INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team phoenix

Programming Language Technology For Communication Services

Bordeaux - Sud-Ouest

Theme : Distributed Systems and Services

Table of contents

The Phoenix group is located at the INRIA Bordeaux-Sud Ouest center. Phoenix is an INRIA Project-Team joint with University of Bordeaux and CNRS (LaBRI, UMR 5800).

1. Team

Research Scientist

Emilie Balland [INRIA Research Associate (CR), from October 1, 2009]

Faculty Member

Charles Consel [Team Leader, Professor, ENSEIRB, HdR]

External Collaborator

Julia Lawall [Associate Professor, University of Copenhagen (DIKU)]

Technical Staff

Stéphanie Gatti [Research Assistant, from October 1, 2009, to January 31, 2009]
Quentin Enard [Expert Engineer, from November 1, 2009, to January 31, 2010]
Benjamin Bertran [Associate Engineer, from December 17, 2007]
Alexandre Blanquart [Associate Engineer, from October 1, 2007]
Ghislain Deffrasnes [Associate Engineer, from October 1, 2009]

PhD Student

Wilfried Jouve [INRIA scholarship, PhD defense in April, 2009]
Julien Mercadal [Ministerial scholarship, from October 2, 2006, University of Bordeaux 1]
Zoé Drey [Thales scholarship, from November 2, 2006 to October 31, 2009 / INRIA Research Assistant, from November 1, 2009]
Damien Cassou [Ministerial scholarship, from October 1, 2007, University of Bordeaux 1]
Julien Bruneau [Thales scholarship, from October 1, 2008]
Henner Jakob [INRIA scholarship, from May 1, 2008]
Hongyu Guan [Region scholarship, from February 9, 2009]
Pengfei Liu [INRIA scholarship, from October 1, 2009]

Post-Doctoral Fellow

Nicolas Loriant [INRIA scholarship, from October 1, 2009, to September 30, 2010]

Administrative Assistant

Sylvie Embolla [Group Assistant, from September 4, 2006]

2. Overall Objectives

2.1. Overall Objectives

The frantic pace 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) service creation must preserve robustness because communication services are heavily relied on; (3) the growing multimedia nature of communication services imposes high-performance requirements on services and underlying layers.

To explore this research area, the Phoenix research group develops principles, techniques and tools for the development of communication services:

- the specification of robust communication services based on innovative Domain-Specific Languages (DSLs),
- the study of the layers underlying communication services to improve flexibility and performance,
- the application to concrete areas such as pervasive computing or IP telephony to validate our approach.

2.2. Highlights of the year

The main highlight of this year is the DIASUITE software platform. This development environment dedicated to pervasive computing has been successfully applied to automate a 13,500 square meters building, hosting the ENSEIRB (an engineering school) and research groups. The presentation of this development by Blanquart et al. [18] was awarded as "Best demonstration" by the ACM International Conference on Pervasive Services. In the context of the HomeSIP project [20], DIASUITE was also successfully applied for developing a demonstration platform of home automation at France Telecom.

3. Scientific Foundations

3.1. Introduction

Our proposed project builds upon results previously 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 could 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 [33]. 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 [25]. 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 [22]. 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* [28], [29], [27], [30].

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 [42]. 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 [41].

3.2.5. Specializing software architectures

The sources 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 be applied systematically to optimize these mechanisms [32], [33].

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) [37]. This approach was based on DSLs and enables rapid development of robust OSes. Such an approach is critically needed in application domains, 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 [39]. 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 [38]: 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 [40]. 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 [45]. Although covering a very restricted domain, this language was a very successful proof of concept.

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 [24]. This analysis is at the heart of our program specializer for C, named Tempo [24]. 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 [35].

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 [31]. 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 [23]. 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 [44]. 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 [41]. 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 [43].

4. Application Domains

4.1. Introduction

After having explored DSLs in isolated domains in the past, we now generalize this experience to attack a larger domain, namely, communication services. Generalizing our work on telephony, we investigated the coordination of *networked entities*, whether or not operated by users. The two main application domains are the pervasive computing systems and the telephony services.

