.t_1$ owns m and t_2 is waiting to lock m.

Let $prec^*$ be the symmetric transitive closure of $prec$.

The property that we chose and proved for our algorithm is:

If each non-waiting thread eventually releases all the mutexes it owns **and if** for each waiting thread t the number of threads t' s.t. $t' prec^* t$ does not tend toward $+infty$ over time **then** each waiting thread eventually gets the mutexes it is waiting to lock.

We have implemented our library and integrated it to Hop. We haven't released it yet. An article is in preparation.

6.3. Web programming

Participants: Gérard Berry, Yoann Couillec, Ludovic Courtès, Cyprien Nicolas, Vincent Prunet, Tamara Rezk, Marcela Rivera, Bernard Serpette, Manuel Serrano [correspondant].

6.3.1. Colored λ-calculus

We have extended the bicolored λ-calculus to a polychromic one. With two colors, we were able to abstract the Hop language with its '`$`' and '`~`' annotations. With more than two colors, we can also modele embedded languages as a query based language, for example. As for the bicolored version, we have defined a static transformation aggregating expressions of the same color. We have formally proved, with the Coq system, the correctness, the confluence and the terminaison of the transformation. This work has been accepted for publication at the conference JFLA'14. [14].

6.3.2. Multitier Debugging

The distributed nature of Web applications makes debugging difficult. The programming languages and tools commonly used make it even more complex. Generally the server-side and the client-side are implemented in different settings and the debugging is treated as two separated tasks: on the one hand, the debugging of the server, on the other hand, the debugging of the client. Most studies and tools focus on this last aspect. They concentrate on the debugging of JavaScript in the browser. Although useful, this only addresses one half of the problem. Considering the debugging of Web applications as a whole raises the following difficulties:

- As the server-side and the client-side are generally implemented in different languages, debuggers for the Web do not capture the whole execution of the application. Programming the server and the client in the same language helps but is not sufficient to let the debugger expose a coherent view of the whole execution as this also demands a runtime environment that enforces consistent representations of data structures and execution traces.
- The JavaScript tolerant semantics tends to defer errors raising. For instance, calling a function with an insufficient number of arguments may lead to filling a data structure with the unexpected `undefined` value which, in turn, may raise a type error when accessed. The *distance* between the error and its actual cause may be arbitrarily long which can make the relation between the two difficult to establish.

[1] According to a few rules that we impose

- The JavaScript event loop used for the GUI splits the execution into unrelated callback procedures which get called upon event receipts. When an error occurs, the active stack trace only contains elements relative to the current callback invocation. It is oblivious of the context of the callback. Understanding the cause of the error is then not easy.

Pursuing our research on multitier programming for the Web, we have built a programming environment which eliminates most of these problems.

- When an error is raised, the full stack trace is reported. This stack trace might contain server stack frames, client stack frames, or both. We call this a *multitier stack trace*.
- When an error occurs, either on the client or on the server, its source location is reported by the debugger.
- In *debugging mode*, types, arities, and array bounds, are strictly enforced on the server and on the client. Hence, when the execution of the program deviates from the formal semantics of the language, an error is raised immediately.

A paper currently submitted presents this debugger and exposes the salient aspects of its implementation is under submission.

6.3.3. Hop and HipHop : Multitier Web Orchestration

Our aim is to help programming rich applications driven by computers, smartphones or tablets; since they interact with various external services and devices, such applications require orchestration techniques that merge classical computing, client-server concurrency, web-based interfaces, and event-based programming. To achieve this, we extend the Hop multitier web programming platform [5] by the new HipHop domain specific language (DSL), which is based on the synchronous language Esterel. HipHop orchestrates and synchronizes internal and external activities according to timers, events generated by the network, GUIs, sensors and devices, or internally computed conditions.

