INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team OBASCO

OBjects, ASpects, and COmponents

Rennes - Bretagne Atlantique

THEME COM

Activity Report

2007

Table of contents

1. Team **1**
2. Overall Objectives **1**
2.1. Presentation 1
2.2. Highlights of the year 2
3. Scientific Foundations **2**
3.1. Genesis 2
3.2. Post Object-Oriented Languages 3
3.2.1. From Objects to Components 3
3.2.2. From Objects to Aspects 4
3.2.3. From Aspects to Domain-Specific Languages 6
4. Application Domains **6**
4.1. Overview 6
4.2. Middleware and Enterprise Information Systems 7
4.3. (Distributed) Operating Systems 7
4.4. Autonomic Computing 7
5. Software **8**
5.1. Arachne 8
5.2. Bossa 9
5.3. FScript 10
5.4. Reflex 10
5.5. Awed 11
5.6. Baton 11
6. New Results **12**
6.1. Components 12
6.1.1. Support for Explicit Protocols 12
6.1.1.1. Extending Components with Aspects using Protocols 12
6.1.1.2. Java Implementation of Rendezvous for Symbolic Transition Systems 12
6.1.2. Support for Component Verification 13
6.1.2.1. Ensuring composition properties using VPA-based protocols 13
6.1.2.2. Runtime and Library Support for Formal Components 13
6.1.3. Support for Component Adaptation 13
6.1.3.1. Invasive Patterns to Address Modularization Problems in CBSE 13
6.1.3.2. Safe Dynamic Reconfigurations of Fractal Architectures 14
6.1.3.3. A Declarative Approach for QoS-Aware in Web Service Orchestrations 14
6.1.4. Application to Software Product Lines 14
6.2. Aspects 14
6.2.1. Formal models for AOP 15
6.2.1.1. Concurrent EAOP 15
6.2.1.2. VPA-based Aspects 15
6.2.1.3. Common Aspect Semantics Base (CASB) and Preserved Properties 16
6.2.2. Language Design 16
6.2.2.1. Type Systems for Aspect Languages 16
6.2.2.2. Aspects With Explicit Distribution (AWED) 16
6.2.3. Aspects in Practice 17
6.2.3.1. Design Pattern Aspectualization : AspectJ and JHotDraw 17
6.2.3.2. Invasive Patterns for Distributed System 17
6.2.3.3. Aspects for C Applications 17
6.2.3.4. Distribution of an Automatic Toll System 18
6.3. DSLs 18

6.3.1. Process scheduling and energy management 18
6.3.2. Automating Collateral Evolutions in Linux Drivers 18
6.3.3. Zebu: a DSL for Improving the Robustness of Network Application Protocols 19
7. Contracts and Grants with Industry 19
7.1. Siemens AG, Munich 19
7.2. VirtualLogix Cifre grant 19
7.3. France Télécom R&D PhD about Web Services 19
7.4. France Télécom R&D PhD about Fractal Architectures 20
8. Other Grants and Activities 20
8.1. Regional Actions 20
8.1.1. MILES project / Software Engineering Cluster 20
8.1.2. Arantèle project 20
8.2. National Projects 20
8.2.1. ARC INRIA VeriTLA+ 20
8.2.2. ANR/RNTL Selfware 21
8.2.3. ANR/RNTL SADAJ 21
8.2.4. ANR non thématique Coccinelle 21
8.2.5. ANR non thématique FLFS (Languages Family for Systems Family) 22
8.3. European Projects 23
8.3.1. NoE AOSD-Europe 23
8.3.2. STREP AMPLE 23
8.4. Collaboration with Foreign Research Groups 24
8.4.1. ECO-NET Project # 16293RG 24
8.4.2. CONICYT Chili - INRIA CORDIAL Project 24
9. Dissemination 24
9.1. Animation of the Community 24
9.1.1. Animation 24
9.1.2. Steering, journal, conference committees 25
9.1.3. Thesis committees 26
9.1.4. Evaluation committees and expertise 26
9.2. Teaching 26
9.3. Collective Duties 26
10. Bibliography 27

1. Team

OBASCO is a joint project between École des Mines de Nantes (EMN) and INRIA.

Head of Project-team
Pierre Cointe [Professor, EMN, HdR]

Vice-head of Project-team
Mario Südholt [Associate Professor, EMN, HdR]

Staff Member INRIA
Jean-Marc Menaud [CR2 INRIA on leave from EMN 1/09/2006-30/08/2008]

Faculty Members from EMN
Rémi Douence [Associate Professor]
Thomas Ledoux [Associate Professor]
Gilles Muller [Professor, HdR]
Jacques Noyé [Associate Professor]
Jean-Claude Royer [Professor, HdR]

Administrative Assistants
Diana Gaudin [Part-time (50%)]
Élodie Lizé [Part-time (50%)]

Post-doctoral Fellows
Pierre-Charles David [ANR SELFWARE grant]
Joost Noppen [IST AMPLE grant 1/09/2007-]
Yoann Padioleau [ANR COCCINELLE grant 1/10/2005-1/08/2007]

Visiting Scientist
Julia Lawall [DIKU, University of Copenhagen, 29/05/07-27/07/2007]

