Activity Report 2012

Project-Team INDES

Secure Diffuse Programming

RESEARCH CENTER
Sophia Antipolis - Méditerranée

THEME
Distributed Systems and Services

Table of contents

Project-Team INDES

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

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

1. Members

Research Scientists
Gérard Berry [Research Director, Inria, HdR]
Gérard Boudol [Research Director, Inria, HdR]
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]

PhD Students
Zhengqin Luo [MENRT]
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. Scientific Foundations

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 [4]. 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 [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 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 [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 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. 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 [Inria], Cyprien Nicolas [Inria], Bernard Serpette [Inria], Manuel Serrano [correspondant].

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/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.

5.3. Web programming

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

5.3.1. 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 http://mmc36.informatik.uni-augsburg.de/acmmm2007/. It is released under the GPL license. It is available at http://hop.inria.fr.

5.4. Language-based security

Participants: Zhengqin Luo [Inria], Tamara Rezk [correspondant].

5.4.1. 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 http://www.msr-inria.inria.fr/projects/sec/cflow/index.html.

5.4.2. 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 http://www.msr-inria.inria.fr/projects/sec/cflow/index.html.

5.4.3. 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/.

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`.

6. New Results

6.1. Security

Participants: Ilaria Castellani, Zhengqin Luo, Tamara Rezk [correspondant], José Santos, Manuel Serrano.

6.1.1. *Session types with security*

We have pursued our work on integrating security constraints within session types, in collaboration with our colleagues from Torino University. This resulted in the journal paper [8]. This article extends a previous conference paper with full proofs, additional examples and further results. In particular, [8] presents new properties of information-flow security, which is stronger and more compositional (*i.e.*, more robust with respect to parallel composition of processes) than that originally proposed, while being still ensured by the same session type system.

All the work on session types was partially funded by the ANR-08- EMER-010 grant PARTOUT. It is expected to continue within the starting COST Action BETTY.

6.1.2. *Mashic Compiler: Mashup Sandboxing Based on Inter-frame Communication*

Mashups are a prevailing kind of web applications integrating external gadget APIs often written in the Javascript programming language. Writing secure mashups is a challenging task due to the heterogeneity of existing gadget APIs, the privileges granted to gadgets during mashup executions, and Javascript's highly dynamic environment.

We propose a new compiler, called Mashic, for the automatic generation of secure Javascript-based mashups from existing mashup code. The Mashic compiler can effortlessly be applied to existing mashups based on a wide-range of gadget APIs. It offers security and correctness guarantees. Security is achieved by using the Same Origin Policy. Correctness is ensured in the presence of benign gadgets, that satisfy confidentiality and integrity constraints with regard to the integrator code. The compiler has been successfully applied to real world mashups based on Google maps, Bing maps, YouTube, and Zwibbler APIs.

This work appeared in CSF'12 [14]. See also software section.

6.1.3. 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. This work will appear in the journal of Mathematical Structures in Computer Science [6].

6.2. Models, semantics, and languages

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

6.2.1. Memory Models

As regards the theory of multithreading, we have extended our operational approach to capture more relaxed memory models than simple write buffering. A step was made in this direction by formalizing the notion of a speculative computation, but this was not fully satisfactory as an operational approach to the theory of memory models: indeed, in the speculative framework one has to reject a posteriori some sequences of executions as invalid. In [13] we have defined a truly operational semantics, by means of an abstract machine, for extremely relaxed memory models like the one of PowerPC. In our new framework, the relaxed abstract machine features a "temporary store" where the memory operations issued by the threads are recorded, in program order. A memory model then specifies the conditions under which a pending operation from this sequence is allowed to be globally performed, possibly out of order. The memory model also involves a "write grain," accounting for architectures where a thread may read a write that is not yet globally visible. Our model is also flexible enough to account for a form of speculation used in PowerPC machines, namely branch prediction. To experiment with our framework, we found it useful to design and implement a simulator that allows us to exhaustively explore all the possible relaxed behaviors of (simple) programs. The main problem was to tame the combinatory explosion due to the massive non-deterministic interleaving of the relaxed semantics. Introducing several optimizations described in [13], we were able to run a large number of litmus tests successfully.