Like Esterel, Hiphop is a concurrent language based on the perfect synchrony hypothesis: a HipHop program repeatedly reacts in conceptual zero-delay to input events by generating output events; synchronization and communication between parallel statements is also performed in conceptual zero-delay. Perfect synchrony makes concurrent programs deterministic and deadlock-free, the only non-determinism left being that of the application environment. Its implementation is cycle-based, execution consisting of repeated atomic cycles "read inputs / compute reaction / generate outputs" in coroutine with the main Hop code. Concurrency is compiled away by static or dynamic sequential scheduling of code fragments. Cyclic execution atomicity avoids interference between computation and input-output, which is the usual source of unexpected non-determinism and synchronization problems for classical event-handler based programming.

While Esterel is limited to static applications, HipHop is designed for dynamicity. Its implementation on top of Hop makes it possible to dynamically build and run orchestration programs at any time using Hop's reflexivity facilities. It even makes it possible to modify a HipHop program between two execution cycles. It also simplifies the language by importing Hop's data definition facilities, expressions, modular structure, and higher-order programming features. It relies on the Web asynchronous concurrency and messaging already supported by Hop.

Using HipHop for real-life applications such as multimedia applications has been presented in an invited paper of the conference ICDCIT'14 [10].

This year, we extended the HipHop language with dynamic constructions, namely `genpar&` and `dyngenpar&`. These new constructs allow HipHop applications to parallelize treatment of event's values without knowing *a priori* the number of values carried by a given event. genpar& is alike a delayed parallel map execution, while dyngenpar& may create new parallel branches on-demand.

HipHop was also extended with listeners, alike HTML/DOM ones. The programmer can attach functions to any element of the HipHop programe that will be triggered when an instruction is started, suspended, resumed, terminated or aborted. These listeners enable us to trace a specific part of a program, easing its debugging.

6.3.4. Hop Programming Environment

In Linux-based environments, Hop is launched using the command line or using OS init scripts. This is inadequate for the Mac OS environment where graphical user interfaces are generally used to start, stop, and control applications. To fit the Mac OS users habits we have implemented a graphical front-end to Hop. It allows users to monitor and manage Hop processes. The implemented high level graphical interface is a XCode project which has been developed in objective-C for Mac OS X 10.7/10.8.

The main functionalities developed in this graphical front-end are:

- easily manage Hop processes. This GUI allows users to start, to stop and to restart the execution of Hop processes in a simplified manner. This process is executed in an independent thread in order to prevent impacts in the main program. Even though the process becomes independent, the main program can still control the execution of the Hop process by stopping or restarting it.
- Display messages in a user-frienly way. All messages as well as standard and error outputs, generated after the launch Hop server, are captured and redirected to be displayed in the main program. This allows the user to monitor the execution of the Hop process at any time. In this way, the user can search for a specific output or display pattern. All messages can be saved in external files for further analysis.
- Launch a Hop process with specific settings. It is possible to specify the port number on which the Hop server will accept connection. The verbosity and debugging level can also be specified according to the needs. At the same time the user can activate/deactivate specific options like Zeroconf and Webdav.
- Specify additional arguments to run a Hop process in a "command line like" way. In particular situations, advanced users could need to specify some options when launching the Hop server.

Additionally, a set of scripts have been developed to facilitate the generation and distribution of this work. A group of scripts allows one to compile and build the Hop GUI without needing a graphic Xcode interface. In this way, it is possible to generate an application bundle in a local machine as well as in a remote one without needing additional graphical interfaces.

The other group of scripts allows one to generate a "ready to use" dmg image containing Hop files. This dmg image can either include the graphical user interface (GUI) or not. Due to the continuous evolution of Hop based on new requirements and bugs fixed, the latter set of scripts provides a powerful tool to improve the releasing of new product versions.

This front-end has been integrated in the main Hop development tree. The MacOS pre-compiled version is publicly available on the Hop web site http://hop.inria.fr.