Ph. D. Students
Hugo F. Arboleda Jimenez [MINES & Los Andes (Bogota, Colombia) University grant, 1/1/2006-]
Ali Assaf [MESR grant 3/10/2005-]
Christophe Augier [Cifre grant with JALUNA 1/10/2004-]
Luis Daniel Benavides Navarro [MINES grant, 1/10/2005-]
Simon Denier [EMN grant 1/10/2003-30/08/2007]
Simplice Djoko Djoko [INRIA grant shared with the project PopArt, 1/10/2005-]
Fabricio de Alexandria Fernandes [CAPES grant from Brazil 1/10/2006-]
Fabien Hermenier [EMN grant 1/10/2006-]
Kelly Garcés [ANR FLFS grant 1/10/2007-]
Nicolas Loriant [EMN grant 1/10/2004-]
Dong Ha Nguyen [REGIONAL COUNCIL & AOSD-Europe grant, 15/04/2005-]
Angel Núñez [MINES & STREP AMPLE grant 1/10/2006-]
Richard Urunuela [REGIONAL COUNCIL grant 1/10/2004-]

2. Overall Objectives

2.1. Presentation

OBASCO addresses the general problem of adapting software to its uses by developing tools for building software architectures based on components and aspects [53]. We are developing and (re)using results developed in the programming languages and software engineering areas, in particular object-oriented technologies.

Our perspective is the evolution from programming in the small, as supported by object-oriented languages *à la* Smalltalk, Java, and C#, towards programming in the large, as it emerges with component models. We contribute to the evolution from an object model to a unified model supporting programming in the large and adaptation by integrating objects and aspects on the one hand, objects and components on the other hand. We are working along three directions:

Component-Oriented Programming: Definition of a language mechanisms and implementation techniques (i) to develop explicit protocols to support composition properties both at the structural and behavioral level and (ii) to manage their adaptation all along their life cycle. To this aim, we rely on techniques from generative programming, in particular, reflection and specialization techniques, and explore the use of aspect-oriented methods in component-based systems. We are also looking at how to interface such a language with *de facto* industrial standards such as EJB, .NET, and CCM.

Aspect-Oriented Programming: Study of more expressive languages for aspect-oriented programming in order to support more declarative modularization of crosscutting concerns. We are particular interested in the development of a (formal) foundation for AOP of concurrent and distributed systems and in optimized implementations, in particular, using computational reflection, as well as program analysis and transformation techniques.

Domain Specific Languages: Methodology to develop DSLs in general and Domain Specific Aspect Languages (DSALs) in particular. More generally we consider coupling DSLs and aspect oriented languages, to express program transformations in both a secure and expressive ways.

In order to question and validate our approach, we are developing applications with a focus on the various layers of enterprise information systems: from operating systems, via system-level components and middleware to web services.ovov

2.2. Highlights of the year

- Mario Südholt wrote a synthesis of his work on formalizing, modeling and implementing Aspect-Oriented languages. This synthesis is the core of his HDR thesis defended in July 2007 [16]. As part of his work Mario Südholt has provided a comprehensive account of a large set of approaches to AOP and the relationship between AOP and component-based programming. Concretely, based on a set of characteristics of the model of Event-based AOP, he has presented a unifying presentation of the formal properties of this model and an overview of the large variety of aspect languages. Furthermore, he has presented how aspects over interaction protocols realize a useful integration of AOP with white-box and grey-box models of composition. Finally, he has shown how modularization problems in several large-scale real-world applications can be handled using this approach.
- Mario Südholt and Charles Consel co-edited the ECOOP'06 workshops reader [13] that appeared in 2007. This reader gives a survey of challenging questions in the field of CBSE.

3. Scientific Foundations

3.1. Genesis

The OBASCO project was created in 2003 to investigate the possibility of a *continuum* between objects, aspects and components to reason about adaptable software architectures [47], [53], [50]. Consequently we study formal model of objects and components and we implement prototypes to investigate new programming paradigms with a particular emphasis on metaprogramming. aspect-oriented programming (AOP) and domain specific languages (DSLs).

Historically the core members of OBASCO have a strong background in the design and implementation programming languages (more particularly OOL) [51], [44],[10]. This background has been enriched by an expertise in middleware and operating systems [64][8]. Our goal is to take advantage of this complementarity by developing a methodology and a set of techniques covering in a uniform way the software process from the OS level to the application level.

3.2. Post Object-Oriented Languages

Object: *An object has a set of "operations" and a "state" that remembers the effect of operations. Objects may be contrasted with functions, which have no memory. A language is object-based if it supports objects as a language feature* (page 168 of [83]).

Components: *Components are for composition. Composition enables prefabricated components to be reused by rearranging them in ever-new composites. Software components are executable units of independent production, acquisition and deployment that can be composed into a functioning system. To enable composition a software component adheres to a particular component model and targets a particular component platform* [77].

Aspects: *Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways. Examples of aspects include memory access patterns and synchronization of concurrent objects* (page 226 of [63]).

Reflection: *A process's integral ability to represent, operate on, otherwise deal with itself in the same way that it represents, operates and deals with its primary subject matter* [76].

DSL: *A domain-specific language (DSL) is a specialized, problem-oriented language. Domain-specific languages play an important role in Generative Programming because they are used to "order" concrete members of a system family* (page 137 of [55]).

Component-based systems and reflection have been research topics for many years and their importance has grown in tandem with the success of object-oriented languages. Since the end of the seventies, Object-Oriented technology has matured with the realization of a considerable amount of industrial projects based on languages such as Smalltalk, ObjectiveC, C++, Eiffel or Java. Today, the support of objects as a programming language feature is a *de facto* standard, in particular, in industry.