6.2.2. Dynamic Synchronous Language with Memory

We have investigated the language DSLM (Dynamic Synchronous Language with Memory), based on the synchronous reactive model. In DSLM, systems are composed of several sites, each of which runs a number of agents. An agent consists of a memory and a script. This script is made of several parallel components which share the agent's memory. A simple form of migration is provided: agents can migrate from one site to another. Since sites have different clocks, a migrating agent resumes execution at the start of the next instant in the destination site. Communication between a migrating agent and the agents of the destination site occurs via (dynamically bound) events. The language uses three kinds of parallelism: 1) synchronous, cooperative and deterministic parallelism among scripts within an agent, 2) synchronous, nondeterministic and confluent parallelism among agents within a site, and 3) asynchronous and nondeterministic parallelism among sites. Communication occurs via both shared memory and events in the first case, and exclusively via events in the other two cases. Scripts may call functions or modules which are handled in a host language. Two properties are assured by DSLM: reactivity of each agent and absence of data-races between agents. 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. In this way one can model the load-balancing of agents over a site.

A secure extension of the language DSLM, called DSSLM (Dynamic Secure Synchronous Language with Memory), is currently under investigation. This language uses the same deterministic parallel operator for scripts as DSLM. It adds to DSLM a let operator that assigns a security level to the defined variable. Security levels are also assigned to events and sites, to allow information flow control during interactions and migrations. The study of different security properties (both sensitive and insensitive to the passage of the instants) and of type systems ensuring these properties is currently under way.

6.2.3. jthread

The jthread library (working name) is a Bigloo library featuring threads and mutexes and most notably a deadlock-free locking primitive. The jthread library appears as an alternative to Bigloo's pthread (POSIX threads) library and relies on it for its implementation.

The locking primitive is the following: `(synchronize* ml [:prelock mlp] expr1 expr2 ...)` where ml and mlp are lists of mutexes.

This primitive evaluates the expressions that constitute its body after having locked the mutexes in ml and before unlocking them back. The meaning of the *prelock* argument is to be explained below.

The absence of deadlocks is guaranteed by two complementary mechanisms:

- Each mutex belongs to a *region* defined by the programmer. Regions form a lattice which is inferred at runtime. A thread owning a mutex belonging to region R0 can only lock a mutex belonging to region R1 if R1 is lower than R0 in the lattice. This rule is enforced at runtime and guarantees the absence of deadlocks involving mutexes belonging to different regions.
- Under the previous condition, a thread owning a mutex M1 can lock a mutex M2 belonging to the same region only provided that M2 appeared in the *prelock* list of the `synchronize*` that locked M1. This rule is enforced at runtime and allows a *deadlock-avoiding* scheduling of threads based on previous work by Gérard Boudol and on Lamport's Bakery algorithm.

The library has been implemented. It is currently being integrated to Bigloo and benchmarked. It has not been released yet.

6.3. Web programming

Participants: Zhengqin Luo, Cyprien Nicolas, Tamara Rezk, Bernard Serpette, Manuel Serrano [correspondant].

6.3.1. Reasoning about Web Applications: An Operational Semantics for HOP

We propose a small-step operational semantics to support reasoning about web applications written in the multi-tier language HOP. The semantics covers both server side and client side computations, as well as their interactions, and includes creation of web services, distributed client-server communications, concurrent evaluation of service requests at server side, elaboration of HTML documents, DOM operations, evaluation of script nodes in HTML documents and actions from HTML pages at client side. We also model the browser same origin policy (SOP) in the semantics. We propose a safety property by which programs do not get stuck due to a violation of the SOP and a type system to enforce it. This work appeared in TOPLAS [7].

6.3.1.1. Hiphop

We pursued the development of the Hiphop orchestration language. The first version was written as a DSL with very few connection to Hop. During this year, we changed Hiphop syntax to blend it better with Hop. All Hiphop objects are now Hop values, and thus Hiphop programs can benefit from all Hop featues. The Hiphop development has enabled us to improve Hop stability and quality in client code generation.

We have found a new use-case for Hiphop: Robotics. We are currently working with the Inria Coprin team to pilot their robot using Hop and Hiphop. We have already used Hop to program with low-level motors API (using the Phidget libraries). Hop enabled us to distribute the robot control application over HTTP, in order to control the robot from a smartphone or tablet.

