



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Moscova

*Mobility, Security, Concurrency,
Verification and Analysis*

Rocquencourt

THEME 1C

Activity
R
Report

2003

Table of contents

1. Team	1
2. Overall Objectives	1
3. Scientific Foundations	2
3.1. Concurrency theory	2
3.2. Type systems	2
4. Application Domains	2
4.1. Telecoms and Interfaces	2
5. Software	3
5.1. Acute, a prototype implementation of a type-safe distributed programming language	3
5.2. Pattern matching in Ocaml	3
5.3. Hevea	3
6. New Results	4
6.1. Join-Calculus	4
6.2. Type safety for distributed computing	4
6.3. Semantics for distributed programming with dynamic rebinding and version control	4
6.4. A security analysis for libraries	5
6.5. A framework for history based stack inspection	5
6.6. Typing dynamic linking	5
6.7. Objective-Join	6
6.8. FOC project	6
6.9. A mechanical proof of the Four Color theorem using the Calculus of Constructions	7
6.10. Fault tolerancy and safety properties for mobile and concurrent systems	8
7. Contracts and Grants with Industry	9
7.1. Ozone	9
8. Other Grants and Activities	9
8.1. National actions	9
8.1.1. ACI Modulogic	9
8.1.2. France telecom	9
8.2. European actions	9
8.2.1. Projet PEPITO	9
9. Dissemination	10
9.1. Animation of research	10
9.2. Teaching	10
9.3. Partipations to conferences, Seminars, Invitations	10
9.3.1. Participations to conferences	10
9.3.2. Other Talks	11
9.3.3. Visits	11
10. Bibliography	12

1. Team

Head of project-team

Georges Gonthier [from 01/01/2003 to 10/30/2003]

Jean-Jacques Lévy [from 12/01/2003 to 12/31/2003]

Vice-head of project team

Luc Maranget [CR]

Administrative assistant

Sylvie Loubressac [TR]

Staff member Inria

Damien Doligez [CR]

James Leifer [CR]

Jean-Jacques Lévy [DR]

Staff member University Paris 6

Thérèse Hardin [Professor]

Ph. D. student

Tomasz Blanc [détaché du Corps des TELECOM, since 08/01/2003]

Richard Bonichon [allocataire MESR, since 09/01/2003]

Jean Krivine [allocataire MESR, since 09/01/2003]

Ma Qin [allocataire MESR]

Gilles Peskine [allocataire AMN]

Student intern

Pierre Habouzit [Ecole polytechnique, from 04/01/2003 to 07/30/2003]

Jean Krivine [DEA, from 04/01/2003 to 09/30/2003]

2. Overall Objectives

The research in Moscova centers around the theory and practice of concurrent programming in the context of distributed and mobile systems.

Concurrent programming is delicate. It requires a clear understanding of the underlying models and programming primitives. Our systems are asynchronous, loosely coupled, reconfigurable, and allow mobility. We started from programming languages such as B. Thomsen's Facile [26], Cardelli's Obliq at DEC/SRC, Pierce and Turner's Pict [25], and Sewell and Wojciechowski's Nomadic Pict; in 1996, we defined the Join-Calculus as a variant of Milner's π -calculus [27] stressing permanent receivers with unique locations; we developed in the last years two implementations of the Join-Calculus: the Join-Calculus language 1.05 and Jocaml.

On the theoretical side, we have worked on process algebras and their equivalences, proofs of security protocols, frameworks for flow analysis, resource control and concurrent objects. In 2002, we started a collaboration with U. of Cambridge, EPFL, SICS, KTH and UCL on symmetric distributed applications, within the framework of the Esprit Global Computing action.

This year, we have developed several new activities related to concurrency and mobile systems. J. Leifer and G. Peskine built a theory of safe communications between communicating ML processes, which allows exchanges of abstract data types values. This research is joint with Sewell and Wansbrough in U. of Cambridge. T. Blanc, a new PhD student, started a work on flow analysis by stack inspection, a work initiated with Fournet and Gordon during a long internship at Microsoft Research. J. Krivine, new PhD student, also started a work on transactional models through process algebras. Finally, G. Gonthier developed a new theory of dynamic linking with system E and Hilbert's choice operator. The work by Qin and Maranget on concurrent objects in the Join-Calculus has also been pursued in 2003.

Since 2001, the research-team Moscova has worked in formal proofs systems. G. Gonthier is continuing his very long proof of the four color theorem within Coq. D. Doligez and T. Hardin are building the FOC system. This calculus, devoted initially to certified computer algebra, has also shown itself to be important for more general specifications of programs. One of the designers, Virgile Prevosto, defended his thesis at U. of Paris 6; his supervisor was D. Doligez.

In 2003, D. Doligez and L. Maranget pursued their work in the development of the Objective Caml system, on garbage collection and pattern-matching.

G. Gonthier left the project-team Moscova, on Dec 1, 2003, and joined Microsoft Research in Cambridge. Finally, three ex-PhD students (S. Conchon, F. le Fessant, A. Schmitt) of our team in 2002 were hired at U. of Orsay, INRIA-Futurs and INRIA-Rhône Alpes in 2003.