But the construction of large software system, which constitute the main challenge of today's software industry, has revealed a number of deficiencies in the object-oriented paradigm. Issues such as how to build reliable systems out of independently developed components or how to adapt system behavior to new application-specific requirements have now to be addressed [27], [26].

3.2.1. From Objects to Components

The generalization of object-oriented languages has contributed[1] to improve software reusability. In spite of this first success, there is still work to do. This is due to the inherent difficulties of the OO *white-box* model of reuse whereby reusing a class through inheritance (or an object through cloning) requires a good understanding of the *implementation* of the class (or object). The applications have also changed of scale and scope. Integrating heterogeneous pieces of software, based on shared technical services (distribution, transactions, security...), has become a fundamental issue. Taking these issues into account has led to the importance of *software components* [77] as building blocks for today's software systems. The basic idea, as initially explained by M.D. McIlroy in 1968 [66], is to industrialize software reuse by setting up both an industry and a market of interchangeable parts. This corresponds to a strong decoupling between component producers and consumers, with new stages in the life cycle of a component (*e.g.*, packaging, deployment). This also leads to a kind of layered programming *in the small/in the large* with standard object-oriented languages

[1]See the discussions at http://www.dreamsongs.com/Essays.html and those at http://www-poleia.lip6.fr/~briot/colloque-JFP/, in particular [53][27].

used to implement *primitive* components, and a *component-oriented* language used to implement *compound* components, which can also be seen as *software architectures*. The two main features that a component-oriented language should support are:

- *composability*: A component strongly encapsulates its *implementation* behind an *interface* and represents a unit of composition (also called *assembly*). Composition relates *provided* and *required services* (*e.g.*, methods) with well-defined interaction protocols (with synchronous or asynchronous communications) and properties. This defines the structure and behavior of the compound component. Ideally, this composition should be language neutral (with respect to the implementation language).
- *adaptability*: A component is designed as a generic entity that can be adapted to its different context of uses, all along its life cycle. This adaptation can be static (*e.g.*, at assembly time) but also dynamic (*e.g.*, at runtime). Very flexible architectures can be created by considering components as first-class citizens (*e.g.*, by being able to return a component as the result of a service). This has to be contrasted with the standard notion of *module*.

These properties raise new challenges in programming language design and implementation. They require an integration of ideas coming from module interconnection languages [74], architecture description languages (ADLs) [75], [67] and object-oriented languages. Modules provide an interesting support for component structure. In particular, recent proposals around so-called *mixin modules* combine parametrization, recursive module definitions, and late binding. ADLs address many of the above-mentioned issues although at a description level, rather than programming level. Finally, object-oriented languages remain a major source of inspiration. Interesting extensions have indeed been worked out in this context like notions of explicit protocols that can be seen as finite state automata but also integrated within the language as types. Recently, a number of *connection-oriented languages* [77] have been developed as Java extensions. These languages focus on component structure.

At the implementation level, an important issue is the exacerbated conflict between *early* and *late binding* due to, on the one hand, strong encapsulation and the need to address errors as early as possible in the life cycle, and, on the other hand, the possibility to adapt a component all along its life cycle. Software specialization (*e.g.*, partial evaluation [60]) and reflection have a key role to play here.

3.2.2. From Objects to Aspects

Join points: well defined points in the execution of a program that can be referred to by aspects.

Pointcuts: means of referring to collections of join points and the corresponding execution context at those join points (in order to execute associated advice at these join points).

Advice: definition of a modification an aspect performs when a pointcut matches, that is, behavior that is added to or executed instead of join points.

Aspect: (i) concern crosscutting a set of traditional modular units (classes, packages, modules, components...); (ii) corresponding unit of modularization for such concerns and that is typically defined in terms of pointcuts and advice.

Weaver: tool that takes a base program and several aspects and produces an executable (woven) program.

Aspect interaction: two aspects interact if weaving them in different orders yields execution computing different results. A frequently-used specific notion of interaction is interactions stemming from the simultaneous application of different aspects at one join point matched by a pointcut.

The object-oriented and reflective communities, together, have clearly illustrated the potential of separation of concerns in the fields of software engineering and open middleware [81]. Aspect-oriented software development is a recent and very active field of research whose key characteristics are [2]:

- *abstractions for the modularization of crosscutting concerns:* These new units of independent behaviors called aspects, support the identification, the encapsulation and then the manipulation of a set of properties describing a specific domain (such as distribution, transactions, security...),
- *non-invasiveness:* When taking into account new concepts, goals, needs or services, and to satisfy the modularity principle, the added aspects should not pollute the base application. Consequently, the aspects have to be specified as independent units in a non-intrusive way and then woven with the associated base program.

Historically, object-oriented languages have contributed to the field of *separation of concern* in - at least - two different ways:

Reflection: The reflective approach makes the assumption that it is possible to separate in a given application, its *why* expressed at the base level, from its *how* expressed at the metalevel.

