DELIVERABLE SUMMARY SHEET

Project Number:	IST-2001-33100
Project Acronym:	PROFUNDIS
Title:	Proofs of Functionality for Mobile Distributed Systems

Deliverable Number:	9
Due date:	Project month 36
Delivery date:	March 15, 2005

Short Description: Algebraic tehcniques for typed processes; combination of types with techniques from term rewriting to ensure properties of termination; information flow type systems that enforce non-interference for realistic assembly languages and preserved through compilation from high level languages; logical formulations of non-interference.
Application to advances programming constructs of types for: specifying the state transitions of a protocol (session types); statically ensuring bounds on the resources needed for the execution of concurrent systems.
Techniques based on control flow analysis for problems of access control policies based on stack inspection.
An abstract machine for the execution of Safe Ambients that improves the efficiency of the machine designed in Year 1.
Some experiments with integration of type systems to the verification tools developed in WP1.

Partners owning:	INRIA
Partners contributed:	FFCT, INRIA, Pisa, UU
Made available to:	

Contents

1 Summary

Globally, the work in WP 3 has proceeded according to plans. As expected, most of the work has gone into Tasks 3.3 and 3.4, and aimed at understanding the applicability of the type-based, or type-inspired, techniques developped during the project. There has been also some works on the other Tasks: part of this in the form of polished/extended versions of shorter/draft papers that appeared in last year's deliverables; other work, in Task 3.2, has continued research efforts that have been active throughout the whole project.

In this deliverable, we present the results obtained this year. We first discuss below some discrepancies w.r.t. what was announced in the TA.

- The effort on the subtask "advanced programming constructs" (Task 3.3) has been bigger than expected. The reason for this is that we have found a number of challenging cases and programming constructs that we thought worth investigating.

- In the context of access control policies based on stack inspection and dynamic security policies, and continuing a line of work that began last year, we have used static techniques (control flow analysis) that do not use type systems, although they were initially inspired by them.

- We have fewer results than expected when we wrote the TA on the subtasks "Case studies" (Task 3.4), "expressiveness" and "space in types" (Task 3.2), "type inference" (Task 3.3) We did put effort on these topics. We have however encountered unexpected technical difficulties. For instance, on "space in types", at present the situation is as follows. Ravara started some preliminary work on some notion of spatial types for processes. The basic idea was to define a decidable proof system for π-calculus processes, composed from a set of syntax directed proof rules, and a more powerful subtyping relation, where subtyping would be interpreted as entailment in some subset of spatial logic. A preliminary version of this work was presented at last year's APPSEM workshop in Tallinn, but has not lead to publishable results till now. More recently, the theme is being

also revisited by a student of Caires. In a parallel development, Caires, with Xudong Guan (postdoc researcher that joined Lisbon Profundis team around April 2004) and Vasco Vasconcelos, has been developping a different notion of spatial type, in the context of systems of concurrent objects, which seems particularly promising. In this research direction, spatial-behavioral types are being used to express interaction patterns of multi-party service providers, where a form of spatial conjunction allows us to express properties such as deadlock freeness and race freeness as an effect of spatial distribution. Although quite close to completion, this line of work is unlikely tobecome "ready"in time to any publication before the end of Profundis. Further, on this topic it should be pointed out that Ferrari (Pise) visited Lisbon, and Hugo Viera (Lisbon) is currently visiting Pise.

We were plannig to carry our main case studies using the tools, after having enhanced them with type information. Magnus Johansson (UU) has visited Bologna and Pise, with the objective to work on this topic. Here as well progress has been slow. People in Pise did some experiments but we have not written any paper on it (basically we have a front-end for MIHDA which exploits a form of type annotation in the generation of the HD automaton).

Concerning type inference, some papers on type considers it (for instance [10]) but without it as the main objective; we are currently working on problems of type inference and type checking for session types.

We do not regard the above discripancies as failures because the quality and quantity of our results in the other subtasks are satisfying, sometimes going beyond what was expected.

2 Detailed description of work

We describe the work carried out, divided by tasks.

Task 3.1

The question of access control (Task 3.1) has been studied from the point of view of static control of migrating processes in a distributed framework. We have defined a type system for $lsd\pi$, an explicitely distributed version of the π-calculus. The role of this type system is to control remote communication, process migration and channel creation, in order to guarantee compliance w.r.t. a control policy on migration of code. This work is reported in [14], and is a polished version of a draft that appeared in last year's Deliverable 8.

