INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team Phoenix

Programming Language Technology For Communication Services

Futurs

THEME COM

Activity Report

2005

Table of contents

1. Team

The Phoenix group is located in Bordeaux. Phoenix is a joint research group with LaBRI (Laboratoire Bordelais de Recherche en Informatique) – the computer science department at the University of Bordeaux I – CNRS (Centre National de la Recherche Scientifique) – a French national scientific research center – and ENSEIRB (Ecole Nationale Supérieure en Electronique, Informatique et Radiocommunications de Bordeaux) – an electronics, computer science, and telecommunications engineering school at Bordeaux. The group is physically located at ENSEIRB.

Team Leader

Charles Consel [Professor, ENSEIRB]

ENSEIRB personnel

Laurent Réveillère [Associate Professor, ENSEIRB]

External collaborator

Julia Lawall [Associate Professor at the University of Copenhagen]

Ph.D. students

Laurent Burgy [From October 1, 2004, regional scholarship]
Fabien Latry [From October 1, 2004, Inria scholarship]
Nicolas Palix [From October 1, 2004, Inria scholarship]
Sapan Bhatia [From January 1, 2003, Inria and regional scholarship]
Mathieu Minard [Thomson Multimedia (industrial Ph.D. student)]
Wilfried Jouve [From October 3, 2005, Phoenix scholarship]
Julien Lancia [From November 14, 2005, Thales (industrial Ph.D. student)]

2. Overall Objectives

2.1. Context

Keywords: *Operating systems*, *client-server model*, *communication services*, *compilation*, *domain analysis and engineering*, *language design*, *networking*, *program analysis and transformation*, *specialization*, *telephony*.

The frantic nature of technological advances in the area of multimedia communications, compounded with the effective convergence between telecommunication and computer networks, is opening up a host of new functionalities, placing service creation as a fundamental vehicle to bring these changes to end-users.

This situation has three main consequences: (1) service creation is increasingly becoming a *software intensive area*; (2) because communication services are often heavily relied on, intensive service creation must preserve *robustness*; (3) the growing multimedia nature of communication services imposes *high-performance requirements* on services and underlying layers.

2.2. Overview

Keywords: *Operating systems*, *client-server model*, *communication services*, *compilation*, *domain analysis and engineering*, *language design*, *networking*, *program analysis and transformation*, *specialization*, *telephony*.

The phoenix group aims to develop principles, techniques and tools for the development of *communication services*. To address the requirements of this domain, the scope of our research comprises the key elements underlying communication services: the infrastructure that enables communication to be set up (*e.g.*, signalling platform, transport protocols, and session description); the software architecture underlying services (*e.g.*, the client-server model, programming interfaces, and the notion of service logic); and, communication terminals (*e.g.*, terminal features and embedded systems).

Our approach covers three key aspects of the area of communication services: (1) definition of new Domain-Specific Languages (DSLs), using programming language technology to enable the specification of robust services; (2) study of the layers underlying communication services to improve flexibility and performance; (3) application to concrete areas to validate our approach.

3. Scientific Foundations

3.1. Introduction

Our proposed project builds upon results that have been obtained by the Compose research group whose aim was to study new approaches to developing adaptable software components in the domain of systems and networking. In this section, we review the accomplishments of Compose, only considering the ones achieved by the current project members, to demonstrate our expertise in the key areas underlying our project, namely

- Programming language technology: language design and implementation, domain-specific languages, program analysis and program transformation.
- Operating Systems and Networking: design, implementation and optimization.
- Software engineering: software architecture, methodologies, techniques and tools.

By combining expertise in these areas, the research work of the Compose group contributed to demonstrating the usefulness of adaptation methodologies, such as domain-specific languages, and the effectiveness of adaptation tools, such as program specializers. Our work aimed to show how adaptation methodologies and tools can be integrated into the development process of real-size software components. This contribution relied on advances in methodologies to develop adaptable programs, and techniques and tools to adapt these programs to specific usage contexts.

3.2. Adaptation Methodologies

Although industry has long recognized the need to develop adaptable programs, methodologies to develop them are still at the research stage. We have presented preliminary results in this area with a detailed study of the applicability of program specialization to various software architectures [27]. Our latest contributions in this area span from a revolutionary approach based on the definition of programming languages, dedicated to a specific problem family, to a direct exploitation of specialization opportunities generated by a conventional programming methodology.

3.2.1. Domain-Specific languages

DSLs represent a promising approach to modeling a problem family. Yet, this approach currently suffers from the lack of methodology to design and implement DSLs. To address this basic need, we have introduced the Sprint methodology for DSL development [20]. This methodology bridges the gap between semantics-based approaches to developing general-purpose languages and software engineering. Sprint is a complete software development process starting from the identification of the need for a DSL to its efficient implementation. It uses the denotational framework to formalize the basic components of a DSL. The semantic definition is structured so as to stage design decisions and to smoothly integrate implementation concerns.