- In the case of a reflective programming language *à la Smalltalk*, the principle is to reify at the metalevel its structural representation *e.g.,* its classes, their methods and the error-messages but also its computational behavior, *e.g.,* the message sending, the object allocation and the class inheritance. Depending on which part of the representation is accessed, reflection is said to be structural or behavioral. Meta-objects protocols (MOPs) are specific protocols describing at the meta-level the behavior of the reified entities. Specializing a given MOP by inheritance, is the standard way [51] [52], [62] to extend the base language with new mechanisms such as multiple inheritance, concurrency or metaclass composition [45].
- In the case of open middleware [64], the main usage of behavioral reflection is to control message sending by interposing a metaobject in charge of adding extra behaviors/services (such as transaction, caching, distribution) to its base object. Nevertheless, the introduction of such *interceptor/wrapper* metaobjects requires to instrument the base level with some *hooks* in charge of causally connecting the base object with its metaobject [9].

Design Pattern: The *Model-View-Controller* (MVC) developed for Smalltalk [57] is the first design-pattern making the notion of aspects explicit. The main idea was to separate, at the design level, the *model* itself describing the application as a class hierarchy and two separate concerns: the *display* and the *control*, themselves described as two other class hierarchies. At the implementation level, standard encapsulation and inheritance were not able to express these crosscutting concerns and not able to provide the coupling between the model, its view, and its controller. This coupling necessitated:

- the introduction of a *dependence mechanism* in charge of notifying the observers when a source-object changes. This mechanism is required to automatically update the display when the state of the model changes.
- the instrumentation of some methods of the model to raise an event each time a given instance variable changes its value.

On the one hand, object-oriented languages have demonstrated that *reflection* is a general conceptual framework to clearly modularize implementation concerns when the users fully understand the metalevel description. In that sense, reflection is solution oriented since it relies on the protocols of the language to build a solution. On the other hand, the *MVC* design-pattern has provided the developer with a problem-oriented methodology based on the expression and the combination of three separate concerns/aspects. The *MVC* was the precursor of *event programming* - in the Java sense - and contributed to the emergence of aspect-oriented programming by making explicit the notion of *join-point*, *e.g.,* some well defined points in the execution of a *model* used to dynamically weave the aspects associated to the *view* and the *controller*.

Conclusion: We have identified the following main issues that OBASCO strives to address concerning the relationship between computational reflection and aspects [2]. A first issue is to get a better understanding of how to use reflective tools to model aspects languages and their associated crosscutting and advice languages [10], [79]. A second issue is to study the integration of aspects and objects to i) propose an alternative to inheritance as a mechanism for reuse and to ii) reify design patterns . A third issue is to emphasize the use of reflection in the field of generic programming and component adaptation as soon as self-reasoning is important [48]. A fourth issue is to apply domain-specific languages to the definition of aspects [49].

3.2.3. From Aspects to Domain-Specific Languages

Domain-Specific Languages (DSLs) are programming languages dedicated to a particular application domain. They represent a proven approach to raising the abstraction level of programming. They offer high-level constructs and notations dedicated to a domain, structuring program design, easing program writing, masking the intricacies of underlying software layers, and guaranteeing critical properties [72], [69], [65].

A DSL is a high-level language providing constructs appropriate to a particular class of problems. The use of such a language simplifies programming, because solutions can be expressed in a way that is natural to the domain and because low-level optimizations and domain expertise are captured in the language implementation rather than being coded explicitly by the programmer. The avoidance of low-level source code in itself improves program robustness. More importantly, the use of domain-specific constructs facilitates precise, domain-specific verifications, that would be costly or impossible to apply to comparable code written in a general-purpose language (*e.g.* verification of termination properties) [80].

The advantages of DSLs have drawn the attention of rapidly evolving markets (where there is a need for building families of similar software, *e.g.*, product lines), as well as markets where reactivity or software certification are critical: Internet, cellular phones, smart cards, electronic commerce, embedded systems, bank ATM, etc. Some companies have indeed started to use DSLs in their development process: ATT, Lucent Technologies, Motorola, Philips, and Microsoft.

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 [54].

For OBASCO, DSLs are also interesting because they can be used as a model to describe specific and crosscutting aspects of a system.

4. Application Domains

4.1. Overview

Keywords: *autonomic computing*, *enterprise information systems*, *middleware*, *operating systems*.

At the application level, our goal is to improve support for software components manufacturing (see technology #7 of [56]). Because of its distributed nature, component-based technology is a key technology in the field of telecommunication and enterprise information systems. When such software components have to be produced just in time, it becomes strategic to define product lines for generating components in an automated fashion (see 8.3). OBASCO is investigating new paradigms related to separation of concerns and automatic software generation captured by the so called generative programming [55], [48]. In particular OBASCO is coupling aspect-oriented programming, components and domain specific languages in the attempt to provide adaptable architectures.

Concretely, we are currently working in three directions: (i) in the middleware area and enterprise information systems we are considering system configuration and aspectualization of non-modular functionalities. (ii) in the OS field we provide support for advanced crosscutting functionalities, such as process scheduling, drivers evolution and grid services; (iii) in the domain of autonomic computing we investigate how to dynamically adapt software components to their execution contexts;

4.2. Middleware and Enterprise Information Systems

Stimulated by the growth of network-based applications, middleware technologies become increasingly important. They cover a wide range of software systems, including distributed objects and components, mobile applications and finally ubiquitous computing. As part of the so-called process of Enterprise Application Integration (EAI), companies and organizations are now using middleware technologies to build enterprise-wide information systems by integrating, and partially developing new glue code for, previously independent applications.