6.3.5. Web of Data

We are extending the Hop programming language in order to improve its data management: the amount of data it can access, the increasing number of sources of data and the heterogeneity of data it can accept. We have made an implementation of the SPARQL query language and the ORC orchestration language in Hop. We have written a configurable interpreter of the ORC language. The parallelism of the interpreter can be activated or not, for each operator of the ORC language. This specificity allows different executions of an ORC application, depending on the execution context or constraints, such as an execution on a client which disallows any parallelism. Within the X-Data project, we have participated in the development of a data intensive application in collaboration with Data Publica, the leading company of the project, and with the Inria Zenith research team. This application analyzed data sets provided by the French Insee institute to exhibit population commuting patterns. Our incentive for participating in this development was to acquire knowledge on programming data intensive applications. In the mid-term, we will rest on this expertise to create new data-aware programming languages or programming language extensions.

6.4. Web robotics

Participants: Ludovic Courtès, Cyprien Nicolas, Vincent Prunet [correspondant], Manuel Serrano.

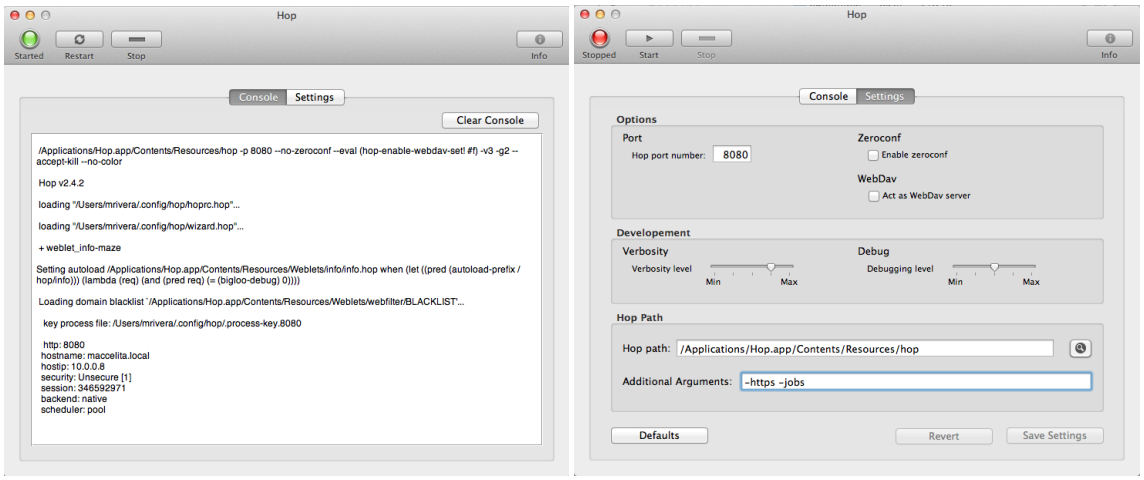

Figure 1. A screenshot of the MacOS X Hop graphical interface.

6.4.1. Cable driven robots

The sound design of modern robotic applications demands for the configuration-time integration of various subsystems which together constitute a robot. APIs and protocols such as ROS (Robot Operating System) provide robot designers with tools to combine software and hardware components into a complete robot. In addition, more and more robots need to share information or interact with diffuse objects available in the robot neighborhood, and also with remote services, to log data (typically activity monitoring data in the case of an assistance robot), to send information messages (alarms or triggering events to some other infrastructure), to subscribe to services provided by objects or remote servers, to provide services that may help peer entities, to get new behaviors by downloading and installing applications within the robot. We develop tools and architectures to address these requirements using Hop as our main platform. We experiment software architectures involving robots, web objects, several integration models with third party components (hardware, software computation libraries for robotics, stand-alone robots), protocols, and libraries.

We pursued the joint work with Coprin Team about using Hop to coordinate a cable-driven robot. We changed the hardware on which Hop runs to a mini-PC instead of a standard laptop, plugged a wireless router, and used the wireless network from a tablet to move the robot. We also improved the robot hardware and software. The setup has been summarized in a paper [13] and presented at a national conference on robotics.