3. Scientific Foundations

3.1. Concurrency theory

Milner started the theory of concurrency in 1980 at Edinburgh. He proposed the calculus of communicating systems (CCS) as an algebra modeling interaction [23]. This theory was the source of many articles, most of them are quite subtle. In 1989, R. Milner, J. Parrow and D. Walker [24] introduced a new calculus, the *pi-calculus* capable of handling reconfigurable systems. This theory has been refined by D. Sangiorgi (Edinburgh/INRIA Sophia/Bologna). Many variants of the pi-calculus have been developed since 1989.

We considered the variant of the Join-Calculus[5][6], a variant easier to implement in a distributed environment. Its purpose is to avoid the use of atomic broadcast to implement fair scheduling of processes. The Join-Calculus allows concurrent and distributed programming, and simple communication between remote processes. It was designed with locations of processes and channels. It leads smoothly to the design and implementation of high-level languages which take into account low-level features such as the locations of objects.

The Join-Calculus has higher-order channels as in the pi-calculus; channels names can be passed as values. However there are several restrictions: a channel name passed as argument cannot become a receiver; a receiver is permanent and has a single location, which allows one to identify channel names with their receivers. The loss of expressivity induced by these restrictions is compensated by joined receivers. A guard may wait on several receivers before triggering a new action. This is the way to achieve rendez-vous between processes. In fact, the notation of the Join-Calculus is very near the natural description of distributed algorithms.

The second important feature of the Join-Calculus is the location. A location is a set of channels co-residing at the same place. The unit of migration is the location. Locations are structured in a tree of locations. A location migrates with all its sublocations.

The Join-Calculus, renamed Jocaml, has been fully integrated into Ocaml. Locations and channels are new features; they may be manipulated by or can handle any Ocaml values. Unfortunately the newer versions of Ocaml are not supporting it. We are still planning for both systems to converge.

3.2. Type systems

Types[7] are used in the theory of programming languages to guarantee (usually static) integrity of computations. Types are also used for fancier static analysis of programs. The theory of types is used in Moscova to insure safety properties about abstract values exchanged by two run-time environments.

4. Application Domains

4.1. Telecoms and Interfaces

Key words: *telecommunications, distributed applications, security, verification.*

Distributed programming with mobility appears in the programming of the World Wide Web and in autonomous mobile systems. It can be used for customization of user interfaces and for communications between several clients. The telecommunications are an other example of application, with active networks, hot reconfigurations, and intelligent systems. For instance, France Telecom (Lannion) is designing a system programmed in mobile Erlang.

5. Software

5.1. Acute, a prototype implementation of a type-safe distributed programming language

Participants: James Leifer, Peter Sewell [U. of Cambridge], Keith Wansbrough [U. of Cambridge].

Acute provides a test implementation of our current work on type safe semantics for distributed computing. As such, Acute is equipped with a full semantic definition (of typing, compilation, and execution).

Acute is an interpreted language whose core features consist of a subset of Ocaml's. It includes a module language (though no functors); so-called "marks" for delimiting ranges of modules to be shipped or rebound when marshalling values; a language of version numbers attached to module declarations; and a language of version constraints, which are attached to imports.

Though an interpreter, Acute has two distinct phases: the first performs compilation (preprocessing) and the second run-time execution. The former does type checking and calculates hash fingerprints for the modules that export abstract types. At run time, the marshalling primitive transforms a value into a string, packing in the modules to be shipped with the value (as specified by the mark scoping) and value's type. The unmarshalling primitive unpacks the value, checking the actual and expected types to ensure type safety and, thanks to the compile-time calculation of hash types, abstraction safety. The value may have intentionally been shipped without all required modules in order to force relinking to local resources at the receiver side: this relinking involves a version-satisfaction relation that verifies that the version constraints needed by the value are satisfied by the version numbers of the locally available modules.

Our prototype consists of about 10 000 lines of code and is written in FreshOcaml, a variant of INRIA's Ocaml with support for automatic alpha conversion. It is expected to be released in December 2003.

5.2. Pattern matching in Ocaml

Participant: Luc Maranget.

Key words: *ocaml, pattern matching.*

Luc Maranget has worked on warning messages for pattern matching in the Objective Caml compiler. His results were published in the *Journées Francophones des langages applicatifs* [14]. This year Luc Maranget implemented additional analysis related to pattern-matching. When activated, the new feature send warning messages about pattern matching that would probably be wrong (and would not be flagged by others warnings) if the matched value type is altered and the whole program recompiled. The new feature facilitates Objective Caml usage in a industrial context since it helps to enforce strict coding rules aiming at producing robust code. This has been incorporated in the standard distribution of Ocaml since Ocaml 3.06.

5.3. Hevea

Participant: Luc Maranget.