Task 3.2

The work on using types to guarantee properties of termination of processes has continued. In the previous 2 years we had experimented with techniques based on logical relations, trying to transplant proof techniques well-established in funtional languages onto the pi-calculus. However, the class of processes that we could handle had some limitations; we could not allow certain combinations of operators (example: inputs that are nested and non-replicated). We have therefore now experimented a different direction, where termination is guaranteed by means of a combination of types with techniques from term rewriting. Some case studies are reported in the full version of the paper, which we are now finishing to write.

We have been working on enhancements of algebraic and operational techniques for untyped mobile processes using type information [11, 12]. Two variants of typed bisimilarity are considered, both in their late and in their early version. For both of them, proof systems that are sound and complete on the closed finite terms are given. For one of the two variants, a complete axiomatisation for the open finite terms is also presented. The type system has capability types. They allow one to distinguish between the ability to read from a channel, to write to a channel, and both to read and to write. They also give rise to a natural and powerful subtyping relation. A conference and journal version of this work have been produced and published [11, 12].

We have also considered [9] extensions of this work to calculi with other features, notably probabilities calculi (these calculi are interesting for distributed systems, where correctness of algorithms and protocols often relies on probabilities). However, since algebraic laws for calculi with probabilities remain largely unexplored, we decided to study the problem first on a purely probabilistic calculus (without types and mobility) [9].

Task 3.3

All the papers produced on Task 3.3 regard the subtask "advanced programming constructs". On this topic we have had 4 strands of work, which are about: resource bounds; session types; information flow type systems that enforce noninterference; control flow analysis, initially inspired by type system works, in the context of access control policies based on stack inspection and dynamic security policies.

Resource bounds

The work on *'resource allocation'* had been started in Year 1 in the context of the calculus of Mobile Ambients The goal was to control the number of ambients that be at the same time within a given ambient. This work had then been extended to to related formalisms for distributed computation such as Boxed Ambients, Nomadic Pict, and the Kell calculus (these models are developed in other projects belonging to the GC initiative). This year, we tried to understand the robustness of the techniques developped, by using them to control

resources such as memory or disk usage. In order to study this problem, we have introduced [16] the Controlled π-calculus, an extension of the pi-calculus with a notion of recovery of unused resources with an explicit (parametrized) garbage-collection and dead-process elimination. We discuss the definition of garbage-collection and dead-process elimination for concurrent, communicating applications, and provide a type-based technique for statically proving resource bounds. Selected examples are presented, and show the potential of the Controlled π-calculus.

David Teller's PhD thesis [17], which presents the line of work above and was entirely carried out within Profundis, has been completed.

Following the typed assembly language (TAL) approach, we have developped [1] new methods to statically bound the resources needed for the execution of systems of concurrent, interactive threads. Our study is concerned with a synchronous model of interaction based on cooperative threads whose execution proceeds in synchronous rounds called instants. Our contribution is a system of compositional static analyses to guarantee that each instant terminates and to bound the size of the values computed by the system as a function of the size of its parameters at the beginning of the instant. Our method generalises an approach designed for first-order functional languages that relies on a combination of standard termination techniques for term rewriting systems and an analysis of the size of the computed values based on the notion of quasi-interpretation. These two methods can be combined to obtain an explicit polynomial bound on the resources needed for the execution of the system during an instant.

Information flow

We have defined [5] an information flow type system for a sequential JVM-like language that includes classes, objects, and exceptions. Furthermore, we show that it enforces non-interference. Our work provides, to our best knowledge, the first analysis that has been shown to guarantee non-interference for a realistic assembly language.

Starting from the seminal work of Volpano and Smith, there has been growing evidence that type systems may be used to enforce confidentiality of programs through non-interference. However, most type systems operate on high-level languages and calculi, and low-level languages have not received much attention in studies of secure information flow. Therefore, we have introduced [2] an information flow type system for a low-level language featuring jumps and calls, and show that the type system enforces termination-insensitive non-interference. Furthermore, information flow type systems for low-level languages should appropriately relate to their counterparts for high-level languages. Therefore, we have introduced a compiler from a high-level imperative programming language to our low-level language, and showed that the compiler preserves information flow types.

Non-interference is a high-level security property that guarantees the absence of illicit information leakages through executing programs. More precisely, non-interference for a program assumes a separation between secret inputs and pub-

lic inputs on the one hand, and secret outputs and public outputs on the other hand, and requires that the value of public outputs does not depend on the value of secret inputs. A common means to enforce non-interference is to use an information flow type system. However, such type systems are inherently imprecise, and reject many secure programs, even for simple programming languages. The purpose of the paper [4] is to investigate logical formulations of non-interference that allow a more precise analysis of programs. It appears that such formulations are often sound and complete, and also amenable to interactive or automated verification techniques, such as theorem-proving or model-checking. We illustrate the applicability of our method in several scenarii, including a simple imperative language, a non-deterministic language, and finally a language with shared mutable data structures. This work has been done with implementations in mind.