We have, among others, investigated the evolution of component-based platforms through a detailed analysis of modularization problems of distribution and transaction functionalities of the replicated cache infrastructure JBoss Cache. We have shown how aspects with explicit abstractions for distribution allow to improve the structure of such applications and therefore facilitate their evolution, in particular through a new notion of invasive distributed programming patterns. These abstractions, which have been embodied in the aspect system AWED (for Aspects With Explicit Distribution) have also been applied to the modularization of distributed web services compositions. The AWED system is currently applied to the distribution of automatic satellite-based toll systems in the context of an industrial cooperation with Siemens AG from Munich, Germany (see 7.1).

Finally, we have started to investigate Software Product Lines (SPLs). A SPL is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. Core assets are produced and reused in a number of products that form a family. These core assets may be documents, models, etc, and of course, software components. SPLs share several important open issues which middlewares for distributed applications, in particular, the configuration of large software systems in the presence of large potential variability and the importance of non-modular functionalities. Aspect-oriented software development can improve the way in which software is modularized, localizing its variability in independent aspects as well as improving the definition of complex configuration logic to customize SPLs (see the AMPLE STREP in 8.3).

4.3. (Distributed) Operating Systems

The development of operating systems is traditionally considered to be an activity on the fringe of software development. In fact, the lack of systematic methodologies for OS design often translates into closed systems that are difficult to extend and modify. Too often generality is sacrificed for performance. The widespread use of unsafe programming languages, combined with extensive manual optimizations, compromises the safety of OS software. The use of DSL is a promising approach to address these issues [46], [70].

A first application direction is to use DSALs to safely program OS behavior (strategies) independently of the target system; a weaver automatically integrates the code of such an aspect into the relevant system components. This approach separates strategies, which are programmed using aspects, from the underlying mechanisms, and thus simplifies system evolution and extension. This combination of AOP and DSL has been validated in the context of Bossa (see 5.2).

A second application direction is to use AOP for re-engineering or dynamically evolve existing complex system such OS device drivers and Grid middlewares like the *Open Grid Service Architecture* (OGSA). AOP helps to develop complex distributed services by facilitating crosscutting functionalities integration such as the grid power management service.

4.4. Autonomic Computing

Distributed systems are becoming more and more complex due to their heterogeneous architectures (on the physical and software level) in which resources are numerous and whose availability evolves at runtime. This complexity and dynamicity requires permanent adaptation, from the application to the system level, and requires to automate the adaptation process.

The essence of autonomic computing is system self-management, freeing administrators and developers of low-level task configuration and management whilst delivering an optimized solution. In an autonomic system, the human operator defines general high level policies and rules that serve as an input for the adaptation process. These policies can be categorized into four functional classes [58], [61]:

- self-configuration: characteristics that enable systems to adapt to changing conditions by changing their own configurations and that allow the addition and removal of components or resources without service disruption.
- self-healing: capacity to recognize and diagnose deviations from normal conditions that could cause service disruption and take action to normalize them.
- self-optimization: ability of the system to monitor its state and performance and proactively tune itself to respond to environmental stimuli.
- self-protection: incorporation of intelligence to recognize and circumvent security threats.

We are focusing in the self-configuration and self-optimization functionalities applied respectively on J2EE applications servers and Grid infrastructure.

Concerning the self-configuration functionality, in the context of the Selfware project (see 8.2), we propose FScript, a dedicated language used to program safe structural reconfigurations of Fractal architectures.

Concerning the self-optimization functionality, we tackle the problem of power management in grid computing by developing a grid-on-demand solution based on the Xen technology (machine-level virtualization). According to the resources requested (induce by transparent monitoring), we can migrate applications and systems dynamically on resources available.

5. Software

5.1. Arachne

Keywords: *AOP*, *C language*, *Dynamic system evolution*, *Proxies*.

Participants: Jean-Marc Menaud [correspondent], Rémi Douence, Mario Südholt, Nicolas Loriant.

We implemented the EAOP model into Arachne, an aspect dynamic weaver for C legacy applications. Arachne has been developed to dynamically evolve a system at runtime without causing service interruptions, for instance, in order to modify prefetching policies in Web caches or security updates in proxies.

C applications, in particular those using operating system level services, frequently comprise multiple crosscutting concerns: network protocols and security are typical examples of such concerns. While these concerns can partially be addressed during design and implementation of an application, they frequently become an issue at runtime, e.g., to avoid server downtime. For examples, a deployed network protocol might not be sufficiently efficient and may thus need to be replaced. Buffer overflows might be discovered that imply critical breaches in the security model of an application. A prefetching strategy may be required to enhance performance.

Hence, these concerns are crosscutting in the sense of AOP and aspects should therefore be a means of choice for their modularization. Such concerns have three important characteristics. First, the concerns must frequently be applied at runtime. A dynamic aspect weaver is therefore needed. Second, such concerns expose intricate relationships between execution points. The aspect system must therefore support expressive means for the definition of aspects, in particular pointcuts. Third, efficiency is crucial in the application domain we consider. To our knowledge, none of the current aspect systems for C meets these three requirements and is suitable for the modularization of such concerns.

The Arachne framework is built around two tools, an aspect compiler and a runtime weaver based on new hooking strategies derived from our previous work on MICRODYNER [78]. Arachne implements weaving by exploiting linking information to rewrite C binary executables on the fly. With this approach we can extend base program using AOP without loss of efficiency and without service interruption. Furthermore, Arachne does not need any preparation of the base program to enable aspect weaving. Finally, Arachne offers an open framework that enables aspect developers to define sets of joinpoints and develop generally applicable pointcuts.