3.2.2. Declaring adaptation

A less drastic strategy to developing efficient adaptable programs consists of making specific issues of adaptation explicit via a declarative approach. To do so, we enrich Java classes with declarations, named *adaptation classes*, aimed to express adaptive behaviors [16]. As such, this approach allows the programmer to separate the concerns between the basic features of the application and its adaptation aspects. A dedicated compiler automatically generates Java code that implements the adaptive features.

3.2.3. Declaring specialization

When developing components, programmers often hesitate to make them highly generic and configurable. Indeed, genericity and configurability systematically introduce overheads in the resulting component. However, the causes of these overheads are usually well-known by the programmers and their removal could often be automated, if only they could be declared to guide an optimizing tool. The Compose group has worked towards solving this problem.

We introduced a declaration language which enables a component developer to express the configurability of a component. The declarations consist of a collection of specialization scenarios that precisely identify what program constructs are of interest for specialization. The scenarios of a component do not clutter the component code; they are defined aside in a *specialization module* [22], [23], [21], [24].

This work was done in the context of C and declarations were intended to drive our C specializer.

3.2.4. Specializing design patterns

A natural approach to systematically applying program specialization is to exploit opportunities offered by a programming methodology. We have studied a development methodology for object-oriented languages, called design patterns. Design patterns encapsulate knowledge about the design and implementation of highly adaptable software. However, adaptability is obtained at the expense of overheads introduced in the finished program. These overheads can be identified for each design pattern. Our work consisted in using knowledge derived from design patterns to eliminate these overheads in a systematic way. To do so, we analyzed the specialization opportunities provided by specific uses of design patterns, and determined how to eliminate these overheads using program specialization. These opportunities were documented in declarations, called specialization patterns, and were associated with specific design patterns [34]. The specialization of a program composed of design patterns was then driven by the corresponding declarations. This work was presented in the context of Java and uses our Java specializer [33].

3.2.5. Specializing software architectures

The source of inefficiency in software architectures can be identified in the data and control integration of components, because flexibility is present not only at the design level but also in the implementation. We proposed the use of program specialization in software engineering as a systematic way to improve performance and, in some cases, to reduce program size. We studied several representative, flexible mechanisms found in software architectures: selective broadcast, pattern matching, interpreters, layers and generic libraries. We showed how program specialization can systematically be applied to optimize those mechanism [26], [27].

3.3. Adaptation in Systems Software

3.3.1. DSLs in Operating Systems

Integrating our adaptation methodologies and tools into the development process of real-size software systems was achieved by proposing a new development process. Specifically, we proposed a new approach to designing and structuring operating systems (OSes) [29]. This approach was based on DSLs and enables rapid development of robust OSes. Such approach is critically needed in application domain, like appliances, where new products appear at a rapid pace and needs are unpredictable.

3.3.2. Devil - a DSL for device drivers

Our approach to developing systems software applied to the domain of device drivers. Indeed, peripheral devices come out at a frantic pace, and the development of drivers is very intricate and error prone. The Compose group developed a DSL, named Devil (DEvice Interface Language), to solve these problems; it was dedicated to the basic communication with a device. Devil allowed the programmer to easily map device documentation into a formal device description that can be verified and compiled into executable code.

From a software engineering viewpoint, Devil captures domain expertise and systematizes re-use because it offers suitable built-in abstractions [31]. A Devil description formally specifies the access mechanisms, the type and layout of data, as well as behavioral properties involved in operating the device. Once compiled, a Devil description implements an interface to an idealized device and abstracts the hardware intricacies.

From an operating systems viewpoint, Devil can be seen as an *interface definition language* for hardware functionalities. To validate the approach, Devil was put to practice [30]: its expressiveness was demonstrated by the wide variety of devices that have been specified in Devil. No loss in performance was found for the compiled Devil description compared to an equivalent C code.

From a dependable system viewpoint, Devil improves safety by enabling descriptions to be statically checked for consistency and generating stubs including additional run-time checks [32]. Mutation analysis were used to evaluate the improvement in driver robustness offered by Devil. Based on our experiments, Devil specifications were found up to 6 times less prone to errors than writing C code.

Devil was the continuation of a study of graphic display adaptors for a X11 server. We developed a DSL, called GAL (Graphics Adaptor Language), aimed to specify device drivers in this context [38]. Although covering a very restricted domain, this language was a very successful proof of concept.

3.3.3. Plan-P - a DSL for programmable routers

Besides device drivers, the Compose group also explored the area of networking in the context of DSLs. More specifically, we developed a language, named Plan-P, that enables the network to be programmable and thus to offer extensibility [37]. As such, Plan-P enables protocols to be defined for specific applications. Plan-P extends a language, named Plan, developed by the University of Pennsylvania and devoted to network diagnostics. Plan-P enables routers to be programmed in a safe and secure way without any loss in bandwidth. To achieve safety and security, the language is restricted, and programs are downloaded into the routers as DSL source code to enable thorough verifications. For efficiency, a light Just-In-Time compiler is generated from the Plan-P interpreter via program specialization. This compiler is installed on routers to compile uploaded Plan-P source code.