6.4.2. Web Robotics

Web Robotics is a two years Inria ADT project targeting the development of technical foundations (libraries and toolkits) and demos of web enabled robots. The project is led by Indes (Vincent Prunet, Manuel Serrano), software development is supported by Inria SED (Ludovic Courtès), robots are provided by Inria Coprin. A demonstrator and dissemination platform consisting of a cable robot and dedicated web services have been set up to enable people to interact with the robot through a web server (web http://webrobotics.inria.fr:8080/hop/welcome).

The web robotics demo demonstrates:

- the programming of robot control functions within Hop
- simulation/real hardware abstraction
- hardware control (Phidget integration)

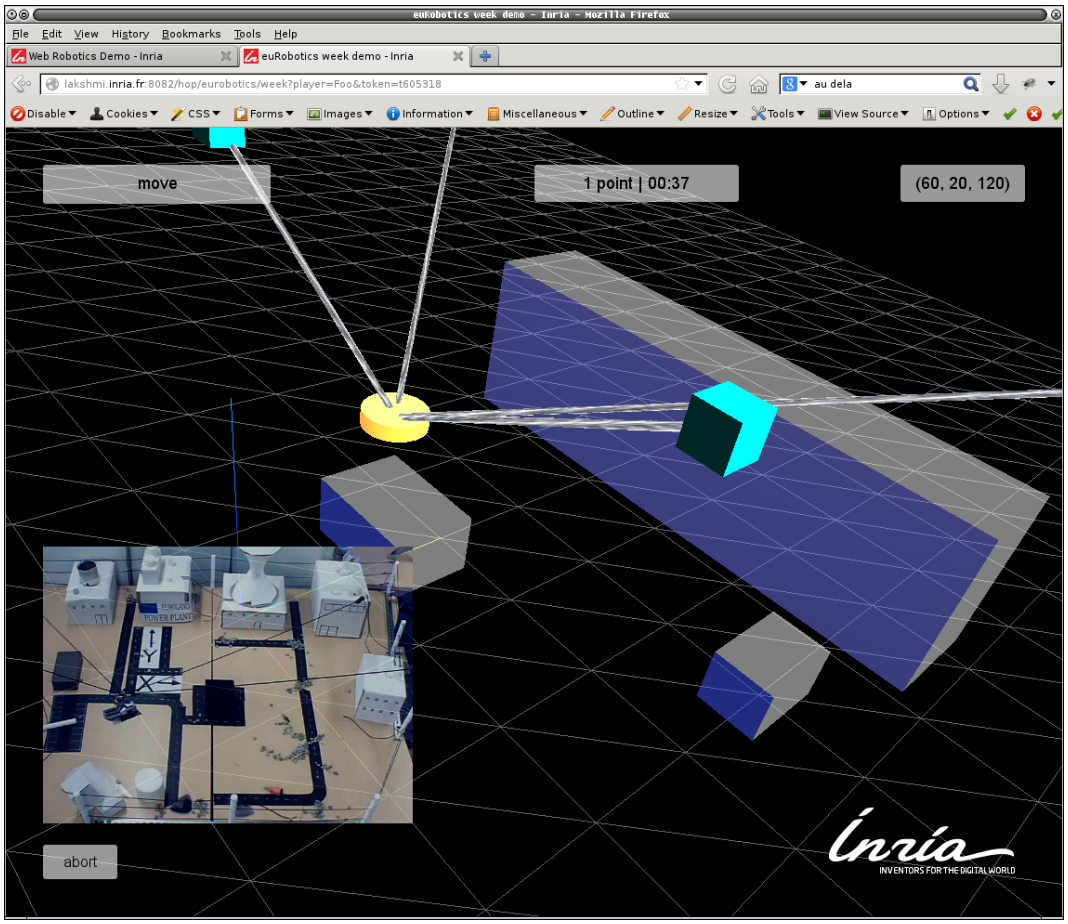

Figure 2. A web-controlled cable robot. The whole application is implemented with Hop.

- integration of specialized robotics libraries
- multi server architecture
- management of multiple users and authenticated access to critical resources.

Hop added value is to provide:

- an unconstrained specification environment where data and services are easily shared among servers and clients;
- a seamless, plugin free, integration into standard web browsers.