4.2. Pervasive Computing Systems

Pervasive computing systems are being deployed in a rapidly increasing number of areas, including building automation, assisted living, and supply chain management. Regardless of their target area, pervasive computing systems have a typical architectural pattern. They aggregate data from a variety of distributed sources, whether sensing devices or software components, analyze a context to make decisions, and carry out decisions by invoking a range of actuators. Because pervasive computing systems are standing at the crossroads of several domains (*e.g.,* distributed systems, multimedia, and embedded systems), they raise a number of challenges in software development:

- *Heterogeneity.* Pervasive computing systems are made of off-the-shelf entities, that is, hardware and software building blocks. These entities run on specific platforms, feature various interaction models, and provide non-standard interfaces. This heterogeneity tends to percolate in the application code, preventing its portability and reusability, and cluttering it with low-level details.
- *Lack of structuring.* Pervasive computing systems coordinate numerous, interrelated components. A lack of global structuring makes the development and evolution of such systems error-prone: component interactions may be invalid or missing.
- *Combination of technologies.* Pervasive computing systems involve a variety of technological issues, including device intricacies, complex APIs of distributed systems technologies and middleware-specific features. Coping with this range of issues results in code bloated with special cases to glue technologies together.
- Dynamicity. In a pervasive computing system, devices may either become available as they get deployed, or unavailable due to malfunction or network failure. Dealing with these issues explicitly in the implementation can quickly make the code cumbersome.
- Testing. Pervasive computing systems are complicated to test. Doing so requires equipments to be acquired, tested, configured and deployed. Furthermore, some scenarios cannot be tested because of the nature of the situations involved (*e.g.,* fire and smoke). As a result, the programmer must resort to writing specific code to achieve ad hoc testing.

4.3. Telephony Services

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 [21]). 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 evolutions 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 [19]) 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.

5. Software

5.1. DiaSuite: a Development Environment for Pervasive Computing Applications

Participants: Damien Cassou [correspondent], Charles Consel, Benjamin Bertran, Julien Bruneau, Julien Mercadal, Nicolas Loriant.

Despite much progress, developing a pervasive computing application remains a challenge because of a lack of conceptual frameworks and supporting tools. This challenge involves coping with heterogeneous devices, overcoming the intricacies of distributed systems technologies, working out an architecture for the application, encoding it in a program, writing specific code to test the application, and finally deploying it.

DIASUITE is a suite of tools covering the development life-cycle of a pervasive computing application:

- *Defining an application area.* First, an expert defines a catalog of entities, whether hardware or software, that are specific to a target area. These entities serve as building blocks to develop applications in this area. They are gathered in a taxonomy definition, written in the taxonomy layer of the DIASPEC language.
- *Architecturing an application.* Given a taxonomy, the architect can design and structure applications. To do so, the DIASPEC language provides an Architecture Description Language (ADL) layer [36]. This layer is dedicated to an architectural pattern commonly used in the pervasive computing domain [26]. Describing the architecture application allows to further model a pervasive computing system, making explicit its functional decomposition.
- *Implementing an application.* We leverage the taxonomy definition and the architecture description to provide dedicated support to both the entity and the application developers. This support takes the form of a Java programming framework, generated by the DIAGEN compiler. The generated programming framework precisely guides the developer with respect to the taxonomy definition and the architecture description. It consists of high-level operations to discover entities and interact with both entities and application components. In doing so, it abstracts away from the underlying distributed technologies, providing further separation of concerns.

- *Testing an application.* DIAGEN generates a simulation support to test pervasive computing applications before their actual deployment. An application is simulated in the DIASIM tool, without requiring any code modification. DIASIM provides an editor to define simulation scenarios and a 2D-renderer to monitor the simulated application. Furthermore, simulated and actual entities can be mixed. This hybrid simulation enables an application to migrate incrementally to an actual environment.
- *Deploying a system.* Finally, the system administrator deploys the pervasive computing system. To this end, a distributed systems technology is selected. We have developed a back-end that currently targets the following technologies: Web Services, RMI, CORBA and SIP. This targeting is transparent for the application code. The variety of these target technologies demonstrates that our development approach separates concerns into well-defined layers.

This development cycle is summarized in the Figure 1.