Arachne, is presented in N. Loriant's PhD thesis [15].

5.2. Bossa

Keywords: *AOP*, *DSL*, *Linux*, *OS*, *process scheduling*.

Participants: Gilles Muller [correspondent], Julia Lawall, Christophe Augier, Richard Urunuela.

Bossa is a framework (DSL, compiler, run-time system) targeted towards easing the development of kernel process scheduling policies that address application-specific needs [71]. Bossa includes a domain-specific language (DSL) that provides high-level scheduling abstractions that simplify the implementation of scheduling policies. Bossa has been validated by reengineering the Linux kernel so that a scheduling policy can be implemented as a kernel extension.

Emerging applications, such as multimedia applications and real-time applications, have increasingly specialized scheduling requirements. Nevertheless, developing a new scheduling policy and integrating it into an existing OS is complex, because it requires understanding (often implicit) OS conventions. Bossa is a kernel-level event-based framework to facilitate the implementation and integration of new scheduling policies. Advantages of Bossa are:

- Simplified scheduler implementation: The Bossa framework includes a domain-specific language (DSL) that provides high-level scheduling abstractions that simplify the implementation and evolution of scheduling policies. A dedicated compiler checks Bossa DSL code for compatibility with the target OS and translates the code into C.
- Simplified scheduler integration: The framework replaces scheduling code scattered throughout the kernel by a fixed interface made up of scheduling events. Integration of a new policy amounts to linking a module defining handlers for these events with the kernel.
- Safety: Because integration of a new policy does not require any changes to a Bossa-ready kernel, potential errors are limited to the policy definition itself. Constraints on the Bossa DSL, such as the absence of pointers and the impossibility of defining infinite loops, and the verifications performed by the Bossa DSL compiler provide further safety guarantees.

Concretely, a complete Bossa kernel comprises three parts:

- A standard kernel, in which all scheduling actions are replaced by Bossa event notifications. The process of re-engineering a kernel for use with Bossa can be almost fully automated using AOP.
- Programmer-provided scheduling policies that define event handlers for each possible Bossa event. Policies can be structured in a hierarchy so as to provide application-specific scheduling behavior.
- An OS-independent run-time system that manages the interaction between the rest of the kernel and the scheduling policy.

Bossa is publicly available at http://www.emn.fr/x-info/bossa for both Linux 2.4 and Linux 2.6. When evaluating the performance of Bossa compared to the original Linux kernel on real applications such as kernel compilation or multimedia applications, no overhead has been observed. Finally, Bossa is currently used at EMN, ENSEIRB and the University of Lille for teaching scheduling.

Bossa was initially developed in the context of a research contract between France Télécom R&D and INRIA's Compose project. It is developed jointly by EMN and the University of Copenhagen (DIKU). We are applying the Bossa approach to a virtual machine monitor (a.k.a nano-kernel) that simultaneously supports multiple OSes on a single machine through a CIFRE grant with VirtualLogix (formerly Jaluna). Additionally, we are extending Bossa to support energy management through a grant of the *Pays de la Loire* council for the Ph.D. of Richard Urunuela.

5.3. FScript

Keywords: *Fractal*, *autonomic computing*, *dynamic reconfiguration*, *self-adaptive components*.

Participants: Thomas Ledoux [correspondent], Pierre-Charles David, Marc Léger.

FScript is a scripting language dedicated to architectural reconfigurations of systems built from Fractal components available at http://fractal.objectweb.org/fscript. FScript provides a special notation called FPath to navigate intuitively inside an architecture and select parts of it. More precisely:

1. FPath is a DSL for querying Fractal architectures. Its domain is restricted to the introspection of architectures, navigating inside them to locate elements of interest by their properties or location in the architecture. This focused domain allows FPath to offer very concise yet powerful and readable notations.
2. FScript is a scripting language that allows for the definition of complex reconfigurations of Fractal architectures, and nothing else [37]. FScript integrates FPath seamlessly in its syntax, FPath queries being used to select the elements to reconfigure. The restricted power of the language ensures that FScript programs can only talk about Fractal architectures, and can not execute "dangerous" and difficult to control code constructs (infinite loops, IO, etc.) as would be the case in a general-purpose scripting language. To ensure the reliability of its reconfigurations, FScript considers them as transactions, with all the usual ACID properties (adapted to our context). The FScript interpreter integrates a back-end system which implements this transactional semantics on top of the Fractal model [30].

Both languages can be used either from Java through a simple API to interact with their interpreters, or using a shell-like console for interactive exploration and interaction.

5.4. Reflex

Keywords: *AOP*, *Java*, *Javassist*, *metaobject protocol*, *reflection*.

Participants: Jacques Noyé [correspondent], Angel Núñez.

Reflex was initially conceived as a open and portable reflective extension of Java (and can still be used as such) but later evolved into a kernel for multi-language AOP. It provides, in the context of Java, building blocks for facilitating the implementation of different aspect-oriented languages so that it is easier to experiment with new AOP concepts and languages, and also possible to compose aspects written in different AOP languages. It is built around a flexible intermediate model, derived from reflection, of (point)cuts, links, and metaobjects, to be used as an intermediate target for the implementation of aspect-oriented languages. This is the level at which aspect interactions can be detected and resolved. Below this composition layer, a reflection layer implements the intermediate reflective model. Above the composition layer, a language layer, structured as a plugin architecture, helps bridge the gap between the aspect models and the intermediate model. In order to be portable, Reflex is implemented as a Java class library. It relies on Javassist to weave hooks in the base bytecode at load-time and connect these hooks to the metalevel, or to add structural elements (methods, classes) according to a Reflex configuration program (which has first to be generated, for each aspect, by the corresponding plugin). Part of this configuration can be modified at runtime through a dynamic configuration API.