Also in 2013, Indes has joined the PAL (Person Assisted Living) Inria project, to develop web enabled applications within the project.

7. Partnerships and Cooperations

7.1. National initiatives

7.1.1. 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.

7.1.2. FUI X-Data

Broadly available big and open data open new perspectives in terms of use and applications. The X-Data project aims at validating this claim by using actual data sets for building realistic applications. The goal is to combine a large variety of data sets coming from different partners (Data Publica, Orange, EDF, La Poste, social networks, ...) to build innovative applications. The Indes team designs and implements new programming language constructs that help programming these applications.

7.1.3. MEALS

The MEALS project (Mobility between Europe and Argentina applying Logics to Systems), IRSES program, started October 1st (2011), and will end September 30th, 2015. The project goals cover three aspects of formal methods: specification (of both requirement properties and system behavior), verification, and synthesis. The Indes members are involved in the task of Security and Information Flow Properties (WP3). The partners in this task include University of Buenos Aires, University of Cordoba, Inria (together with Catuscia Palamidessi, Kostas Chatzikokolakis, Miguel Andrés) and University of Twente.

7.2. European initiatives

7.2.1. FP7 Projects

Program: RAPP

Title: Robot App Store

Collaborator: Inria Coprin

Abstract: RAPP is a 36 months pan-european FP7 project, started in December 2013. Hop is used in the development of prototypes of the Coprin Ang rollator transfer device, for mobility assistance and activity monitoring.

7.2.2. Collaborations in European Programs, except FP7

Program: ICT Cost Action IC1201

Program acronym: BETTY

Project title: Behavioural Types for Reliable Large-Scale Software Systems

Duration: October 2012 - October 2016

Coordinator: Simon Gay, University of Glasgow

Other partners: Several research groups, belonging to 22 european countries

Abstract: The aim of BETTY is to investigate and promote behavioural type theory as the basis for new foundations, programming languages, and software development methods for communication-intensive distributed systems. Behavioural type theory encompasses concepts such as interfaces, communication protocols, contracts, and choreography.

8. Dissemination

8.1. Seminars and conferences

- **Pejman Attar** gave a talk about security in concurrency at Paris 7 in february 2013. He participated March in *rencontres du numérique de l'ANR* and presented a poster about his work and the PARTOUT ANR team work.
- **Nataliia Bielova** was invited to present her work on web security in Labex Security day.
- **Ilaria Castellani** participated in March in the 1st working group (WG) meeting of the BETTY Action, in Rome, where she animated the discussion of the Security WG. In September, she took part in the 2nd WG meeting of BETTY, in Madrid, where she again animated the discussion of the Security WG. In August 2013, Ilaria Castellani participated in the workshop "25 Years of Combining Compositionality and Concurrency" (WS25CCC), in Königswinter, Germany, and in the 8th International Symposium on Trustworthy Global Computing (TGC 2013), in Buenos Aires, Argentina. In both of them, she presented her joint work with Pejman Attar [11]. In Buenos Aires, she also took part in the annual meeting of the IFIP WG 1.8 on Concurrency Theory.
- **Cyprien Nicolas** gave a talk about "Cable-Driven Robots with Wireless Control Capability for Pedagogical Illustration of Science" at the 8th National Conference of "Control Architecture of Robots" in Angers, France, June, 12th 2013 [13].
- **Tamara Rezk** was invited to present her work on web security in IRISA Rennes in January and in University of Cordoba in March. She gave a talk in the Labex Security day.
- **Manuel Serrano** gave a seminar about Web programming in the seminar series associated with the course of Gérard Berry at Collège de France called *Informatique du Temps et des Évenements*. He participated in a one-week seminar in Brussels on Secure Cloud and Reactive Internet Programming Technology, where he gave a talk on HipHop. He presented the results of the PWD projet at the ANR *Rencontres du numérique* in Paris. Manuel Serrano gave a lecture on Hop at the ECOOP Programming Summer School.