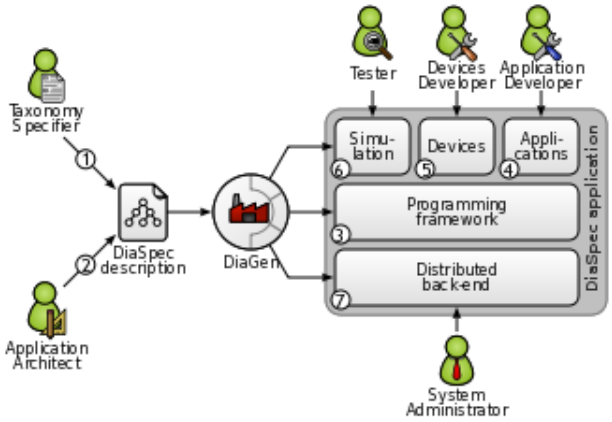

Figure 1. DIASUITE *Development Cycle*

See also the web page http://diasuite.inria.fr.

5.1.1. DiaSpec: a Domain-Specific Language for Networked Entities

The core of the DIASUITE development environment is the domain specific language called DIASPEC and its compiler DIAGEN:

- DIASPEC is composed of two layers:
  - The *Taxonomy Layer* allows the declaration of entities that are relevant to the target application area. An entity consists of sensing capabilities, producing data, and actuating capabilities, providing actions. Accordingly, an entity description declares a data source for each one of its sensing capabilities. As well, an actuating capability corresponds to a set of method declarations. An entity declaration also includes attributes, characterizing properties of entity instances. Entity declarations are organized hierarchically allowing entity classes to inherit attributes, sources and actions. A taxonomy allows separation of concerns in that the expert can focus on the concerns of cataloging area-specific entities. The entity developer is concerned about mapping a taxonomical description into an actual entity, and the application developer concentrates on the application logic.

  - The *Architecture Layer* is based on an architectural pattern commonly used in the pervasive computing domain [26]. It consists of context components fueled by sensing entities. These components process gathered data to make them amenable to the application needs. Context data are then passed to controller components that trigger actions on entities. Using an architecture description enables the key components of an application to be identified, allowing their implementation to evolve with the requirements (*e.g.,* varying light management implementations in a controller component to optimize energy consumption).

- DIAGEN is the DIASPEC compiler and runtime, performs both static and runtime verifications over DIASPEC declarations and produces a dedicated programming framework that guides and eases the implementation of components. The generated framework is independent of the underlying distributed technology. As of today, DIAGEN supports multiple targets: Local, RMI, SIP and a simulation target (the Web Services and the Corba targets being currently in development).

5.1.2. DiaSim: a Parametrized Simulator for Pervasive Computing Applications

Pervasive computing applications involve both software and integration concerns. This situation is problematic for testing pervasive computing applications because it requires acquiring, testing and interfacing a variety of software and hardware entities. This process can rapidly become costly and time-consuming when the target environment involves many entities.

To ease the testing of pervasive applications, we are developing a simulator for pervasive computing applications: DIASIM. To cope with widely heterogeneous entities, DIASIM is parameterized with respect to a DIASPEC specification describing a target pervasive computing environment. This description is used to generate with DIAGEN both a programming framework to develop the simulation logic and an emulation layer to execute applications. Furthermore, a simulation renderer is coupled to DIASIM to allow a simulated pervasive system to be visually monitored and debugged. The simulation renderer is illustrated in Figure 2.

5.2. Pantagruel: a Visual Domain-Specific Language for Ubiquitous Computing

Participants: Zoé Drey [correspondent], Julien Mercadal, Alexandre Blanquart, Charles Consel.

Pantagruel aims at easing the description of an orchestration logic between networked entities of a pervasive environment. First, the developer defines a taxonomy of entities that compose the environment, This step provides an abstraction of the entities capabilities and functionalities. Second, the developer defines the orchestration logic in terms of rules. To facilitate its programming, we provide a visual domain-specific language based on the sensor-controller-actuator paradigm. An example of a visual orchestration is given in Figure 3 where a shower automatically runs at the right temperature when someone enters the bathroom and closes the door.

Pantagruel brings a high-level layer intended to complement existing tools in the activity of safe orchestration logic description, allowing novice-programmers to prototype pervasive applications. The Pantragruel compiler generates code compliant with the DIASUITE toolset. Pantagruel is being completed by tools aimed at verifying safety properties like termination and reachability.