Load-time configuration makes it possible to limit program transformation to the program points of interest (partial reflection with spatial selection). Runtime configuration makes it possible to activate/deactivate the hooks (partial reflection with temporal selection).

An important property of Reflex is that the MOP of its underlying reflective layer is not fixed but can also be configured. This makes it possible to configure Reflex in order to support efficient static weaving but also makes it possible to support dynamic weaving (although a minimal overhead at the level of the hooks cannot be avoided after unweaving).

A prototype of Reflex [17] is available at http://www.emn.fr/x-info/reflex. Reflex has been used for teaching reflection and aspect-oriented programming at the Master level within our EMOOSE and ALD curricula (see Section 9.2) as well as at the University of Chile. It is developed jointly by EMN and the University of Chile.

5.5. Awed

Keywords: *AOP*, *distributed programming*, *invasive patterns*.

Participants: Mario Südholt [correspondent], Luis Daniel Benavides Navarro.

The Aspects With Explicit Distribution model (AWED, http://www.emn.fr/x-info/awed) supports the modularization of crosscutting functionalities of distributed applications. It addresses the problem that common aspect systems do not provide features for distributed programming. It features notably three main aspect abstractions: remote pointcuts, remotely-executed advice and distributed aspects (see 6.2). It can therefore be seen as an instance of our model of Event-based AOP to distributed programming.

AWED has been fully implemented. A particularly interesting feature of this implementation is that some of the core features of the language model, such as sharing of aspect-internal state, has been implemented using AWED aspects themselves. Furthermore, it features support for remote pointcuts through a broadcast scheme of remote events. Recently, we have extended the AWED implementation by support for synchronization via transparent futures and the passing of remote references. The implementation has been realized on top of the JAsCo dynamic weaver for Java and is now part of the regular JAsCo distribution http://ssel.vub.ac.be/jasco.

Finally, AWED has been used to define *invasive distributed patterns*, a new notion of patterns supporting the compositional construction of large-scale distributed applications. The AWED system has been applied to replicated caching infrastructures of distributed component platforms and web services (see 5.5). We have shown, in particular, how to flexibly distribute a sequential implementation of a satellite-based toll system as part of a contract with Siemens AG, Munich.

5.6. Baton

Keywords: *AOP*, *FSP*, *Java*, *LTS*, *Reflex*, *concurrency*.

Participants: Jacques Noyé [correspondent], Rémi Douence, Angel Núñez.

Baton is a prototype implementation of CEAOP. CEAOP models (stateful) aspects as well as base program components as Finite State Processes (FSP) (see also section 6.2) and makes it possible to compose them in various ways using composition operators. Using Baton, it is possible to:

- Define aspects using an FSP-like syntax.
- Relate these aspects to a base program by mapping aspect events into AspectJ-like pointcuts (defining the events of interest in the base program) and actions into Java methods (implemented by the base program).
- Compose the aspects and the base program using predefined operators.

The base program can be given as standard Java bytecode. The compilation of a Baton program (using Stratego http://www.program-transformation.org/Stratego/) generates Java code for the aspects, their instantiation and their composition. It also generates a Reflex configuration responsible for instrumenting the base program at load time, weaving in proper synchronization code.

This prototype is publicly available at http://www.emn.fr/x-info/baton.

6. New Results

6.1. Components

Keywords: *adaptation*, *communications*, *components*, *composition*, *interaction*, *objects*, *protocols*, *services*.

Participants: Jacques Noyé, Jean-Claude Royer, Rémi Douence, Thomas Ledoux, Mario Südholt, Pierre-Charles David, Hugo Arboleda, Fabien Baligand, Luis Daniel Benavides Navarro, Fabricio Fernandes, Marc Léger.

Our results on components have focused on four main activities providing support for component with protocols, verification, adaptation and applications. Some of these results strongly link OBASCO's work on CBSE with that on AOP by means of modifications to component interaction protocols that are expressed in terms of aspects. Furthermore, they have been partially integrated into three publicly available software systems, in particular the AWED prototype for aspects with explicit distribution (see Sec. 5.5), the FScript scripting language for reconfiguration of Fractal components (see Sec. 5.3). Finally, they have also been applied to two of OBASCO's main application domains: distributed operating systems and enterprise information systems, in particular, web services.

6.1.1. Support for Explicit Protocols

The integration of protocols into components has been studied with two different models respectively, labelled transition system, and symbolic transition system.

6.1.1.1. Extending Components with Aspects using Protocols

Once components equipped with explicit protocols represented as labeled transition systems, one may want to alter the behavior of such components based on their protocols. It is then natural to specify these alterations using stateful aspects, which can also be modeled as labeled transition systems. We have considered a simple component model closed to Darwin and shown how such a model can be seamlessly extended with *aspect components* [35].

We use the term *aspect component* rather than *aspect* to capture the fact that, as a component, an aspect is made of both an implementation and an explicit interface. In order to deal with invasive component modifications without breaking modularity, we have also extended the interface of a component interface with an *action interface*. This interface, inspired by the Open Modules of Jonathan Aldrich, specifies the internal actions that can be advised.