3.4. Adaptation Tools and Techniques

To further the applicability of our approach, we have strengthened and extended adaptation tools and techniques. We have produced a detailed description of the key program analysis for imperative specialization, namely binding-time analysis [19]. This analysis is at the heart of our program specializer for C, named Tempo [19]. We have examined the importance of the accuracy of these analyses to successfully specialize existing programs. This study was conducted in the context of systems software [28].

Tempo is the only specializer which enables programs to be specialized both at compile time and run time. Yet, specialization is always performed in one stage. As a consequence, this process cannot be factorized even if specialization values become available at multiple stages. We present a realistic and flexible approach to achieving efficient incremental run-time specialization [25]. Rather than developing new techniques, our strategy for incremental run-time specialization reuses existing technology by iterating a specialization process. Our approach has been implemented in Tempo.

While program specialization encodes the result of early computations into a new program, *data specialization* encodes the result of early computations into data structures. Although aiming at the same goal, namely processing early computations, these two forms of specialization have always been studied separately. The Compose group has proposed an extension of Tempo to perform both program and data specialization [17]. We showed how these two strategies can be integrated in a single specializer. Most notably, having both strategies enabled us to assess their benefits, limitations and their combination on a variety of programs.

Interpreters and run-time compilers are increasingly used to cope with heterogeneous architectures, evolving programming languages, and dynamically-loaded code. Although solving the same problem, these two strategies are very different. Interpreters are simple to implement but yield poor performance. Run-time compilation yields better performance, but is costly to implement. One approach to reconciling these two strategies is to develop interpreters for simplicity but to use specialization to achieve efficiency. Additionally, a specializer like Tempo can remove the interpretation overhead at compile time as well as at run time. We have conducted experiments to assess the benefits of applying specialization to interpreters [36]. These experiments have involved bytecode and structured-language interpreters. Our experimental data showed that specialization of structured-language interpreters can yield performance comparable to that of the compiled code of an optimizing compiler.

Besides targeting C, we developed the first program specializer for an object-oriented language. This specializer, named JSpec, processes Java programs [33]. JSpec is constructed from existing tools. Java programs are translated into C using our Java compiler, named Harissa. Then, the resulting C programs are specialized using Tempo. The specialized C program is executed in the Harissa environment. JSpec has been used for various applications and has shown to produce significant speedups [35].

4. Application Domains

4.1. Telephony Services

Keywords: *SIP*, *adaptation*, *multimedia*, *telecommunications*.

IP telephony materializes the convergence between telecommunications and computer networks. This convergence is dramatically changing the face of the telecommunications domain moving from proprietary, closed platforms to distributed systems based on network protocols. In particular, a telephony platform is based on a client-server model and consists of a *signalling server* that implements a particular signalling protocol (*e.g.*, the Session Initiation Protocol [15]). A signalling server is able to perform telephony-related operations that include resources accessible from the computer network, such as Web resources, databases...This evolution brings a host of new functionalities to the domain of telecommunications. Such a wide spectrum of functionalities enables Telephony to be customized with respect to preferences, trends and expectations of ever demanding users. These customizations critically rely on a proliferation of telephony services. In fact, introducing new telephony services is facilitated by the open nature of signalling servers, as shown by all kinds of servers in distributed systems. However, in the context of telecommunications, such evolution should lead service programming to be done by non-expert programmers, as opposed to developers certified by telephony manufacturers. To make this evolution worse, the existing techniques to program server extensions (*e.g.*, Common Gateway Interface [14]) are rather low level, involves crosscutting expertises (*e.g.*, networking, distributed systems, and operating systems) and requires tedious session management. These shortcomings make the programming of telephony services an error-prone process, jeopardizing the robustness of a platform.

We are developing a DSL, named SPL (*Session Processing Language*), aimed to ease the development of telephony services without sacrificing robustness.

4.2. Multimedia Streaming Services

Keywords: *adaptation*, *multimedia*, *streaming*, *telecommunications*.

Mobility and wireless networks pose a major challenge to media delivery: how does one mass-deliver media while at the same time personalizating it to account for diverse needs such as multiple heterogeneous rendering terminals, user requirements, network bandwidth, *etc.*? Such personalization involves transcoding and transforming multimedia resources along the image chain.

To do so, various treatments, commonly supported by hardware, are gradually being shifted to software, to face unpredictable needs. On the one hand, this shift helps to keep pace with the rapidly evolving domain of media delivery. On the other hand, it imposes very high-performance requirements for treatments that were earlier hardware supported. As a consequence, developing a streaming application often involves low-level programming, critical memory management, and finely tuned scheduling of processing steps.

To address these problems, we have designed and implemented a DSL, named *Spidle*, for specifying streaming applications [18]. Our approach consists in

- Identifying (and possibly modifying) a protocol (*e.g.*, RTSP) for multimedia streaming.