See also the web page http://pantagruel.bordeaux.inria.fr.

6. New Results

6.1. A Tool-Based Methodology for Developing Pervasive Computing Applications

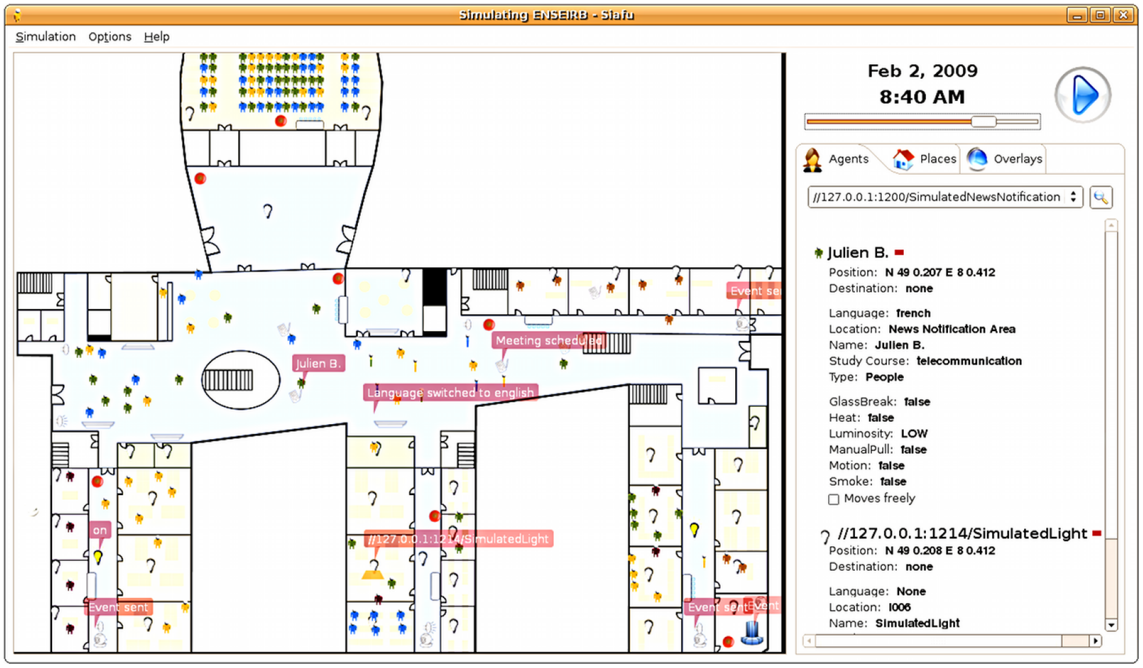

Figure 2. A screenshot of the DIASIM *simulator*

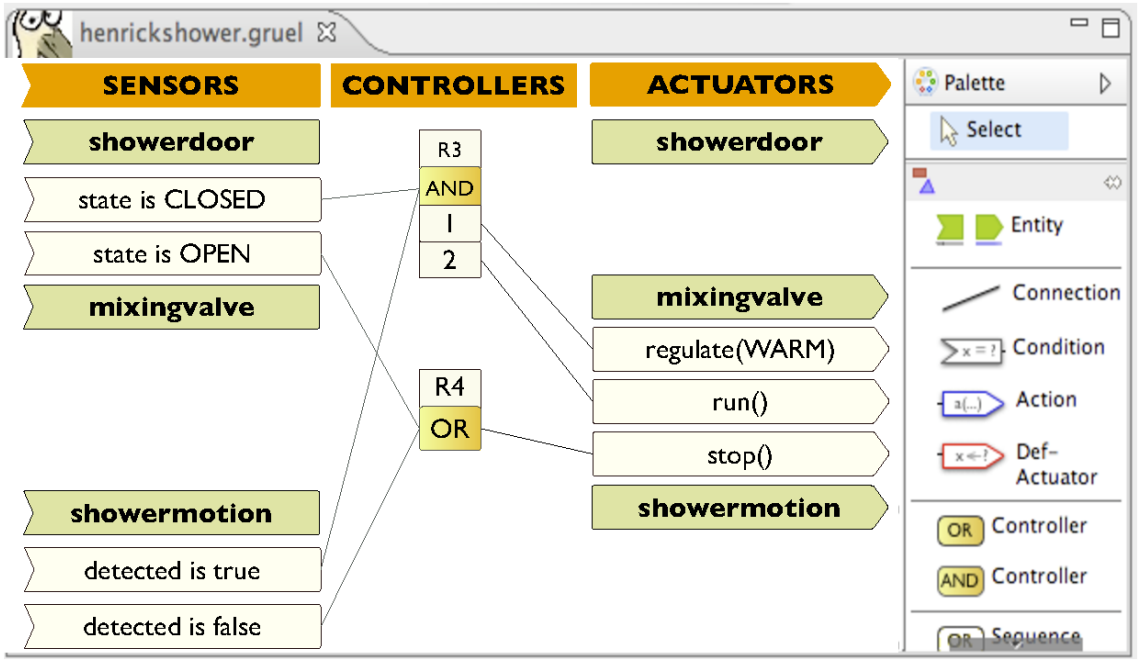

Figure 3. A screenshot of the Pantagruel graphical editor (2)

Pervasive computing systems are being deployed in a growing number of areas, including building automation, assisted living, and supply chain management. These systems involve a wide range of devices and software components, communicate using a variety of protocols, and rely on intricate distributed systems technologies. Besides requiring expertise on underlying technologies, developing a pervasive computing application also involves domain-specific architectural knowledge to collect context information, process it and perform actions. Because of the heterogeneity of the devices, their combination is often achieved by using ad hoc design and implementation approaches. As a consequence, the resulting platforms are usually closed and limited, preventing usage scenarios from evolving and impeding creativity.

To address this challenge, we have developed a language-based approach [14] for managing the development life-cycle of applications coordinating networked entities. Our approach covers the characterization of the environment, the specification of coordination applications, the verification of the environment and its deployment. It is carried out in practice by a tool platform, named DIASUITE that covers the whole cycle of development (from the architectural specification to the deployment). A prominent example area is pervasive computing. Our work in this area has been validated with numerous applications for assisted living, building management [14], and advanced telephony services [12].

Our contributions can be summarized as follows:

- *A design language.* We have introduced DIASPEC, a design language dedicated to describing a taxonomy of area-specific entities and application architectures. This design language provides a conceptual framework to support the development of a pervasive computing system, assigning roles to the stakeholders and providing separation of concerns. DIASPEC raises the level of knowledge that can be shared and reused by the stakeholders.