8.2. Animation

- **Ilaria Castellani** is a member of the editorial board of *Technique et Science Informatiques*. She is a member of the IFIP WG 1.8 on Concurrency Theory. She is a member of the Management Committee of the BETTY Action, and the chair of the BETTY working group on Security. She was a member of the programme committee of the workshop EXPRESS/SOS 2013.
- **Tamara Rezk** was a member of the programme committees of JAIIO'13, CIBSI'13, TGC'13, LATIN'14, TIBETS'13, SOFSEM'14. She was invited to be in the PC of CSF'14. She was invited by the Estonian Research Council as an external expert to the evaluation of a research project. She is part of the organizing committee for the Labex Security day with **Nataliia Bielova**.

- **Manuel Serrano** is a member of the editorial board of the *Journal of Functional Languages*. He is the coordinator of the ANR DEFIS project PWD. He served on the program committee of the *16th Practical Aspects of Declarative Languages* (PADL'14) conference. He was a referee for the *Lisp in Summer Projects*. He was the co-program chair of the *European Lisp Symposium* (ELS'13).

8.3. Teaching - Supervision - Juries

8.3.1. Teaching

DUT: **Cyprien Nicolas**, *Introduction aux systèmes informatiques*, 36ETD, S1, IUT Nice Côte d'Azur, UNS, France. *Architecture Systèmes et Réseaux*, 16ETD, S3, IUT Nice Côte d'Azur, UNS, France.

Licence: **Cyprien Nicolas**, *Algorithmique et Complexité*, 30ETD, Licence Professionnelle SIL, IUT Nice Côte d'Azur, UNS. **Yoann Couillec**, *Algorithmique - Programmation objet - Python*, 36 ETD, L2, University of Nice Sophia Antipolis. **Vincent Prunet**, *Algorithms and Data Structures*, 80 ETD, L2, Lycée International de Valbonne, (Inria action to promote early CS courses in all scientific curricula).

Master: **Ilaria Castellani**, *Programmation et sécurité des applications du web*, 13.5 ETD, M2, University of Nice Sophia Antipolis. **Tamara Rezk**, *Programmation et sécurité des applications du web*, M2, University of Nice. *Programming the Diffuse Web*, 13.5 ETD, M2, University Paris 6 (UPMC), France. *Provable cryptography* 28h ETD, M2 University of Nice Sophia Antipolis. **Manuel Serrano**, *Programming the Diffuse Web*, 13.5 ETD, M2, University Paris 6 (UPMC), France.

PhD: **Manuel Serrano** gave a full-day seminar on Hop at the *École des Jeunes Chercheurs en Programmation* (Rennes).

8.3.2. Supervision

PhD: **Pejman Attar**, *Towards a safe and secure synchronous language*, University of Nice, 1/10/2010, **Frédéric Boussinot** and **Ilaria Castellani**.

PhD in progress: **Cyprien Nicolas**, *Orchestrating multi-tier programming languages*, University of Nice, 1/09/2010, **Gérard Berry** and **Manuel Serrano**.

PhD in progress: **Johan Grande**, *Conception et implantation d'un langage de programmation concurrente modulaire*, University of Nice, 1/10/2010, **Gérard Boudol** and **Manuel Serrano**.

PhD in progress: **Yoann Couillec**, *Langages de programmation et données ouvertes*, University of Nice, 1/10/2012, **Manuel Serrano** and **Patrick Valduriez**.

PhD in progress: **José Santos**, *Language based approach for information flow analysis in distributed mobile code*, University of Nice, 1/12/2010, **Tamara Rezk**.

Master internship: **Jérôme Brunel** master thesis at University of Nice-Sophia Antipolis, tutored by **Tamara Rezk**. **Gerard Boudol** has supervised the intership (Master Recherche) of Arthur Guillon, on relaxed memory models. The long term objective was to investigate the quantitative aspects of such models, but a first phase of the study consisted in refining an abstract model previously introduced by Boudol, Petri and Serpette. More precisely, this model was refined so as to provide an adequate semantics for PowerPC memory barriers, a notoriously difficult topic. To this end we extended the notion of visibility, attached to memory write operations, to these barriers. In this way, we achieved an accurate semantics of these synchronization operations with respect to the series of tests on PowerPC machines developed by Luc Maranget. In a second phase, some requirements for a notion of probabilistic memory model were identified.