Key words: *tex, latex, html, web.*

Hevea is a fast translator from L^AT_EX to HTML, written in Objective Caml. Hevea was first released in 1997 and is still maintained and developed by Luc Maranget. A continuous (although informal) collaboration around Hevea exists, including Philip H. Viton (Ohio State University) for the MS Windows port and Ralf Treinen

(ENS Cachan) for Debian developments. Hevea consists in about 20 000 lines of Caml and about 5 000 lines of package sources written “almost” in $\text{T}_{\text{E}}\text{X}$ (the language understood by Hevea). In November, Hevea was downloaded from 300 sites and is now part of the Debian distribution. This year saw release 1.07. Additionally Luc Maranget wrote an article for the $\text{T}_{\text{E}}\text{X}$ users association of Greece [11].

6. New Results

6.1. Join-Calculus

Participants: Georges Gonthier, Pierre Habouzit, Luc Maranget.

The work on the join-calculus and the corresponding language Jocaml has been slowed down in 2003. The main implementer Fabrice le Fessant could not be hired at INRIA Rocquencourt.

Anyhow, G. Gonthier finalized his journal paper with C. Fournet (Microsoft) on a the hierarchy of equivalences for asynchronous process calculi [10].

P. Habouzit studied the distributed garbage collector of le Fessant during his summer internship. He achieved a graphical demo, written in C++, for simulation of le Fessant’s algorithm.

6.2. Type safety for distributed computing

Participants: James Leifer, Gilles Peskine, Peter Sewell [U. of Cambridge], Keith Wansbrough [U. of Cambridge].

Type abstraction is a key feature of ML-like languages for writing large programs. Marshalling is necessary for writing distributed programs, exchanging values via network byte-streams or persistent stores. In this work we combine the two, developing compile-time and run-time semantics for marshalling, that guarantee abstraction-safety between separately-built programs.

We obtain a namespace for abstract types that is global, i.e. meaningful between programs, by hashing module declarations. We examine the scenarios in which values of abstract types are communicated from one program to another, and ensure, by constructing hashes appropriately, that the dynamic and static notions of type equality mirror each other. We use singleton kinds (Harper and Lillibridge) to express abstraction in the static semantics; abstraction is tracked in the dynamic semantics by colored brackets. These allow us to prove preservation, erasure, and coincidence results.

Publications: [13] and [19].

6.3. Semantics for distributed programming with dynamic rebinding and version control

Participants: James Leifer, Peter Sewell [U. of Cambridge], Keith Wansbrough [U. of Cambridge].

As distributed computation becomes commonplace, it will be increasingly useful to be able to exchange arbitrary language values, preserving their structure, rather than require the programmer to translate back and forth to some “external” formats (which would have to be kept in sync with the internal data structures). Moreover, the difficulty of debugging such systems makes it important to detect errors as early as possible — as far as possible, interaction should be guaranteed to be type-safe.

This work goes beyond the areas normally considered in language semantics (sequential or concurrent) by addressing not just distributed execution and communication, but distributed development and deployment taking place across many administrative domains on a long time-scale. For such applications it is often impossible to force software updates to be synchronized: there may be many coexisting versions of many applications, sharing some libraries but not others, that need to communicate.

We address the semantics of ML-like programming languages for distributed computation, focusing on support for type-safe marshalling of arbitrary language values. In particular: (1) unmarshalling can involve *rebinding* to local resources available on the receiver’s side of a communication; (2) values of *abstract*

types can be communicated, and a globally-coherent notion of type equality ensures that unmarshalling respects abstraction; and (3) interoperation between separately-built programs with different *versions* of shared modules is supported, with fine-grain version control.

Publications: [20].

6.4. A security analysis for libraries

Participants: Tomasz Blanc, Cédric Fournet [Microsoft Research], Andrew Gordon [Microsoft Research].

We considered a new formal static analysis to help identify security defects in libraries for runtimes, such as the JVMs or the CLR, that relies on stack inspection for access control. Our tool inputs a set of libraries plus a description of the permissions granted to unknown, potentially hostile code. It constructs a permission-sensitive call graph, which can be queried to identify potential defects. We describe the tool architecture, various examples of security queries, and a practical implementation that analyses large pre-existing libraries for the CLR. We also develop a new formal model of the essentials of access control in the CLR (types, classes and inheritance, access modifiers, permissions, and stack inspection) In this model, we state and prove the correctness of the analysis.

This work has been submitted for publication, coauthored with A. Gordon and C. Fournet (Microsoft Research Cambridge).

6.5. A framework for history based stack inspection

Participants: Tomasz Blanc, Jean-Jacques Lévy.

The initial work by Gordon, Fournet can be revisited via a history-based labeled lambda calculus. Standard properties such as confluency in the labeled calculus stress different algebraic laws from the ones in the initial framework for stack inspection by Gordon and Fournet. This preliminary work is a starting point for a formal theory of stack inspection based on the history of redexes such as sketched by Abadi and Fournet.

This work can be useful for security analysis of reconfigurable systems based on tracing of new interactions.

6.6. Typing dynamic linking

Participant: Georges Gonthier.

In most modern operating systems, applications access system and user program libraries using *dynamic linking*. This allows several applications to coexist while sharing code and, more importantly, implementation of basic data types, which means they can interoperate efficiently. As a rule, a software module is written and tested under the assumption that it will be executed with adequate libraries, and that all other modules will be using the same libraries.