- *A tool-based methodology.* We have built DIASUITE, a suite of tools which, combined with our design language, provides a conceptual framework and support for each phase of the development of a pervasive computing system. A DIASPEC description is compiled by DIAGEN to produce a dedicated high-level programming framework. DIASIM allows to simulate a pervasive computing application prior to its deployment. A back-end targets a specific distributed systems platform.
- *Validation.* We have developed a variety of applications in areas including advanced telecommunications, home/building automation, and healthcare. Our largest case study is a building management system involving 6 applications, 35 classes of entities and over 400 entity instances.

6.2. A Taxonomy-Driven Approach to Visually Prototyping Pervasive Computing Applications

Ubiquitous computing environments are physical spaces (e.g. houses) characterized by a wide use of technologies, like mobile devices, sensors, software components. These entities can be orchestrated in order to automating some everyday tasks for the inhabitants of such space, relieving them of some routines. Orchestrating entities relies on various information, like device-sensed data (e.g. temperature), user settings (e.g. agenda meetings), or computed data (e.g. calculated means).

One of the challenges addressed by the area of pervasive computing is to provide tools for end-users or novice programmers, so that they can easily program applications on pervasive environments like home, hospital, museums, or any building that aim at helping their users' everyday life. To address this challenge, we formalize a development approach [16] that aims at writing high-level specifications of the orchestration of networked entities in a pervasive computing environment. This work is based on Pantagruel, a visual domain-specific language.

The main contributions of this work are the following:

- *Area-specific approach.* We have introduced a novel approach to visual programming of pervasive computing applications that is parameterized with respect to a description of a pervasive computing environment. This makes our approach applicable to a range of areas.