8.3.3. Juries

Ilaria Castellani was a member of the jury of the PhD thesis of Fabrizio Montesi, IT University of Copenhagen.

8.4. Popularization

The Web is becoming the richest platform on which to create computer applications. Its power comes from three elements: modern Web browsers enable highly sophisticated graphical user interfaces (GUIs) with 3D, multimedia, fancy typesetting, among others; calling existing services through Web APIs makes it possible to develop sophisticated applications from independently available components; and open-data availability allows access to a wide set of information that was unreachable or that simply did not exist before. The combination of these three elements has already given birth to revolutionary applications such as GoogleMaps, radio podcasts, and social networks.

The next step is likely to be incorporating the physical environment into the Web. Recent electronic devices are equipped with various sensors (GPS, cameras, microphones, metal detectors, speech commands, thermometers, motion detection, and so on) and communication means (IP stack, telephony, SMS, Bluetooth), which enable applications to interact with the real world. Web browsers integrate these features one after the other, making the Web runtime environment richer every day. The future is appealing, but one difficulty remains: current programming methods and languages are not ideally suited for implementing rich Web applications. This is not surprising as most have been invented in the 20th century, before the Web became what it is now.

Traditional programming languages have trouble dealing with the asymmetric client-server architecture of Web applications. Ensuring the semantic coherence of distributed client-server execution is challenging, and traditional languages have no transparent support for physical distribution. Thus, programmers need to master a complex gymnastics for handling distributed applications, most often using different languages for clients and servers. JavaScript is the dominant Web language but was conceived as a browser only client language. Servers are usually programmed with quite different languages such as Java, PHP, Ruby, etc. Recent experiments such as Node.js propose using JavaScript on the server, which makes the development more coherent; however, harmonious composition of independent components is still not ensured.

In 2006, three different projects, namely, GWT from Google, Links from the University of Edinburgh, and HOP from Inria (`http://www.inria.fr`) [6], offered alternative methods for programming Web applications. They all proposed that a Web application should be programmed as a single code for the server and client, written in a single unified language. This principle is known as multitier programming.

Links is an experimental languages in which the server holds no state and functions can be symmetrically called from both sides, allowing them to be declared on either the server or the client. These features are definitely interesting for exploring new programming ideas, but they are difficult to implement efficiently, making the platform difficult to use for realistic applications.

GWT is more pragmatic. It maps traditional Java programming into the Web. A GWT program looks like a traditional Java/Swing program compiled to Java bytecode for the server side and to JavaScript for the client side. Java cannot be considered as the unique language of GWT, however. Calling external APIs relies on JavasCript inclusion in Java extensions. GUIs are based on static components declared in external HTML files and on dynamic parts generated by the client-side execution. Thus, at least Java, Javascript, and HTML are directly involved.

The HOP language takes another path relying on a different idea: incorporating all the required Web-related features into a single language with a single homogeneous development and execution platform, thus uniformly covering all the aspects of a Web application: client-side, server-side, communication, and access to third-party resources. HOP embodies and generalizes both HTML and JavaScript functionalities in a Scheme-based platform that also provides the user with a fully general algorithmic language. Web services and APIs can be used as easily as standard library functions, whether on the server side or client side.

In order to popularize HOP, we have written a paper for targeting engineers which presents on overview of the HOP language and its development environment. It has been simultaneously published in ACM Queue and Communications of the ACM [5]. We have also given several demonstrations of the system. In particular, Cyprien Nicolas has co-developed application software for an educational cable robot (Coprin) presented at the Fête de la Science, in November. The demo consisted in a Cable bot built by a Coprin student and piloted

by Hop, the software being written by a Indes student. The demo took place in front of four classes of High School students.