Unfortunately, modeling dynamic linking in the standard module type system, based on the Harper-MacQueen existential types, is quite awkward. Since linking in this system is represented by an explicit functor application, it is necessary to include the dynamic linking code along with the user code. This breaks abstraction, as dynamic linking is a system operation, and should therefore be part of the language semantics.

Starting with a visit to UCSC in December 2002, G. Gonthier reformulated his work with M. Abadi (U. Santa Cruz) and B. Werner (Logical) on a new approach to typing dynamically linked modules, based on a second-order version of Hilbert's choice symbol (usually denoted ε), rather than existential quantifiers.

They succeeded in overcoming the technical limitations of an earlier attempt at exploiting this insight (in 2000), which were caused by incompatibilities between the intricate cut elimination procedure of the ε -calculus, and the natural order of computation for functional programs. The improvement comes from replacing the structural comparison of interfaces (a library is identified by the text of its description) by name comparison (a library is identified by a unique reference name). This change simplifies the type system, and simultaneously restricts the underlying logic to the so-called ε^* fragment of the ε calculus, whose computational interpretation is much simpler.

The resulting type system, dubbed System E, is an improvement in almost every respect on its predecessor. System E can, for instance, be consistently extended with parametric polymorphism and explicit modules. Moreover a novel but plausible abstract machine was designed, that implements System E dynamic linking. Remarkably, the machine can handle incremental libraries, in which the implementation of an interface depends on another, previous implementation of the same interface. The machine achieves this by using a novel form of dynamic binding, based on a digest of the call context, to select the appropriate version of a library.

6.7. Objective-Join

Participants: Ma Qin, Luc Maranget.

Combining object oriented and concurrent programming is one of our long-term goals. We start from the initial work by Didier Rémy (project-team CRISTAL), Cédric Fournet (Microsoft Research) and Cosimo Laneve (Universita di Bologna) [9]. This work presents a class-based object-oriented extension of the join-calculus. Current work on this topics is performed by Ma Qin and Luc Maranget. Ma Qin now completes her second thesis year, her tutors are Luc Maranget and Pierre-Louis Curien (team PPS, CNRS).

Ma Qin's thesis work consists in integrating the approach of [9] in a real programming language. However this approach looks quite complex and may prove difficult use in practice as it stands now. One can think for instance about sacrificing a bit of expressivity so that to produce a simpler system. Our work on the practical side of an object extension to the join-calculus made good progress this year, with unexpected results. We showed that the original system is not as expressive as we first thought. Types do not contain enough information on object synchronization to allow the typing of quite ordinary (and obviously type-error free) classes built by inheritance. Abstraction on types as performed by this original system was somehow artificial and resulted in types that would have been difficult to understand by programmers and even more difficult to write, as modular programming requires. This reinforces our statement on the complexity of the system, which is a serious drawback not only for developing a practical compiler but also for writing programs. We now propose a conceptually simpler, more expressive, typing system. The key idea lies in including all synchronization information in class types. From the join-calculus point of view this amounts to including join-patterns in class types and strengthening the common view of classes as templates for objects. As our solution hinders program abstraction we are now working on restoring abstraction by method hiding. We plan to investigate the impact of this simple solution on the synchronization information present in class types.

This work is published this year at the *The First Asian Symposium on Programming Languages and Systems* [15].

6.8. FOC project

Participants: Richard Bonichon, Damien Doligez, Thérèse Hardin, Virgile Prévosto, Pierre Weis [Project-team Cristal].

The FOC project is a joint effort of three teams (D. Doligez, P. Weis at INRIA; T. Hardin, M. Jaume, V. Ménéssier-Morain, R. Rioboo at LIP6; D. Delahaye, C. Dubois, O. Pons, V. Vigié at CNAM-CEDRIC; and several PhD students).

The objective of the FOC project is the design and implementation of a development framework for certified and modular software, in which required properties are formally asserted. Its features support the description of the requirements (by declarations and statements); then a step-by-step introduction of new statements, declarations; and, finally, the implementation for which every declaration has been associated to a definition and every statement has received a proof.

Declarations, definitions, statements, and proofs are recorded in FOC *units* (which correspond to classes or modules, in other programming languages). Units are grouped into libraries. Several tools are given to build new units: multiple inheritance, late binding, full parameterisation upon units and values. The proofs are currently done with the Coq proof assistant. Once achieved and checked, the proof is recorded in the unit and stored for *a posteriori* verification purposes.

The presence of proofs and their combination with object-oriented features introduces potential for inconsistencies. Syntactic restrictions can prevent some of these inconsistencies. The compiler performs static analyses to detect the remaining inconsistencies, and rejects inconsistent units. After checking the units, the compiler performs translations into Ocaml, and also into documentation languages (Latex and XML). The compiler also synthesizes all the information coming from the structure of the unit and translates it to Coq in order to provide the user with the correct context to build the proofs. With this context, the proof of an implementation is based not only on the properties of the algorithm, but also on the framework where the definition is inserted. This technique also checks inter-unit consistency.