6.3.2. A CPS definition of HipHop

Since the Esterel model is used very dynamically in the HipHop framework, we have begun studying new frameworks of computations. We designed a definition of a HipHop-core, which is similar to Esterel-core, based on continuations. This approach allows a specification close to the implementation. The main problem was to define a predicate assuring the *absence* of a specific signal in the current instant. For this, we have designed a static analysis that predicts, for each program point and for each signal, the number of emissions remaining to be done until the end of the instant. The prediction may be over-estimated but when a null value is reached the corresponding signal can be considered as absent for the analyzed instant.

Contrary to existing analyses, this prediction can be done at compile time. Nevertheless some extra computations must be inserted in the evaluator to adjust a runtime prediction. For example. this is done when one branch of a conditional is dynamically taken, but adjusting the prediction only involves subtractions on global counters.

The continuation based definition doesn't prevent a space efficiency implementation. Esterel is known to be compiled to hardware and thus able to run a program in a fixed space of silicon; in the same manner we have implemented an evaluator than doesn't allocate extra memories while running a program: all the continuations can be allocated at compile time.

We have also extended the language to reach the HipHop definition. Some dynamic extensions (`mappar`) may dynamically allocate some resources but we were able to tune the static analysis to insure both confluence and constructive absence detection.

7. Bilateral Contracts and Grants with Industry

7.1. Contracts and Grants with Industry

7.1.1. Collaboration with Xiring

In 2011, Tamara Rezk collaborated with a french company based in Paris, Xiring. She visited the company several times in 2011 to carry out this collaboration.

7.1.2. Microsoft Research and Inria Joint Lab

Since 2007, Tamara Rezk is part of the Secure Distributed Computations and their Proofs project of the MSR-Inria lab in Saclay. She travelled several times in 2011 to visit the lab and continue with several collaborations concerning the project.

8. Partnerships and Cooperations

8.1. National initiatives

8.1.1. ANR DEFIS ParTout

The PARTOUT project (PARTOUT = PARallélisme parTOUT) is funded by the ANR Défis programme for 4 years, starting January 2009. The partners of this project are the teams INDES (coordinator), CNAM/CÉDRIC, and LRI, Université d'Orsay.

8.1.2. 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.

8.1.3. 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.

8.1.4. CIRIC

Indes participated in the proposal of the CIRIC project, a joint lab between Inria and Chile, that will start in 2012. Indes members are involved in the line: Internet Research and Development.

8.2. European initiatives

8.2.1. 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 17 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.

9. Dissemination

9.1. Seminars and conferences

- **Pejman Attar** gave a talk about "Safe and Secure Concurrency" at Paris 6 University. In November 2012, he presented the work "A CPS definition of Esterel" by B. Serpette at the workshop Synchron'12 at Le Croisic.
- **Gérard Berry** gave an invited talk in April 2012, at Edinburgh, "Reconciling Semantics and Implementation", for the Milner Symposium. He gave the talk "Informatique et art, quel avenir?" at the Villa Medicis, Rome, in February. He gave an invited conference at IRCAM "Parler du temps, mais de manière formelle", in Paris in June, and "Turing et ses contemporains: machines, langages, logiques" a talk at ENS Lyon in July.
- **Ilaria Castellani** participated in the workshop CoTiCo (Contracts and Behavioural Types), in Lucca, Italy, where she presented her current work on session types for security. In October 2012, she participated in the kick-off management committee meeting of the BETTY european Action, in Brussels.
- **Cyprien Nicolas** gave a talk about Hop at a Free and Open Source Software event named *Journées Méditerranéennes du Logiciel Libre* (JM2L). JM2L are a small local event, proposing around 30 conferences over two days, and targetting general public. It welcomed around 250 participants.
- **José Santos** gave a talk about his work on information flow in the CREST Open Workshop on Interference and Dependence organized by the Department of Computer Science of the University College of London in May 2012. In June 2012, he presented the article [12] in the Programming Languages and Analysis for Security (PLAS) Workshop in Beijing, China.
- **Tamara Rezk**, presented the work on the Mashic compiler in the Computer Security Foundations conference in Harvard, Boston, June 2012. In June 2012, she chaired the PLAS workshop, Beijing. In October 2012, she was invited to the Dagstuhl Seminar *Web Application Security* where she also participated to the *JavaScript sessions*.
- **Manuel Serrano** participated in a one-week seminar in Dagstuhl on Dynamic Languages, where he gave a talk on HipHop. He also participated in a one-week IFIP seminar which took place in Austin, Texas. He gave a talk on diffuse computing.