8.5. Transfer

8.5.1. Diffuse Robotics

Dissemination of the HOP technology has become a priority for the team now that HOP is actually used to develop large projects. In 2012, a further step was taken with the allocation of dedicated resources missioned to develop and transfer the application portfolio to the industry. The team has focused on bringing web awareness to personal assistance robots developed by the Coprin team, also at Inria CRISAM, in line with one of the top strategic orientations of Inria. Using web protocols as a native framework greatly simplifies the integration of the robot as a web entity, and the use of remote web services to manage, monitor or extend the features of the robot. The behavior of a HOP robot is specified in HOP and orchestrated within diffuse HOP run time agents embedded within the robot elements, in charge of handling communication and control between platforms and with remote web services. The project, code-named *Diffuse Robotics*, builds on the experience gained in using HOP for home automation over the recent years, adding in 2012 the support of versatile robotic computing platforms and associated mechanics and sensor hardware and a state of the art plug and play framework for automatic device and service discovery. Among the direct benefits of relying on a web framework are the ability to use any web enabled device such as a smartphone or tablet to drive the robot. Also, it is much simpler to put in place remote diagnostic and monitoring services by leveraging on existing robot sensors and the HOP framework.

9. 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, pp. 2-18

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

[3] G. BOUDOL, Z. LUO, T. REZK, M. SERRANO. *Reasoning about Web Applications: An Operational Semantics for HOP*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", 2012, vol. 34, n^{o} 2

[4] 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, pp. 323-335

[5] M. SERRANO, G. BERRY. *Multitier Programming in Hop - A first step toward programming 21st-century applications*, in "Communications of the ACM", August 2012, vol. 55, n^{o} 8, pp. 53–59 [*DOI :* 10.1145/2240236.2240253], http://cacm.acm.org/magazines/2012/8/153796-multitier-programming-in-hop/abstract

[6] 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

[7] M. SERRANO. *Bee: an Integrated Development Environment for the Scheme Programming Language*, in "Journal of Functional Programming", May 2000, vol. 10, n^{o} 2, pp. 1–43

Publications of the year

Doctoral Dissertations and Habilitation Theses

[8] P. ATTAR. , *Towards a safe and secure synchronous language*, Université Nice Sophia Antipolis, December 2013, http://hal.inria.fr/tel-00920152

Articles in International Peer-Reviewed Journals

[9] G. BARTHE, D. PICHARDIE, T. REZK. *A certified lightweight non-interference Java bytecode verifier*, in "Mathematical Structures in Computer Science", June 2013, vol. 23, n^{o} 5, pp. 1032-1081 [*DOI :* 10.1017/S0960129512000850], http://hal.inria.fr/hal-00915189

Invited Conferences

[10] G. BERRY, M. SERRANO. *Hop and HipHop : Multitier Web Orchestration*, in "International Conference on Distributed Computing and Internet Technology", Bhubaneswar, India, February 2014, http://hal.inria.fr/hal-00911782

International Conferences with Proceedings

[11] P. ATTAR, I. CASTELLANI. *Fine-grained and coarse-grained reactive noninterference*, in "Trustworthy Global Computing 2013", Buenos Aires, Argentina, December 2013, To appear in LNCS, Springer, http://hal.inria.fr/hal-00915241

[12] M. SERRANO, G. JOHAN. *Locking Fast*, in "Symposium on Applied Computing", Gyeongju, Korea, Republic Of, ACM, March 2014, http://hal.inria.fr/hal-00912569

National Conferences with Proceedings

[13] J. A. DIT SANDRETTO, C. NICOLAS. *Cable-Driven Robots with Wireless Control Capability for Pedagogical Illustration in Science*, in "CAR - 8th National Conference on "Control Architecure of Robots"", Angers, France, May 2013, http://hal.inria.fr/hal-00862752

Other Publications

[14] B. SERPETTE, P. MANOURY, E. CHAILLOUX. , *Unification des couleurs dans un lambda-calcul polychrome*, December 2013, http://hal.inria.fr/hal-00918944