The first release of the language was done in July 2003. It contains the compiler, a library of Computer Algebra algorithms devoted to polynomials, and several development tools, among them an automatic documentation tool, based on Openmath, to allow interfacing with other Computer Algebra systems.

Virgile Prévosto has completed his PhD, which was devoted to the design and implementation of the FOC compiler. His dissertation also contains the proof of correctness of the compiler with respect to the formal specification of FOC written by S. Boulmé in Coq. Damien Doligez and Thérèse Hardin worked on various aspects of FOC, among them the design of a proof search tool based on the tableaux method. Richard Bonichon, a new PhD student, is working on a Tableaux-based version of deduction modulo[8]. The FOC group is involved in the research action “ACI Modulogic”, which is coordinated by T. Hardin. Recently, Pierre Weis joined the FOC group, where he will handle compiler development.

6.9. A mechanical proof of the Four Color theorem using the Calculus of Constructions

Participants: Georges Gonthier, Benjamin Werner [Project team LOGICAL].

The four color theorem asserts that four colors suffice to paint the regions of any planar map in such a way that any two adjacent regions are given different colors. It is one of the more famous results in discrete mathematics, largely because it took over a century, and a breakthrough use of computers, to prove it. However this very use of computer programs casts some doubt on the proof, at least on a philosophical level, because the validity of the proof becomes contingent on the informal, often ambiguous, semantics of the programming language.

As the computer cannot be dispensed of, because of the combinatorial complexity of the proof, the only way to avoid the above problems is to use formally verified programs, which means carrying the entire proof inside a formal demonstration system. This makes the four color theorem an ideal case study to test and demonstrate the maturity of a formal proof assistant such as the Coq system developed by the Logical research team. Building on his 1995 experience on the formal verification of the safety of a concurrent garbage collection algorithm, G. Gonthier has undertaken, with B. Werner (Logical) the implementation in Coq of a recent (1997) proof of the four color theorem due to Robertson, Sanders, Seymour and Thomas.

Like the first (1994) Appel and Haken proof, this proof has two parts:

1. Check, by enumerating all maps of radius 2 consisting of penta-to-octagonal regions around a penta-to-unidecagonal hub, that the (suitably modified) Euler formula implies that any near 6-connected cubic planar map contains one of 633 specific submaps.
2. Check, by analyzing the structure of its set of border colorings, that each of these 633 sub-maps is “reducible”, that is, that the task of coloring any planar map that contains one of these submaps can be reduced to one of coloring a smaller map.

It is the latter part that mandates the use of a computer. Indeed in their initial proof Appel and Haken manually checked over 10 000 cases to establish part 1, but this brave effort could not be carried over to part 2, where most of the 633 submaps require checking over 20 000 000 coloring topologies.

The Coq implementation therefore started with the combinatorial aspect of task 2, as the this was clearly the one which would test the performance of the system – fully verified programs can take significantly more time and space to execute than their equivalent informal versions. This implementation used reflection, that

is, the possibility of embedding ML-like programs inside logical statements of the Calculus of Inductive Constructions, and running them as part of the verification process. Using reflection, the problem of verifying a complex result is reduced to that of verifying the correctness of a program that establishes it, which in this case means substantial size savings in both proof script (the user input to Coq) and proof witness (the “compiled proof” that is passed on to the kernel checker of Coq).

Although the implementation needed to use somewhat elaborate algorithms to compensate the limitations of the Coq language (in this case, the absence of arrays), this first task went surprisingly well. One of its offshoots was a new Coq proof script style that allows a more effective combination of deductive, equational, and computational proof steps. Several features of this style have already been adopted in the recent versions of the Coq system.

The second subtask was implementing the basic planar graph theory in which the purely combinatorial results of the first subtask could be interpreted. This proved to be surprisingly difficult, mostly because the “obvious” demonstrations of many of the results in this theory are based on visual inspections that are extremely tedious to formalize, and use topological arguments outside the scope of the Coq formalization (doing the theory of complex integration to establish the Jordan Curve Theorem being out of the question). Instead, a new, purely combinatorial, definition of planar maps was used, rediscovering and extending results of Tutte and Stahl. This definition is highly regular, extending the familiar map-graph duality to a triple edge/node/face duality, and allows a simple, purely combinatorial definition of the Jordan Curve property, which is shown to be equivalent to the Euler formula. This work was completed in 2002, and presented at an invited lecture at TLCA’03.

The third subtask, carried out in 2003, was the integration of the results of the two first, to yield the theorems really covering part 2. One of these theorems, due to Birkhoff, asserts that coloring any cubic map that is *not* near 6-connected can be reduced to coloring a smaller map. Its proof turned out to be a very good test case for the integration process, because it could be reduced to a variant of the reducibility check, using the verified programs developed in the first subtask. The main difficulty in the integration subtask was that it was necessary to completely unify the basic data types used in the first two subtasks, as the complexity of dealing with two different representations (even if only slightly so) turned out to be overwhelming. The unification process was complex because it involved unifying the properties describing the representations, not just the data types. The unification process required several iterations, each of which meant a pass over most of proof scripts developed in the first two subtasks. It was an excellent test of the robustness of the proof script style that was developed for this project.