Session types

We have defined [19] a language whose type system, incorporating session types, allows complex protocols to be specified by types and verified by static typechecking. A session type, associated with a communication channel, specifies the state transitions of a protocol and also the data types of messages associated with transitions; thus typechecking can verify both correctness of individual messages and correctness of sequences of transitions. Previously session types have mainly been studied in the context of the π-calculus; instead, our formulation is based on a multi-threaded functional language with side-effecting input/output operations. Our typing judgements statically describe dynamic changes in the types of channels, our channel types statically track aliasing, and our function types not only specify argument and result types but also describe changes in channels. We have formalized the syntax, semantics and typing rules of our language, and proved subject reduction and runtime type safety theorems.

In another paper on session types [18], we have proposed the use of session types to extend with behavioural information the simple descriptions usually provided by software component interfaces. We have showed how session types allow not only high level specifications of complex interactions, but also the definition of powerful interoperability tests at the protocol level, namely compatibility and substitutability of components. We have presented a decidable proof system to verify these notions, which makes our approach of a pragmatic nature.

In [7], a new static analysis is proposed for programming languages with access control based on stack inspection. This analysis allows for various security-aware program optimizations. A novel feature of our static analysis is that it is parametric with respect to the security policy in force, so it needs not to be recomputed when the access rights are dynamically updated.

Control flow and stack inspection

In [8], A new model for access control is proposed, based on policy framings embedded into histories of execution. This allows for policies that have a possibly nested, local scope. In spite of the increased expressive power of our model, we present a way to use standard model checking for history verification.

Task 3.4

As a case study has been carried out on the work on static analysis for stack inspection [6]. Here, we consider languages that use stack inspection as an access control mechanism, and concentrate on a specific optimization technique, namely method inlining. By exploiting a static analysis, we specify when this optimization is possible, preserving the policy for access control associated with applications. Remarkably, our proposal works even in the presence of dynamic linking.

The rest of the work in this task concerns implementations. The formalization in the Isabelle/HOL proof assistant of a type system for a concurrent language with scheduling, based on work by Boudol and Castellani, has been described in [4]. This contribution represents, to the best of our knowledge, the first machine-checked account of non-interference for a concurrent language. As a benefit of using a proof assistant, we are able to deal with a more general language than the one studied by Boudol and Castellani.

The abstract machine for the execution of Safe Ambients, that was designed in Year 1, has been further studied and developped. [13] defines an optimised abstract machine and proves its correctness w.r.t. the original machine. The improved machine is made more efficient by adapting some standard algorithms in distributed programming, such as forwarder chains contraction using Tarjan sets. The correctness proof, which establishes bisimilarity between the original machine and its optimised version, is challenging, and has led to the development of some specialised proof technology. A distributed implementation of the machine has been implemented and tested on a few examples. A description of the implementation is available at [15].

There has also been work on enhancing the Profundis tool with type information, but no paper or public document has been produced yet.

3 Sum up per task

We recall below the tasks and subtasks, as given in the TA, and present our contributions in Year 3.

- Task 3.1: Interference and access control
  - Classification of interferences: finished in Year 1
  - Resource allocation: types for migration control [14]

  - Declassification: this subtask is obsolete (see Year 1's Deliverable number 3)
  - Secure composition of components: finished in Year 2 (this topic is now continued as part of Task 3.3 "advanced programming constructs").

- Task 3.2: Integration of types with operational and logic techniques
  - Operational and logical techniques for typed calculi: Algebraic tehcniques for typed processes [11, 12], and probabilistic calculi [9]; combination of types with techniques from term rewriting to ensure properties of termination [10].
  - Expressiveness: no result yet
  - Space in types: no result yet.
  - Integration of type systems: finished in Year 2
  - Approximation techniques: the work on types for termination of mobile processes can also be seen as part this subtask [10].

- Task 3.3: practicality and scalability
  - Type-inference algorithms: some papers considers it (for instance [10]) but without it as the main objective
  - Advanced programming constructs: information flow type systems that enforce non-interference for realistic assembly languages and preserved through compilation from high level languages [5, 2]; logical formulations of non-interference [3]; application to advances programming constructs of types for specifying the state transitions of a protocol (session types) [19, 18], and for statically ensuring bounds on the resources needed for the execution of concurrent systems [1, 16].