9.2. Animation

- **Gérard Berry** is a member of the *Académie des sciences*, of the *Académie des technologies*, and the *Academia Europaea*. He is member of the steering committee of the *ANR*.
- **Ilaria Castellani** is member of the editorial board of *Technique et Science Informatiques*. She was a member of the programme committee of EXPRESS'12.
- **Tamara Rezk** was a member of the programme committees of Bytecode, PLAS (co-chair with Sergio Maffeis), ARES, and LatinCrypt. She is is a member of the programme committee of JAIIO'13. She was a reviewer for CSF'12, Crypto'12, CCS'12, and the QIF journal.
- **Manuel Serrano** is the coordinator of the ANR DEFIS project PWD. He served on the program committees of the following conferences:
  - TFP'12, Symposium on Trends in Functional Programming.
  - COORDINATION'12, 7th International Federated Conferences on Distributed Computing Techniques.

9.3. Teaching - Supervision - Juries

9.3.1. Teaching

Licence: **Johan Grande**, *Structures de données et programmation*, 6 ETD, L1, University of Nice. **Vincent Prunet**, *Algorithms and Data Structures*, 42 ETD, L2, Lycée International de Valbonne (24 hours in 2012), (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. **Manuel Serrano**, *Programming the Diffuse Web*, 13.5 ETD, M2, University Paris 6 (UPMC), France.

PhD: **Pejman Attar** gave a 3 hour open course for PhD students on "Multi- core Languages and their problems" at the University of Eindhoven, Netherlands.

9.3.2. Supervision

PhD in progress: **Pejman Attar**, *Safe and secure reactive programming for multicore architectures*, 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, **Gérard Boudol** and **Tamara Rezk**.

9.3.3. Juries

Manuel Serrano was a member of the jury of the PhD Thesis of Luca Saiu (Universities Paris 13). He was a member of the selection committee of PPS (Paris 6).

9.4. 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`) [4], 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 Javascipt 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 [16]. 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.

9.5. Transfer

9.5.1. 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.

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", 2002, vol. 281, n^{o} 1, 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", May 2000, vol. 10, n^{o} 2, p. 1–43.

Publications of the year

Articles in International Peer-Reviewed Journals

[6] G. BARTHE, D. PICHARDIE, T. REZK. *A Certified Lightweight Non-Interference Java Bytecode Verifier*, in "Mathematical Structures in Computer Science", 2012, to appear.

[7] 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", 2012, vol. 34, n^{o} 2.

[8] S. CAPECCHI, I. CASTELLANI, M. DEZANI-CIANCAGLINI. *Typing Access Control and Secure Information Flow in Sessions*, in "Information and Computation, special issue on Security and Rewriting Techniques", 2012, to appear.

[9] M. MENDLER, T. SHIPLE, G. BERRY. *Constructive Boolean Circuits and the Exactness of Timed Ternary Simulation*, in "Formal Methods for Systems Design", 2012, n^{o} 3, p. 283–329.

[10] M. SERRANO, C. QUEINNEC. *A multi-tier semantics for Hop*, in "Higher Order and Symbolic Computation (HOSC)", 2012, vol. 23, n^{o} 4, p. 409-431.

Articles in Non Peer-Reviewed Journals

[11] G. BARTHE, D. PICHARDIE, T. REZK. *A Certified Lightweight Non-Interference Java Bytecode Verifier*, in "Journal of Mathematical Structures in Computer Science", 2012.

International Conferences with Proceedings

[12] A. ALMEIDA-MATOS, J. FRAGOSO SANTOS. *Typing Illegal Information Flow as Program Effects*, in "Proceedings of the 7th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security", 2012.

[13] G. BOUDOL, G. PETRI, B. SERPETTE. *Relaxed Operational Semantics of Concurrent Programming Languages*, in "Expressiveness in Concurrency/Structural Operational Semantics (EXPRESS/SOS)", EPTCS, 2012, vol. 89, p. 19-33.

[14] Z. LUO, T. REZK. *The Mashic Compiler: Mashup Sandboxing using Inter-frame Communication*, in "Computer Security Foundations (CSF)", 2012.

National Conferences with Proceeding

[15] E. CHAILLOUX, B. SERPETTE. *Séparation des couleurs dans un lambda-calcul bichrome*, in "Actes des journées JFLA", Carnac, France, 2012.

Scientific Popularization

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