The stage is now set for a fourth subtask, which would complete the proof by establishing part 1. Most of this task is programming-oriented: compiling the list of 633 submaps into a (reflected) program that checks for their presence in a radius-2 submap, and verifying the correctness of that compilation. Based on the experience with the first subtask, which was also program-oriented, it can be expected to proceed more smoothly than the second and third subtasks.

6.10. Fault tolerancy and safety properties for mobile and concurrent systems

Participants: Vincent Danos [PPS], Jean Krivine.

Classical models for distributed systems (CCS, pi-calculus, Join-calculus) offer unidirectional semantics. They model normal evolution of distributed systems through time by the observation of their visible communications. The study of distributed system behavior in the presence of failure can be critical. In particular, we may want to be able to tolerate faults and to recover to a consistent state. This problem is classical in the design of distributed systems, where the failure of a process forces it to rollback to the last saved state.

We want to use a reversible process algebra to express reversibility in the syntax of processes, and to be able to track dependencies of communications between processes. When a failure occurs, a process may return to a previous state by canceling its communications. This framework, named CCS-R, is proved minimal in the sense that it captures equivalence up to permutation of independent transitions and is not sensitive to canceled branches of the calculus.

A first version of this setting was considered in the case of formal molecular biology. Regev, Shapiro and a number of authors have proposed that process algebras such as the pi-calculus, developed for the modeling of communication in decentralized computational systems, might be useful in biology. In pi-calculus, there is much latitude in the way a given biological phenomenon is represented. Therefore, we think that a more restrictive setting will better correspond to biological systems (such as the EGF-MAPK cascade in Regev's formalization).

Moreover, the examples that were given so far are always running forward, rarely letting a binding or an activation be undone. While this might be a quite reasonable assumption in some cases, for instance when describing a signal transduction pathway, it is certainly a departure from actual biochemistry where practically all reactions are reversible.

This has been presented at the BioConcur workshop [12].

7. Contracts and Grants with Industry

7.1. Ozone

Participant: Georges Gonthier.

Moscova contributes to Inria's participation, coordinated by V. Issarny (project-team ARLES), in the European Project Ozone, on the development of a demonstration platform for Ambient Intelligence Systems, with industrial partners Philips and Thomson.

The Moscova contribution concerns the security and privacy features of the system. On the one hand, during monthly meetings held between G. Gonthier, J. Shao (Thomson), and P. Lenoir (Philips), a novel architecture for open software Digital Copyright Management was developed, for which patent applications are under way. On the other hand, G. Gonthier elaborated on the ideas underlying the digital rights architecture to produce a security policy specification language design [18][17].

8. Other Grants and Activities

8.1. National actions

8.1.1. ACI Modulogic

This project is a consortium with CNAM (C. Dubois), LIX (G. Dowek), project-team PROTHEO (I. Gnaedig), project-team MIRO (L. Liquori), LIP6 (T. Hardin) and MOSCOVA (D. Doligez). The aim is to build secure software with the use of formal specifications. The challenges are to fully specify a programming language with the ρ -calculus, to build proof tools based on *deduction-modulo* (a logical system design by G. Dowek, C. Kirchner, T. Hardin), and to compile these proofs into Coq.

8.1.2. France telecom

J.-J. Lévy is co-supervisor, with P. Crégut, of the doctoral work of Guillaume Chatelet, who is working on extensions of Erlang with primitives for mobility. This PhD will be defended in 2004 at Ecole polytechnique.

8.2. European actions

8.2.1. Projet PEPITO

2003 is the second year of the European project PEPITO (*Peer to Peer, Implementation and Theory*), part of the Global Computing action inside Esprit. This project is a consortium with KTH (Seif Haridi, coordinator), U. of Cambridge (Peter Sewell), EPFL (Martin Odersky), SICS (Per Brand), UCL (Peter van Roy). It is concerned with theoretical and practical aspects of distributed and symmetric applications, with mobility. It federates efforts already started with Mozart (Mobile Oz), JoCaml and Scala.

We had a review meeting in Rovereto (February 2003). Our contribution was mainly around calculi for resource controls (Dynamic Join and M-calculus by A. Schmitt) and is now mostly around Type Safe Marshalling, a common effort with U. of Cambridge, which can be viewed as an other way of using safe resources. The previous work by A. Schmitt (who spent a post-doctoral year at U. of Pennsylvania) is now finished and has been presented at POPL 2003 [16].

9. Dissemination

9.1. Animation of research

G. Gonthier is vice-chairperson of the scientific committee of Inria Rocquencourt, and suppleant member of the Inria evaluation commission. He represented Inria at the steering committee of the Cea-EdF-Inria schools.

T. Hardin is vice-chairperson of SPECIF, in charge of research, since June 2001.

J.-J. Lévy has participated to the PhD juries of Marc Lacoste (Imag) and Francesco Zappa Nardelli (ENS).