- Making a streaming server, based on the previously identified protocol, programmable using Spidle. This work will permit streaming adaptations to the client needs and preferences.
- Defining realistic adaptation scenarios to validate our approach. This work may lead us to extend Spidle to cope with the target scenarios.
- Assessing our approach by conducting a thorough experimental study.

5. Software

5.1. Tempo - A Partial Evaluator for C

Keywords: *C language*, *partial evaluation*, *run-time specialization*.

Participants: Charles Consel [correspondent], Julia Lawall.

Tempo is a partial evaluator for C programs. It is an off-line specializer; it is divided into two phases: analysis and specialization.

The input to the analysis phase consists of a program and a description of which inputs will be known during specialization and which will be unknown. Based on this knowledge, dependency analyses propagate information about known and unknown values throughout the code and produce an annotated program, indicating how each program construct should be transformed during specialization. Because C is an imperative language including pointers, the analysis phase performs alias and side-effect analyses in addition to binding-time analyses. The accuracy of these analyses is targeted towards keeping track of known values across procedures, data structures, and pointers. Following the analysis phase, the specialization phase generates a specialized program based on the annotated program and the values of the known inputs. Tempo can specialize programs at compile time (i.e., source-to-source transformation) as well as run time (i.e., run-time binary code generation).

The Tempo specializer has been applied in various domains such as operating systems and networking, computer graphics, scientific computation, software engineering and domain specific languages. It has been made publicly available since April 1998. Its documentation is available on line, as well as tutorial slides.

5.2. SPL - A Domain-Specific Language for Robust Session Processing Services

Keywords: *SIP*, *adaptation*, *services*, *sessions*, *telephony*.

Participants: Charles Consel, Laurent Réveillère [correspondent], Laurent Burgy, Fabien Latry, Nicolas Palix.

SPL is a high-level domain-specific language for specifying robust Internet telephony services.

SPL reconciles programmability and reliability of telephony services, and offers high-level constructs that abstract over intricacies of the underlying protocols and software layers. SPL makes it possible for owners of telephony platforms to deploy third-party services without compromising safety and security. This openness is essential to have a community of service developers that addresses such a wide spectrum of new functionalities. The SPL compiler is nearing completion.

5.3. Stingy - A Domain-Specific Compiler for High-performance Network Servers

Keywords: *Cache Optimizations*, *Domain-specific optimizations*, *Event-driven Programs*.

Participants: Sapan Bhatia [correspondent], Charles Consel, Julia Lawall.

Event-driven programming has emerged as a standard to implement high-performance servers due to its flexibility and low OS overhead. Still, memory access remains a bottleneck. Generic optimization techniques yield only small improvements in the memory access behavior of event-driven servers, as such techniques do not exploit their specific structure and behavior.

The Stingy compiler implementes an optimization framework dedicated to event-driven servers, based on a strategy to eliminate data-cache misses. Our approach exploits the flexible scheduling and deterministic execution of event-driven servers. It is based on a novel memory manager combined with a tailored scheduling strategy to restrict the working data set of the program to a memory region mapped directly into the data cache.

In practice, the Stingy compiler accepts as input an event-driven server written in C and annotated to expose a specific memory management and scheduling interface. As output, it generates C code for an optimized version of the server. The Stingy compiler has been tested on the following servers: The TUX, thttpd, Flash, boa, mathopd. It has also been applied to the Cactus QoS framework and the Squid proxy server. The highest speedup observed under heavy loads is on the TUX server (in the range of 40%). For the remaining servers, gains are in the region of 10-15%.

6. New Results

6.1. A Generative Programming Approach To Developing DSL Compilers

Participants: Charles Consel, Fabien Latry, Laurent Réveillère.

Domain-Specific Languages (DSLs) represent a proven approach to raising the abstraction level of programming. They offer highlevel constructs and notations dedicated to a domain, structuring program design, easing program writing, masking the intricacies of underlying software layers, and guaranteeing critical properties.

On the one hand, DSLs facilitate a straightforward mapping between a conceptual model and a solution expressed in a specific programming language. On the other hand, DSLs complicate the compilation process because of the gap in the abstraction level between the source and target language. The nature of DSLs make their compilation very different from the compilation of common General-Purpose Languages (GPLs). In fact, a DSL compiler generally produces code written in a GPL; low-level compilation is left to the compiler of the target GPL. In essence, a DSL compiler defines some mapping of the high-level information and features of a DSL into the target GPL and underlying layers (e.g., middleware, protocols, objects, . . .).

This paper presents a methodology to develop DSL compilers, centered around the use of generative programming tools. Our approach enables the development of a DSL compiler to be structured on facets that represent dimensions of compilation. Each facet can then be implemented in a modular way, using aspects, annotations and specialization. Because these tools are high level, they match the needs of a DSL, facilitating the development of the DSL compiler, and making it modular and retargetable.