- Task 3.4: Case studies and tool development
  - Case studies: no result yet. In the context of the work on control flow analysis and access control a case study on a specific optimization technique, namely method inlining [6].
  - Implementations:

    An abstract machine [13, 15] for the execution of Safe Ambients that improves the efficiency of the machine designed in Year 1, and the formalization of a type system to guarantee non-interference [4]. Also, some experiments with integration of type systems to the verification tools developed in WP1.

4 Plans for Year 4

During 2005 we expect that the main work will be about polishing, sometimes possibly improving, the results produced during this year. At least two journal papers will be prepared; futher, a first draft of Deng's PhD thesis (funded by Profundis to work on WP3) will be ready.

A few new results and conference papers might however be produced, especially in the topic of access control policies based on stack inspection. We are also currently working on problems of type inference and type checking for session types.

The work on "space in types" and the work on the integration of types into the Profundis tool will continue. We do not expect to have papers ready by the end of the project. Certainly this work will continue after the project (for instance as part of Magnus Johansson's thesis) and we expect that some results will be obtained by the end of the year.

5 Comments

The papers [4, 2, 5, 14] are polished and extended version of draft papers that had appeared in the deliverables of the previous years.

References

[1] Roberto M. Amadio and Silvano Dal Zilio. Resource Control for Synchronous Cooperative Threads. In *CONCUR 2004 – 15th International Conference on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 68–82. Springer-Verlag, August 2004.

[2] G. Barthe, A. Basu, and T. Rezk. Security types preserving compilation. *Journal of Computer Languages, Systems and Structures*, 2005. To appear.

[3] G. Barthe, P. D'Argenio, and T. Rezk. Secure Information Flow by Self-Composition. In R. Foccardi, editor, *Proceedings of CSFW'04*, pages 100–114. IEEE Press, 2004.

[4] G. Barthe and L. Prensa-Nieto. Formally verifying information flow type systems for concurrent and thread systems. In M. Backes, D. Basin, and M. Waidner, editors, *Proceedings of FMSE'04*, pages 13–22. ACM Press, 2004.

[5] G. Barthe and T. Rezk. Non-interference for a JVM-like language. In G. Morrisett and M. Fähndrich, editors, *Proceedings of TLDI'05*. ACM Press, 2005.

[6] Massimo Bartoletti, Pierpaolo Degano, and Gianluigi Ferrari. Method inlining in presence of stack inspection. In *Workshop on Issues in the Theory of Security (WITS'04)*, 2004.

[7] Massimo Bartoletti, Pierpaolo Degano, and Gianluigi Ferrari. Program transformations under dynamic security policies. In *MEPHISTO Final Workshop*, volume 99 of *Electronic Notes in Computer Science*. Elsevier, 2004.

[8] Massimo Bartoletti, Pierpaolo Degano, and Gianluigi Ferrari. Policy framings for access control. In *Workshop on Issues in the Theory of Security (WITS'05)*, 2005.

[9] Yuxin Deng and Catuscia Palamidessi. Axiomatizations for probabilistic finite-state behaviors. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science. Springer, 2005. To appear.

[10] Yuxin Deng and Davide Sangiorgi. Ensuring termination by typability. In *Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science*, pages 619–632. Kluwer, 2004.

[11] Yuxin Deng and Davide Sangiorgi. Towards an algebraic theory of typed mobile processes. In *Proceedings of the 31th International Colloquium on Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2004.

[12] Yuxin Deng and Davide Sangiorgi. Towards an algebraic theory of typed mobile processes. *Theoretical Computer Science*, 2005. To appear.

[13] D. Hirschkoff, D. Pous, and D. Sangiorgi. An Efficient Abstract Machine for Safe Ambients. In *Proc. of COORDINATION '05*, number 2004–63, 2005. to appear.

[14] F. Martins and A. Ravara. Typing migration control in lsdpi. In *Proceedings of FCS'04*, volume 31, pages 1–12. Turku Centre for Computer Science, 2004.

[15] D. Pous. Gcpan Implementation. `http://perso.ens-lyon.fr/damien.pous/gcpan`, 2004.

[16] D. Teller. Recovering resources in the pi-calculus. In *Proceedings of IFIP TCS 2004*. Kluwer, 2004.

[17] D. Teller. *Ressources limitées pour la mobilité: utilisation, réutilisation, garanties*. PhD thesis, École doctorale MathIF, ENS Lyon, 2004.