9.2. Teaching

Our project-team participates to the following courses:

- “Informatique fondamentale”, 2nd year course at the Ecole polytechnique, 250 students (J.-J. Lévy, professor in charge of this course; G. Gonthier and L. Maranget, professors “chargés de cours”). G. Gonthier was chairing the programming projects; J.-J. Lévy edited new lecture notes (270 pages) available on the Web[21]. The contract of Georges Gonthier with Polytechnique was finished on Aug 31, 2003.
- Compilers, 3rd year course at the Ecole polytechnique, 20 students (L. Maranget, professor “chargé de cours”; J. Leifer, “vacataire”, assisted him in laboratory courses). This course is available on the Web[22].
- Concurrency, “DEA Sémantique, preuves et programmation” at U. of Paris 7, 40 students, (Jean-Jacques Lévy and Catuscia Palamidessi from project-team COMÈTE)
- Lambda Calculus, ‘DEA Sémantique, preuves et programmation’ at U. of Paris 7, 40 students, (T. Hardin)

J.-J. Lévy co-authored 3 computer science exercices (4h+2h+2h) of the entrance examination at the Ecole polytechnique in 2003. He has participated to the reform of this examination for Computer Science. This may influence teaching of computer science in the “classes préparatoires”.

9.3. Participations to conferences, Seminars, Invitations

9.3.1. Participations to conferences

- Jan 12-18, J.-J. Lévy attended the POPL’03 conference in New Orleans. He also participated to PADL and FOOL.
- Jan 5-10, D. Doligez participated to a school at Dagstuhl on Verification.
- Jan 26-28, L. Maranget gave a talk at JFLA’03 in Chamrousse.
- Feb 9-14, J. Leifer, J.-J. Lévy participated to the Global Computing meeting and Pepito review at Rovereto.
- Feb 15-19, D. Doligez attended the MKMnet conference in Bertinoro.
- Feb 26, G. Gonthier participated to an Ozone meeting in Rennes.
- Mar 8-17, J. Leifer and G. Peskine visited P. Sewell in U. of Cambridge.

- Mar 14-15, J.-J. Lévy attended the ICALP program committee meeting in Harlem.
- Apr 3-4, D. Doligez and J.-J. Lévy participated to the meeting in Honour of Leslie Lamport at U. of Rennes.
- Jun 12-15, J.-J. Lévy participated to the inauguration of the new computer science building at U. of Pisa.
- Jun 22-25, G. Gonthier participated to the LICS conference and attended workshops on game semantics, quantum computing, and mobile computing in Ottawa in June.
- Jul 5-12, Ma Qin attended the Global Computing School at U. of Edinburgh.
- Jul 9-12, G. Gonthier was attended the TLCA'03 conference in Valencia.
- Sep 8-12, D. Doligez attended the Tableaux'03 and Calculemus meetings in Rome.
- Aug 24-29, J. Leifer and G. Peskine presented an article at ICFP in Uppsala. J. Leifer also attended the Erlang workshop.
- Sep 2-7, J. Krivine and J.-J. Lévy attended CONCUR'03 in Marseille. J. Krivine presented an article at the BioConcur workshop.
- Oct 23-24, G. Gonthier visited Microsoft Research in Cambridge.
- Sep 30-Oct 3, D. Doligez visited the Protheo project in Nancy
- Nov 27-29, Ma Qin and L. Maranget presented an article at APLAS'03 in Beijing.
- Nov 29-Dec 3, J. Leifer participated to a Pepito meeting in Stockholm.
- Nov 25, J.-J. Lévy visited LAAS in Toulouse for the organization of the IFIP TCS 2004 conference.
- Dec 10-12, D. Doligez attended a meeting in Rennes about the ACI Logic Security.

G. Gonthier served on the program committee of Asian'03. J.-J. Lévy served on the program committee of ICALP'03.

9.3.2. Other Talks

- G. Gonthier was invited one week in December 2002 at UCSC by M. Abadi to work on dynamic linking.
- G. Gonthier gave a talk on dynamic linking at Microsoft Cambridge in September.
- J.-J. Lévy gave a talk on the Join Calculus at U. of Pisa in June.

9.3.3. Visits

- April + December, P. Sewell visited Rocquencourt for collaboration with J. Leifer and G. Peskine.
- October, M. Shinwell visited Rocquencourt for discussion on Fresh ML.
- Jun 27, O. Tardieu gave a talk on causality in Esterel