We illustrate our approach with a DSL for telephony services. The structure of the DSL compiler is presented, as well as practical uses of generative tools for some compilation facets. For more information, see: [11].

6.2. Clearwater: Extensible, Flexible, Modular Code Generation

Participant: Charles Consel.

Distributed applications typically interact with a number of heterogeneous and autonomous components that evolve independently. Methodical development of such applications can benefit from approaches based on domain-specific languages (DSLs). However, the evolution and customization of heterogeneous components introduces significant challenges to accommodating the syntax and semantics of a DSL in addition to the heterogeneous platforms on which they must run. In this paper, we address the challenge of implementing code generators for two such DSLs that are flexible (resilient to changes in generators or input formats), extensible (able to support multiple output targets and multiple input variants), and modular (generated code can be rewritten). Our approach, Clearwater, leverages XML and XSLT standards: XML supports extensibility and mutability for inprogress specification formats, and XSLT provides flexibility and extensibility for multiple target languages. Modularity arises from using XML meta-tags in the code generator itself, which supports controlled addition, subtraction, or replacement to the generated code via XML-weaving. We discuss the use of our approach and show its advantages in two non-trivial code generators: the Infopipe Stub Generator

(ISG) to support distributed flow applications, and the Automated Composable Code Translator to support automated distributed application deployment. As an example, the ISG accepts as input an XML description and generates output for C, C++, or Java using a number of communications platforms such as sockets and publish-subscribe. For more information, see: [12].

7. Contracts and Grants with Industry

7.1. ACI Security COrSS

Participants: Laurent Burgy, Charles Consel, Fabien Latry, Nicolas Palix, Laurent Réveillère.

This project, entitled "Composition and refinement of Secure Systems", is a collaboration between groups from the systems and formal methods community.

The goal is to study methods and tools for the development of secure and safe systems services, with a special emphasis on specification. Our contribution focuses on the development of robust telephony services using DSLs. The collaboration with researchers in formal methods aims to use tools (*e.g.*, theorem provers) to formalize and check properties specific to the DSL and the domain of telephony.

7.2. Ambient Intelligence For The Networked Home Environment (IP6 Amigo)

Participants: Laurent Burgy, Charles Consel, Fabien Latry, Nicolas Palix, Laurent Réveillère.

The Amigo project will focus on the usability of a networked home system by developing open, standardized, interoperable middleware. The developed middleware will guarantee automatic dynamic configuration of the devices and services within this home system by addressing autonomy and composability aspects. The second focus of the Amigo project will be on improving the end-user attractiveness of a networked home system by developing interoperable intelligent user services and application prototypes. The Amigo project will further support interoperability between equipment and services within the networked home environment by using standard technology when possible and by making the basic middleware (components and infrastructure) and intelligent user services available as open source software together with architectural rules for everyone to use.

Our work in the Amigo project is based on our DSL paradigm for protocol-based service families, presented in Section. We aim to develop DSLs for service creation. Indeed, the area of networked home systems, targetted by Amigo, relies on protocols for families of services (*e.g.*, SIP, Session Announcement protocol, and Delivery Multimedia Framework). Furthermore, the underlying software architecture in this area relies on a client-server model. This situation should give us an opportunity to further illustrate our approach to making servers DSL-programmable.

7.3. A Platform for the Development of Robust Multimedia Applications in Mobile Terminals – Région Aquitaine

Participants: Laurent Burgy, Charles Consel, Fabien Latry, Nicolas Palix, Laurent Réveillère.

The world of mobile communication terminals (MCT), such as telephones, handheld computers and PCs, has witnessed dazzling advances for the last few years. Most of the effort has been focused on improving the hardware capabilities of the devices rather than the applications offering services to the users. However, as wireless technologies (GPRS, UMTS, BlueTooth, WiFI) are increasingly becoming available on these devices, it is critical to offer robust applications that make the best use of the available resources.

This project aims to develop a platform for the development of robust multimedia services on MCT.

7.4. Service Oriented Architecture for Embedded Systems – Industrial Fellowship (CIFRE / Thales)

Participants: Charles Consel, Julien Lancia.

The goal of this project is to design and develop a SOA architecture for embedded systems. More especially, it takes into account 3 levels of adaptation: (1) the component level (contracts on resources, performances...), (2) the coupling of components level (dependence, security...), and (3) the software architecture level (resource management, robustness...). A contract-based component approach will be considered to describe nonfunctional properties, to define mechanisms for coupling of components, and to define control mechanisms when executing elements of a component. This study will be illustrated by a concrete application. The research work should be a step toward solving key problems such as composition of services, security, component adaptation and performance.

7.5. Capability-based DSLs – Région Aquitaine Fellowship

Participants: Laurent Burgy, Charles Consel, Laurent Réveillère.