- *Area-specific visual programming.* We have extended the sensor-controller-actuator paradigm to allow the programming of the orchestration logic to be driven by an environment description. This extended approach eases programming and enables verifications.
- *Validation.* We have implemented a compiler and successfully used it for applications in various pervasive computing areas such as home automation and building management.

6.3. A Parameterized Simulator for Pervasive Computing Environments

Pervasive computing applications involve both software and integration concerns, for the constituent networked devices of the pervasive computing environment. This situation is problematic for testing because it requires acquiring, testing and interfacing a variety of software and hardware entities. This process can rapidly become costly and time-consuming when the target environment involves many entities.

To cope with widely heterogeneous entities, we have introduced a simulator [18] parameterized with respect to a high-level description of a pervasive computing environment. The main contributions of this work are the following:

- *Generated simulation support.* A pervasive computing environment description is used to generate both an emulation layer to execute applications and a simulation programming framework to develop simulated entities. To abstract over distributed systems technologies, DIASUITE follows a layered architecture for the generated programming frameworks. This approach has made it possible to introduce a simulated environment as just another technology underlying the programming framework.
- *Simulation renderer.* The DIASIM simulation renderer enables the developer to visually monitor and debug a pervasive computing system. Once the primitive services are simulated, we define simulation scenarios to test the pervasive computing system. A simulation scenario is defined for a given spatial layout of services. It consists of a set of initial stimuli and a set of evolution rules for these stimuli. Because of the number of entities involved in a pervasive computing system, a simulation scenario rapidly becomes complicated to follow. To circumvent this problem, we have coupled DiaSim with an existing visualization tool: the Siafu open source context simulator [34]. Siafu is parameterized with respect to information automatically generated from DIASPEC declarations.
- *Hybrid simulation.* Our approach makes it possible for the same code to be simulated or executed in the actual environment. We ensure a functional correspondence between a simulated environment and an actual one by requiring both implementations to be in conformance with the pervasive computing environment description.
- *Validation.* Our approach has been implemented in a tool called DIASIM. This tool has been used to simulate different pervasive computing systems, demonstrating the generality of our parameterized approach. In particular, DIASIM has been used for simulating a 13,500 square meters building, hosting the ENSEIRB (an engineering school) and research groups [18].

7. Contracts and Grants with Industry

7.1. Designing techniques and tools for developing domain-specific languages – Industrial Fellowship (CIFRE / Thales)

Participants: Charles Consel, Zoé Drey.

The goal of this project is to develop a connection between the domain-specific languages and the model driven engineering. We would like to take profit from methodologies, techniques and tools that come from model driven engineering, in order to ease the design and implementation of a domain-specific language (DSL). On another side, the model driven engineering could be combined with the DSL techniques to complete the pure-model vision in a software engineering process where modelling concepts do not suffice or are not relevant. This work will be illustrated and validated with a concrete case study.

7.2. Integrating non-functional properties in an Architecture Definition Language and its execution environment – Industrial Fellowship (CIFRE / Thales)

Participants: Charles Consel, Julien Bruneau.

The goal of this project is to add non-functional properties in the DIASPEC language and in the DIAGEN generator. More especially, these non-functional properties are considered on three different levels:

- *The component level.* The non-functional properties define temporal, physical and software constraints restrictive for a component.
- *The component coupling level.* The non-functional properties define the dependency between the components as well as the Quality of Service provided and required by each component of the environment.
- *The software architecture level.* The non-functional properties describe the resources that must be allocated to a component (memory, processing capacity). They also define the necessary resources for a component to interact with other components (network QoS).

This work will be illustrated and validated with a concrete application.

7.3. SmartImmo: Towards intelligent and environmentally-friendly buildings (french competitiveness pole)

Participants: Charles Consel, Benjamin Bertran, Ghislain Deffrasnes.

The SmartImmo project gathers research groups in pervasive systems and french companies working in the building construction, installation, and management. This project led by Orange Labs aims to make a building able to "communicate" with its occupants and to be environmentally-friendly (*e.g.,* automatic temperature adjusting).

The main objectives of this project are to design a M2M (Machine-To-Machine) box for the heterogeneous equipment communication and to build several services on top of this platform. This project is funded by the SCS (Secured Communicating Solutions), a french pole of competitiveness.