10. Bibliography

Major publications by the team in recent years

- [1] M. ABADI, L. CARDELLI, P.-L. CURIEN, J.-J. LÉVY. *Explicit Substitutions*. in « Journal of Functional Programming », number 4, volume 1, 1991, pages 375–416.
- [2] M. ABADI, C. FOURNET, G. GONTHIER. *Secure implementation of channel abstractions*. in « Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science », pages 105–116, June, 1998.
- [3] P.-L. CURIEN, T. HARDIN, J.-J. LÉVY. *Confluence Properties of Weak and Strong Calculi of Explicit Substitutions*. in « Journal of the ACM », number 2, volume 43, March, 1996, pages 362–397, <ftp://theory.lcs.mit.edu/pub/jacm/jacm.bib>.
- [4] A. DEUTSCH. *On The Complexity of Escape Analysis*. in « 24th Annual ACM Symp. on Principles of Programming Languages », ACM Press, pages 358–371, January, 1997.
- [5] C. FOURNET, G. GONTHIER. *The Reflexive Chemical Abstract Machine and the Join-Calculus*. in « Proceedings of the 23rd Annual Symposium on Principles of Programming Languages (POPL) (St. Petersburg Beach, Florida) », ACM, pages 372–385, January, 1996.
- [6] C. FOURNET, G. GONTHIER, J.-J. LÉVY, L. MARANGET, D. RÉMY. *A Calculus of Mobile Agents*. in « CONCUR '96: Concurrency Theory (7th International Conference, Pisa, Italy, August 1996) », series LNCS, volume 1119, Springer, U. MONTANARI, V. SASSONE, editors, pages 406–421, 1996.
- [7] B. C. PIERCE. *Types and Programming Languages*. The MIT Press, 2002.

Articles in referred journals and book chapters

- [8] G. DOWEK, T. HARDIN, C. KIRCHNER. *Theorem proving modulo*. in « Journal of Automated Reasoning », 2003, to be published.
- [9] C. FOURNET, C. LANEVE, L. MARANGET, D. RÉMY. *Inheritance in the join calculus*. in « Journal of Logics and Algebraic Programming », number 1–2, volume 57, September, 2003, pages 23–29.
- [10] G. GONTHIER, C. FOURNET. *A hierarchy of equivalences for asynchronous calculi*. in « Journal of Logic and Algebraic Programming », number special issue on the π calculus, 2004, to appear.
- [11] L. MARANGET. *On using hevea, a fast \LaTeX to HTML translator*. in « Eutupon », 2004, Democritus University of Thrace.

Publications in Conferences and Workshops

- [12] V. DANOS, J. KRIVINE. *Formal molecular biology done in CCS-R*. in « Proc. BioConcur'03 », 2003, <http://pauillac.inria.fr/~krivine/index.html>.

- [13] J. J. LEIFER, G. PESKINE, P. SEWELL, K. WANSBROUGH. *Global abstraction-safe marshalling with hash types*. in « Proc. 8th ICFP », 2003, <http://pauillac.inria.fr/~leifer/research.html>.
- [14] L. MARANGET. *Les avertissements du filtrage*. in « Journées francophones des langages applications », Chamrousse, January, 2003, In French.
- [15] M. QIN, L. MARANGET. *Expressive Synchronization Types for Inheritance in the Join Calculus*. in « Proceedings of APLAS'03 », series LNCS, Springer, Beijing China, November, 2003.
- [16] A. SCHMITT, J.-B. STEFANI. *The M-calculus: a higher-order distributed process calculus*. in « Proc. of the 30th ACM POPL conference », 2003.

Internal Reports

- [17] G. GONTHIER. *A Security and Privacy Policy Description Language for Ozone*. Technical report, IST Ozone Project, 2003.
- [18] G. GONTHIER, P. J. LENOIR, J. SHAO. *Security Service Architecture for Content Protection Functionality within Ozone*. Technical report, IST Ozone Project, 2003.
- [19] J. J. LEIFER, G. PESKINE, P. SEWELL, K. WANSBROUGH. *Global abstraction-safe marshalling with hash types*. Technical report, number RR-4851, INRIA Rocquencourt, 2003, <http://www.inria.fr/rrrt/rr-4851.html>.
- [20] J. J. LEIFER, P. SEWELL, K. WANSBROUGH. *Marshalling: Abstraction, Rebinding, and Version Control*. Technical report, INRIA Rocquencourt, To appear..
- [21] J.-J. LÉVY. *Informatique Fondamentale*. Ecole polytechnique, Janvier, 2003, <http://www.enseignement.polytechnique.fr/informatique>
- [22] L. MARANGET. *Cours de compilation*. Ecole polytechnique, Janvier, 2003, <http://www.enseignement.polytechnique.fr/profs/inform>

Bibliography in notes

- [23] R. MILNER. *Communication and Concurrency*. Prentice Hall, 1989.
- [24] R. MILNER, J. PARROW, D. WALKER. *A Calculus of Mobile Processes, Parts I and II*. in « Journal of Information and Computation », volume 100, September, 1992, pages 1–77.
- [25] B. C. PIERCE, D. N. TURNER. *Pict: User Manual*. 1997, Available electronically.
- [26] B. THOMSEN, L. LETH, T.-M. KUO. *A Facile Tutorial*. in « CONCUR '96: Concurrency Theory (7th International Conference, Pisa, Italy, August 1996) », series LNCS, volume 1119, Springer, U. MONTANARI, V. SASSONE, editors, pages 278–298, 1996.
- [27] ROBIN MILNER. *The Polyadic π -Calculus: a Tutorial*. Technical report, number ECS-LFCS-91-180, LFCS, University of Edinburgh, October, 1991, Also in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer and H. Schwichtenberg, Springer, 1993.