To answer the fundamental need for innovations in terms of services, existing infrastructures have become increasingly open to external developers. Yet, this openness is done at the expense of the robustness. The aim of this project is to integrate approaches dedicated to finely tuning access to ressources into programming languages. This study will introduce a unique DSL to program services whose interface to resources is configured with respect to the different roles of programmers and so their capabilities.

8. Other Grants and Activities

8.1. International Collaborations

We have been exchanging visits and publishing articles with the following collaborators.

- Julia Lawall, DIKU, University of Copenhagen (Denmark, Copenhagen).
  DSLs, specialization, program analysis.
- Calton Pu, Georgia Institute of Technology (USA, Atlanta).
  DSLs and specialization for operating systems.

8.2. Visits and Invited Researchers

The Phoenix group has been visited by:

- Julia L. Lawall (DIKU, University of Copenhagen, Denmark), from the 1st of January to the 30th of April;
- Georges Necula (University of Berkeley, USA).

9. Dissemination

9.1. Scientific Community Participation

Charles Consel has been involved in the following events as:

- Program committee member of the *International Conference on Compiler Construction* (CC 2005);
- Program committee member of *2ème Journée Francophone sur le Développement de Logiciels Par Aspects* (JFDLPA 2005);
- Member of the SPECIF best thesis award;
- Committee member for Francisco Alberti's thesis, May 2005, universite Paris 7;
- Promotion Committee (University of Utah, University of Singapour, Oregon Graduate Institute, GeorgiaTech);
- Member of the IFIP working group on *Program Generation*.

Laurent Réveillère has been involved in the following events as:

- Program committe member of the *Fourth French Conference on Operating Systems* (CFSE 2005);
- Member of the IFIP working group on *Program Generation*;
- Secretary of the French chapter of ACM SIGOPS.

9.2. Teaching

Charles Consel and Laurent Réveillère have been teaching Master’s level courses on:

- Domain-Specific Languages and Program Analysis;
- Telephony over IP (related protocols, the SIP protocol, existing programming interfaces). Students are also offered practical labs on various industrial-strength telephony platforms.

Charles Consel and Laurent Réveillère are also teaching other courses on Operating Systems, Web programming and Compilation.

9.3. Presentations and Invitations

Charles Consel gave a number of invited presentations.

- Invited lecturer at the university of Freiburg, Germany;
- Invited speaker at Georgia Tech (Atlanta, 3 weeks);
- Invited speaker at Fundamental Research in Software Engineering at European Commission.

Charles Consel and Laurent Réveillère were lectures at the *École des jeunes Chercheurs en Programmation* (EJCP 2005).

9.4. Phoenix in the News

The work of the Phoenix group has been reported in the news.

- L. Burgy, C. Consel, F. Latry, L. Réveillère, and N. Palix. Telephony over IP: Experience and Challenges. *ERCIM News*, 63:53, October 2005 [13];
- Publication of a press article in *Sud-Ouest* "Révolution au bout du fil" by Willy Dallay (Friday, September 23, 2005);
- TV report in *France 3 Aquitaine news* (Thursday, September 22, 2005);
- Publication of a press article in *Le monde Informatique* number 1086 - page 28 "Un langage spécifique orienté communication" by J.-L. R. (Friday, October 14, 2005).

10. Bibliography

Major publications by the team in recent years

[1] C. CONSEL. *Domain-Specific Program Generation; International Seminar, Dagstuhl Castle*, C. LENGAUER, D. BATORY, C. CONSEL, M. ODERSKY (editors). , Lecture Notes in Computer Science, State-of-the-Art Survey, chap. From A Program Family To A Domain-Specific Language, n^{o} 3016, Springer-Verlag, 2004, p. 19-29, http://phoenix.labri.fr/publications/papers/dagstuhl-consel.pdf.

[2] C. CONSEL, J. LAWALL, A.-F. LE MEUR. *A Tour of Tempo: A Program Specializer for the C Language*, in "Science of Computer Programming", 2004, http://phoenix.labri.fr/publications/papers/tour-tempo.ps.gz.

[3] C. CONSEL, R. MARLET. *Architecturing software using a methodology for language development*, in "Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming, Pisa, Italy", C. PALAMIDESSI, H. GLASER, K. MEINKE (editors). , Lecture Notes in Computer Science, vol. 1490, September 1998, p. 170–194, http://phoenix.labri.fr/publications/papers/plilp98.ps.gz.

[4] C. CONSEL, L. RÉVEILLÈRE. *A Programmable Client-Server Model: Robust Extensibility via DSLs*, in "Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003), Montréal, Canada", IEEE Computer Society Press, November 2003, p. 70–79, http://phoenix.labri.fr/publications/papers/Consel-Reveillere_ase03.pdf.

[5] C. CONSEL, L. RÉVEILLÈRE. *Domain-Specific Program Generation; International Seminar, Dagstuhl Castle*, C. LENGAUER, D. BATORY, C. CONSEL, M. ODERSKY (editors). , Lecture Notes in Computer Science, State-of-the-Art Survey, chap. A DSL Paradigm for Domains of Services: A Study of Communication Services, n^{o} 3016, Springer-Verlag, 2004, p. 165 – 179, http://phoenix.labri.fr/publications/papers/dagstuhl04_consel_reveillere.pdf.