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).
- Calton Pu, Georgia Institute of Technology (USA, Atlanta).
- Pierre Cointe, École des Mines in Nantes (France, Nantes).

8.2. Visits and Invited Researchers

The Phoenix group has been visited by:

- Olivier Danvy (Associate Professor at the University of Aarhus, Denmark), from January 23, 2009 to January 25, 2009;
- Pierre-Etienne Moreau (Professor at the École des Mines, Nancy, France), February 6, 2009;
- Didier Donsez (Professor at the Université Grenoble 1, France), March 7, 2009;
- Roy Campbell (Professor at the University of Illinois in Urbana-Champaign, US) from March 6, 2009 to March 11, 2009;
- Walid Taha (Professor at the Rice University in Houston, US), from November 1, 2009 to November 3, 2009;
- Julia Lawall (Associate Professor at the University of Copenhagen, Denmark), from December 6, 2009 to December 12, 2009.

9. Dissemination

9.1. Scientific Community Participation

Charles Consel has been involved in the following events as:

- Program Committee member of
  - ICMT 2009 (International Conference on Model Transformation),
  - IPTComm 2009 (International Conference on Principles, Systems and Applications of IP Telecommunications),
  - SLE 2009 (International Conference on Software Language Engineering),
  - GPCE 2009 (International Conference on Generative Programming and Component Engineering),
  - NFM 2009 (The First NASA Formal Methods Symposium);
- Guest Editor for the Annals of Telecommunications, Springer;
- Member of the scientific commitee on "GDR génie de la programmation du logiciel" (CNRS);
- Member of the INRIA working group on research and perspectives in the domain "Réseaux, systèmes et services, calcul distribué";
- Member of the ANCRE program ("Alliance Nationale sur l'Énergie").

Charles Consel has participated in the following thesis defense committees:

- Caroline Lu, Université de Toulouse/INP, December 14, 2009 ("Robustesse du logiciel embarqué multicouche par une approche réflexive: application à l'automobile").

9.2. Teaching

Charles Consel has been teaching Master 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. These labs are supervised by Benjamin Bertran and Julien Bruneau.

Charles Consel and Damien Cassou are teaching a course on Architecture Description Languages.

9.3. Presentations and Invitations

Charles Consel gave a number of invited presentations:

- at Georgia Institue of Technology in July 2009,
- at the University of California Los Angeles in July 2009,
- at Tsinghua University in Beijing in April 2009.

9.4. PhD Thesis

One student of the Phoenix group obtained his PhD in 2009:

- Wilfried Jouve, "Approche déclarative pour la génération de canevas logiciels dédiés à l'informatique ubiquitaire" [11].

10. Bibliography

Major publications by the team in recent years

[1] D. CASSOU, B. BERTRAN, N. LORIANT, C. CONSEL. *A Generative Programming Approach to Developing Pervasive Computing Systems*, in "GPCE'09: Proceedings of the 8th international ACM SIGPLAN conference on Generative programming and component engineering, Denver USA", ACM, 2009, p. 137-146, http://hal.inria.fr/inria-00405819/en/.

[2] C. CONSEL. *From A Program Family To A Domain-Specific Language*, Lecture Notes in Computer Science, State-of-the-Art Survey, n^{o} 3016, Springer-Verlag, 2004, p. 19–29, http://phoenix.labri.fr/publications/papers/dagstuhl-consel.pdf.

[3] 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 DK .

[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. *A DSL Paradigm for Domains of Services: A Study of Communication Services*, Lecture Notes in Computer Science, State-of-the-Art Survey, n^{o} 3016, Springer-Verlag, 2004, p. 165–179, http://phoenix.labri.fr/publications/papers/dagstuhl04_consel_reveillere.pdf.

[6] Z. DREY, J. MERCADAL, C. CONSEL. *A Taxonomy-Driven Approach to Visually Prototyping Pervasive Computing Applications*, in "DSL'09: 1st IFIP Working Conference on Domain-Specific Languages, Royaume-Uni Oxford", vol. 5658, 2009, p. 78–99, http://hal.inria.fr/inria-00403590/en/.

[7] 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 DK .

[8] 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 US .

[9] 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.

[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.