[18] Antonio Vallecillo, Vasco T. Vasconcelos, and António Ravara. Typing the behavior of software components using session types. Technical report, December 2004. Revised and extended version of Typing the Behavior of Objects and Components using Session Types. In Foclasa 2002, 1st International Workshop on Foundations of Coordination Languages and Software Architectures. Electronic Notes in Theoretical Computer Science, 68(3), 2002.

[19] Vasco T. Vasconcelos, António Ravara, and Simon Gay. Session types for functional multithreading. In *CONCUR'04*, volume 3170 of *Lecture Notes in Computer Science*, pages 497–511. Springer Verlag, 2004.

WP3 Scientific Contributions

APPENDIX

Appendix 1.1.1
F. Martins and A. Ravara. Typing migration control in lsdpi. In *Proceedings of FCS'04*, volume 31, pages 1–12. Turku Centre for Computer Science, 2004.

Appendix 1.2.1
Yuxin Deng and Catuscia Palamidessi. Axiomatizations for probabilistic finite-state behaviors. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science. Springer, 2005. To appear.

Appendix 1.2.2
Yuxin Deng and Davide Sangiorgi. Ensuring termination by typability. In *Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science*, pages 619–632. Kluwer, 2004.

Appendix 1.2.3
Yuxin Deng and Davide Sangiorgi. Towards an algebraic theory of typed mobile processes. In *Proceedings of the 31th International Colloquium on Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2004.

Appendix 1.2.4
Yuxin Deng and Davide Sangiorgi. Towards an algebraic theory of typed mobile processes. *Theoretical Computer Science*, 2005. To appear.

Appendix 1.3.1
D. Teller. Recovering resources in the pi-calculus. In *Proceedings of IFIP TCS 2004*. Kluwer, 2004.

Appendix 1.3.2 Roberto M. Amadio and Silvano Dal Zilio. Resource Control for Synchronous Cooperative Threads. In *CONCUR 2004 – 15th International Conference on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 68–82. Springer-Verlag, August 2004.

Appendix 1.3.3
Vasco T. Vasconcelos, António Ravara, and Simon Gay. Session types for functional multithreading. In *CONCUR'04*, volume 3170 of *Lecture Notes in Computer Science*, pages 497–511. Springer Verlag, 2004.

Appendix 1.3.4
Antonio Vallecillo, Vasco T. Vasconcelos, and António Ravara. Typing the behavior of software components using session types. Technical report, December 2004. Revised and extended version of Typing the Behavior of Objects and Components using Session Types. In Foclasa 2002, 1st International Workshop on Foundations of Coordination Languages and Software Architectures. Electronic Notes in Theoretical Computer Science, 68(3), 2002.

Appendix 1.3.5
G. Barthe, A. Basu, and T. Rezk. Security types preserving compilation. *Journal of Computer Languages, Systems and Structures*, 2005. To appear.

Appendix 1.3.6
G. Barthe, P. D'Argenio, and T. Rezk. Secure Information Flow by Self-Composition. In R. Foccardi, editor, *Proceedings of CSFW'04*, pages 100–114. IEEE Press, 2004.

Appendix 1.3.7
G. Barthe and T. Rezk. Non-interference for a JVM-like language. In G. Morrisett and M. Fähndrich, editors, *Proceedings of TLDI'05*. ACM Press, 2005.

Appendix 1.3.8
Massimo Bartoletti, Pierpaolo Degano, and Gianluigi Ferrari. Program transformations under dynamic security policies. In *MEPHISTO Final Workshop*, volume 99 of *Electronic Notes in Computer Science*. Elsevier, 2004.

Appendix 1.3.9
Massimo Bartoletti, Pierpaolo Degano, and Gianluigi Ferrari. Policy framings for access control. In *Workshop on Issues in the Theory of Security (WITS'05)*, 2005.

Appendix 1.4.1
D. Hirschkoff, D. Pous, and D. Sangiorgi. An Efficient Abstract Machine for Safe Ambients. In *Proc. of COORDINATION '05*, number 2004–63, 2005. to appear.

Appendix 1.4.2
G. Barthe and L. Prensa-Nieto. Formally verifying information flow type systems for concurrent and thread systems. In M. Backes, D. Basin, and M. Waidner, editors, *Proceedings of FMSE'04*, pages 13–22. ACM Press, 2004.

Appendix 1.3.5
Massimo Bartoletti, Pierpaolo Degano, and Gianluigi Ferrari. Method inlining in presence of stack inspection. In *Workshop on Issues in the Theory of Security (WITS'04)*, 2004.