[6] A.-F. LE MEUR, J. LAWALL, C. CONSEL. *Specialization Scenarios: A Pragmatic Approach to Declaring Program Specialization*, in "Higher-Order and Symbolic Computation", vol. 17, n^{o} 1, 2004, p. 47–92, http://phoenix.labri.fr/publications/papers/spec-scenarios-hosc2003.ps.gz.

[7] D. MCNAMEE, J. WALPOLE, C. PU, C. COWAN, C. KRASIC, A. GOEL, P. WAGLE, C. CONSEL, G. MULLER, R. MARLET. *Specialization tools and techniques for systematic optimization of system software*, in "ACM Transactions on Computer Systems", vol. 19, n^{o} 2, May 2001, p. 217–251, http://phoenix.labri.fr/publications/papers/tocs01-namee.pdf.

[8] F. MÉRILLON, L. RÉVEILLÈRE, C. CONSEL, R. MARLET, G. MULLER. *Devil: An IDL for Hardware Programming*, in "Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, San Diego, California", October 2000, p. 17–30, http://phoenix.labri.fr/publications/papers/osdi00-merillon.pdf.

[9] L. RÉVEILLÈRE, G. MULLER. *Improving Driver Robustness: an Evaluation of the Devil Approach*, in "The International Conference on Dependable Systems and Networks, Göteborg, Sweden", IEEE Computer Society, July 2001, p. 131–140, http://phoenix.labri.fr/publications/papers/Reveillere-Muller_dsn2001.pdf.

[10] S. THIBAULT, C. CONSEL, G. MULLER. *Safe and Efficient Active Network Programming*, in "17th IEEE Symposium on Reliable Distributed Systems, West Lafayette, IN", October 1998, p. 135–143,

http://phoenix.labri.fr/publications/papers/srds98-thibault.ps.gz.

Publications in Conferences and Workshops

[11] C. CONSEL, F. LATRY, L. RÉVEILLÈRE, P. COINTE. *A Generative Programming Approach to Developing DSL Compilers*, in "Fourth International Conference on Generative Programming and Component Engineering (GPCE), Tallinn, Estonia", R. GLUCK, M. LOWRY (editors). , Lecture Notes in Computer Science, vol. 3676, Springer-Verlag, September 2005, p. 29–46.

[12] G. SWINT, C. PU, G. JUNG, W. YAN, Y. KOH, Q. WU, C. CONSEL, A. SAHAI, K. MORIYAMA. *Clearwater: Extensible, Flexible, Modular Code Generation*, in "Proceedings of the 20th IEEE International Conference on Automated Software Engineering (ASE 2005), Long Beach, CA", IEEE Computer Society Press, October 2005.

Miscellaneous

[13] L. BURGY, C. CONSEL, F. LATRY, L. RÉVEILLÈRE, N. PALIX. *Telephony over IP: Experience and Challenges*, vol. 63, October 2005, http://www.ercim.org/publication/Ercim_News/enw63/.

Bibliography in notes

[14] *CGI: The Common Gateway Interface*, http://cgi-spec.golux.com/ncsa.

[15] *Session Initiation Protocol (SIP)*, Request for Comments 2543, March 2001.

[16] P. BOINOT, R. MARLET, J. NOYÉ, G. MULLER, C. CONSEL. *A Declarative Approach for Designing and Developing Adaptive Components*, in "Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE 2000), Grenoble, France", IEEE Computer Society Press, September 2000, http://phoenix.labri.fr/publications/papers/ase00-adaptClass.ps.gz.

[17] S. CHIROKOFF, C. CONSEL, R. MARLET. *Combining Program and Data Specialization*, in "Higher-Order and Symbolic Computation", vol. 12, n^o 4, December 1999, p. 309–335.

[18] C. CONSEL, F. LATRY, L. RÉVEILLÈRE, P. COINTE. *A Generative Programming Approach to Developing DSL Compilers*, in "Fourth International Conference on Generative Programming and Component Engineering (GPCE), Tallinn, Estonia", R. GLUCK, M. LOWRY (editors). , Lecture Notes in Computer Science, vol. 3676, Springer-Verlag, September 2005, p. 29–46.

[19] C. CONSEL, J. LAWALL, A.-F. LE MEUR. *A Tour of Tempo: A Program Specializer for the C Language*, in "Science of Computer Programming", 2004.

[20] C. CONSEL, R. MARLET. *Architecturing software using a methodology for language development*, in "Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming, Pisa, Italy", C. PALAMIDESSI, H. GLASER, K. MEINKE (editors). , Lecture Notes in Computer Science, vol. 1490, September 1998, p. 170–194.