Year Publications

Doctoral Dissertations and Habilitation Theses

[11] W. JOUVE. *Approche déclarative pour la génération de canevas logiciels dédiés à l'informatique ubiquitaire*, Université Sciences et Technologies - Bordeaux I, 04 2009, http://tel.archives-ouvertes.fr/tel-00402605/en/, Ph. D. Thesis.

International Peer-Reviewed Conference/Proceedings

[12] B. BERTRAN, C. CONSEL, P. KADIONIK. *A SIP-based home Automation Platform: an experimental study*, in "ICIN'09: 13th International Conference on Intelligence in Service delivery Networks, France Bordeaux", IEEE, 2009, http://hal.inria.fr/inria-00406248/en/.

[13] J. BRUNEAU, W. JOUVE, C. CONSEL. *DiaSim: A Parameterized Simulator for Pervasive Computing Applications*, in "Mobiquitous'09: 6th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Canada Toronto", IEEE, 2009, http://hal.inria.fr/inria-00403421/en/.

[14] D. CASSOU, B. BERTRAN, N. LORIANT, C. CONSEL. *A Generative Programming Approach to Developing Pervasive Computing Systems*, in "GPCE'09: Proceedings of the 8th international conference on Generative programming and component engineering, États-Unis d'Amérique Denver, CO", ACM, 2009, p. 137-146, http://hal.inria.fr/inria-00405819/en/.

[15] R. DOUENCE, X. LORCA, N. LORIANT. *Lazy Composition of Representations in Java*, in "SC'09: International Conference on Software Composition, France Lille", vol. 5634, Springer Verlag, 2009, p. 55-71, http://hal.inria.fr/inria-00403417/en/.

[16] Z. DREY, J. MERCADAL, C. CONSEL. *A Taxonomy-Driven Approach to Visually Prototyping Pervasive Computing Applications*, in "DSL'09: 1st IFIP Working Conference on Domain-Specific Languages, Royaume-Uni Oxford", vol. 5658, 2009, p. 78-99, http://hal.inria.fr/inria-00403590/en/.

[17] S. DUCASSE, D. POLLET, A. BERGEL, D. CASSOU. *Reusing and Composing Tests with Traits*, in "TOOLS-EUROPE'09: 47th International Conference on Objects, Components, Models and Patterns, États-Unis d'Amérique New York", vol. 33, Springer, 2009, p. 252-271, http://hal.inria.fr/inria-00403568/en/.

Other Publications

[18] J. BRUNEAU, A. BLANQUART, N. LORIANT, C. CONSEL. *A Parametrized Simulator for Pervasive Computing Applications*, 2009, http://hal.inria.fr/inria-00412564/en/, Demonstration at ICPS'09.

References in notes

[19] *CGI: The Common Gateway Interface*, 1993, http://www.w3.org/CGI/.

[20] *HomeSip Project: Using the SIP Protocol for Home Automation and M2M*, 2006, http://www.enseirb.fr/cosynux/HomeSIP.

[21] *Session Initiation Protocol (SIP)*, March 2001, Request for Comments 3261.

[22] 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, p. 111–119.

[23] 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.

[24] 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 DK .

[25] 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.

[26] A. K. DEY, G. D. ABOWD, D. SALBER. *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications*, in "Human-Computer Interaction", vol. 16, n^{o} 2, 2001, p. 97–166 US .

[27] 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.

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

[29] 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 DK .

[30] 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 DK .

[31] 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.

[32] 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.

[33] 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.

[34] M. MARTIN, P. NURMI. *A Generic Large Scale Simulator for Ubiquitous Computing*, in "Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services, 2006 (MobiQuitous 2006), San Jose, California, USA", IEEE Computer Society, July 2006, http://dx.doi.org/10.1109/MOBIQ.2006.340388DEFI.

[35] 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 US .

[36] N. MEDVIDOVIC, R. N. TAYLOR. *A Classification and Comparison Framework for Software Architecture Description Languages*, in "IEEE Transactions on Software Engineering", vol. 26, n^{o} 1, 2000, p. 70–93, http://dx.doi.org/10.1109/32.825767US.

[37] 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", ACM Press, September 2000, p. 19-24.

[38] 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.

[39] 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.

[40] 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.

[41] 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 DK .

[42] 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 DK .

[43] 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 DK .

[44] 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 DK .

[45] 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.