[21] A.-F. LE MEUR, C. CONSEL, B. ESCRIG. *An Environment for Building Customizable Software Components*, in "IFIP/ACM Conference on Component Deployment, Berlin, Germany", June 2002, p. 1–14.

[22] A.-F. LE MEUR, C. CONSEL. *Generic Software Component Configuration Via Partial Evaluation*, in "SPLC'2000 Workshop – Product Line Architecture, Denver, Colorado", August 2000.

[23] A.-F. LE MEUR, J. LAWALL, C. CONSEL. *Towards Bridging the Gap Between Programming Languages and Partial Evaluation*, in "ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation, Portland, OR, USA", ACM Press, January 2002, p. 9–18, http://phoenix.labri.fr/publications/papers/lemeur-pepm02.pdf.

[24] A.-F. LE MEUR, J. LAWALL, C. CONSEL. *Specialization Scenarios: A Pragmatic Approach to Declaring Program Specialization*, in "Higher-Order and Symbolic Computation", vol. 17, n^o 1, 2004, p. 47–92.

[25] R. MARLET, C. CONSEL, P. BOINOT. *Efficient Incremental Run-Time Specialization for Free*, in "Proceedings of the ACM SIGPLAN'99 Conference on Programming Language Design and Implementation (PLDI'99), Atlanta, GA, USA", May 1999, p. 281–292.

[26] R. MARLET, S. THIBAULT, C. CONSEL. *Mapping Software Architectures to Efficient Implementations via Partial Evaluation*, in "Conference on Automated Software Engineering, Lake Tahoe, NV, USA", IEEE Computer Society, November 1997, p. 183–192.

[27] R. MARLET, S. THIBAULT, C. CONSEL. *Efficient Implementations of Software Architectures via Partial Evaluation*, in "Journal of Automated Software Engineering", vol. 6, n^o 4, October 1999, p. 411–440.

[28] D. MCNAMEE, J. WALPOLE, C. PU, C. COWAN, C. KRASIC, A. GOEL, P. WAGLE, C. CONSEL, G. MULLER, R. MARLET. *Specialization tools and techniques for systematic optimization of system software*, in "ACM Transactions on Computer Systems", vol. 19, n^o 2, May 2001, p. 217–251.

[29] G. MULLER, C. CONSEL, R. MARLET, L. BARRETO, F. MÉRILLON, L. RÉVEILLÈRE. *Towards Robust OSes for Appliances: A New Approach Based on Domain-Specific Languages*, in "Proceedings of the ACM SIGOPS European Workshop 2000 (EW2000), Kolding, Denmark", September 2000.

[30] F. MÉRILLON, L. RÉVEILLÈRE, C. CONSEL, R. MARLET, G. MULLER. *Devil: An IDL for Hardware Programming*, in "4th Symposium on Operating Systems Design and Implementation (OSDI 2000), San Diego, California", October 2000, p. 17–30.

[31] L. RÉVEILLÈRE, F. MÉRILLON, C. CONSEL, R. MARLET, G. MULLER. *A DSL Approach to Improve Productivity and Safety in Device Drivers Development*, in "Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE 2000), Grenoble, France", IEEE Computer Society Press, September 2000, p. 101–109.

[32] L. RÉVEILLÈRE, G. MULLER. *Improving Driver Robustness: an Evaluation of the Devil Approach*, in "The International Conference on Dependable Systems and Networks, Göteborg, Sweden", IEEE Computer Society, July 2001, p. 131–140.

[33] U. SCHULTZ, J. LAWALL, C. CONSEL, G. MULLER. *Towards Automatic Specialization of Java Programs*, in

"Proceedings of the European Conference on Object-oriented Programming (ECOOP'99), Lisbon, Portugal", Lecture Notes in Computer Science, vol. 1628, June 1999, p. 367–390.

[34] U. SCHULTZ, J. LAWALL, C. CONSEL. *Specialization Patterns*, in "Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE 2000), Grenoble, France", IEEE Computer Society Press, September 2000, p. 197–208.

[35] U. SCHULTZ, J. LAWALL, C. CONSEL. *Automatic Program Specialization for Java*, in "ACM Transactions on Programming Languages and Systems", vol. 25, n^o 4, 2003, p. 452–499.

[36] S. THIBAULT, C. CONSEL, J. LAWALL, R. MARLET, G. MULLER. *Static and Dynamic Program Compilation by Interpreter Specialization*, in "Higher-Order and Symbolic Computation", vol. 13, n^o 3, September 2000, p. 161–178.

[37] S. THIBAULT, C. CONSEL, G. MULLER. *Safe and Efficient Active Network Programming*, in "17th IEEE Symposium on Reliable Distributed Systems, West Lafayette, IN", October 1998, p. 135–143.

[38] S. THIBAULT, R. MARLET, C. CONSEL. *Domain-Specific Languages: from Design to Implementation – Application to Video Device Drivers Generation*, in "IEEE Transactions on Software Engineering", vol. 25, n^o 3, May 1999, p. 363–377.