The protocol of an aspect component observes the service requests and replies, as well as the internal actions of other components, and react to these actions, possibly preventing these actions to happen as is standard with AOP. A nice feature of the model is that an assembly of plain and aspect components can be transformed into and assembly of plain components. All this is done without breaking the black-box nature of the components.

6.1.1.2. Java Implementation of Rendezvous for Symbolic Transition Systems

A crucial issue is to fill the gap between high-level models, needed for design and verification, and implementation of components. Thus we investigated a Java runtime support for our component model with explicit protocols based on symbolic transition systems (STS) [40]. The principles are: i) primitive components are defined gluing a protocol part, and a passive Java class inside a thread, ii) composite components are made of concurrent subcomponents interacting thanks to a rendezvous mechanism. This mechanism is a one way n-ary rendezvous which allows to synchronize STS, each STS can trigger a proper action, it supports guards and receipts on guards. Thus the delicate thing is to implement this complex rendezvous especially the guard with receipt feature. This relies on two synchronization barriers which are implemented using the Java monitor facility. This solution is general in the sense that we do not constrain the ordering of processes to enter, execute their action and leave the synchronization section. We also propose a first optimization to allow several independent synchronizations to process the barrier. This is a first way to distribute the central arbiter mechanism used to synchronize the components. Currently, this work provides an operational interpreter to program primitive components in Java with STSs and a powerful way to compose them.

6.1.2. Support for Component Verification

Component with protocols raise the issue of verification, which was experimented in two different contexts.

6.1.2.1. Ensuring composition properties using VPA-based protocols

In the context the PhD thesis of Dong Ha Nguyen we have investigated protocols that restrict how components may interact. Concretely, we have studied the properties of such protocols defined in terms of visibly pushdown automata. Such protocols are strictly more expressive than the commonly used regular protocols but obey all the same closure properties (as opposed to context-free languages). Hence, they enable certain well-balanced context of arbitrary depth to be used in protocols and basic properties, such as component composability and substitutability, to be easily checked.

We have recently extended the language for the definition of VPA-based protocols [33], [32] and investigated how properties of evolving software components can be formally specified and (semi-automatically) be verified if evolutions are defined using VPA-based aspects. Concretely, we have introduced operators geared towards the definition of evolutions of software components in the language and have defined classes of evolution problems that can be proved in terms of evolution aspects only or evolution aspects and properties or base classes (as opposed to proofs over woven applications that are typically much more tedious and costly).

6.1.2.2. Runtime and Library Support for Formal Components

The STSLIB library aims at providing a framework to define formal components [43]. The general objective is to define a Java tool support allowing the formal design of software components and their execution. Our component descriptions are based on symbolic transition systems and currently implements a strict subset of the Korrigan architectural description model [73]. We need to connect this formalism with usual verification means (general prover and model-checkers) but we also expect to develop complementary ways to check the specifications. We aim at a Java code translation which would be as automated as possible, furthermore a real runtime support not only a specification simulator. We develop a specific way to check the components based on an extension of the synchronous product and the configuration graph computation. This was applied to a cash point case study and we got some results related to our verification experiments. Currently we are also defining a Java interpreter to execute the component descriptions based on the Java implementation of our STS synchronization mechanism.

6.1.3. Support for Component Adaptation

This year we have explored and defined a large range of results concerning support for the adaptation of software components, in particular, concerning the adaptation of specific component platforms, such as Enterprise JavaBeans and the Fractal component model, but also concerning the adaptation of whole classes of component-based applications such as web service based component systems.

6.1.3.1. Invasive Patterns to Address Modularization Problems in CBSE

Component-based systems are frequently subject to modularization problems that hinder their evolution. We have studied this problem for the replication and transaction functionality of JBoss Cache, a caching infrastructure of EJB-based software components. Previous studies have clearly shown strong dependencies between the two functionalities that result in important crosscutting behavior within JBoss Cache and similar infrastructures.

This year we have defined a notion of *invasive patterns* [23], [24] that provides novel language support for distributed programming allowing, in particular, to modularize such crosscutting behaviors and to easily define and refer to large-scale topologies on top of which such infrastructures and applications are frequently built upon.

We have shown, in particular, how invasive patterns can be used to perform a significant restructuring of JBoss Cache in order to obtain a module structure on the code level that is equivalent to high-level pattern-based architecture of the main data-flow governing the replication and transaction functionalities of JBoss Cache. This restructuring of a code base of approximately 16 KLOC has been performed resulting in a clean modularization using only 2 aspects and circa 150 LOC. [23].

6.1.3.2. Safe Dynamic Reconfigurations of Fractal Architectures

Component-based systems must support dynamic reconfigurations to adapt to their execution context, but not at the cost of reliability. Fractal provides intrinsic support for dynamic reconfiguration, but its definition in terms of low-level APIs make it complex to write reconfigurations and to ensure their reliability. We propose a language-based approach to solve these issues: direct and focused language support for architecture navigation and reconfiguration make it easier both to write the reconfigurations and to ensure their reliability. Concretely, we develop two languages: (i) FPath, a DSL which provides a concise yet powerful notation to navigate inside and query Fractal architectures, and (ii) FScript [37], a scripting language which embeds FPath and supports the definition of complex reconfigurations. FScript ensures the reliability of these reconfigurations thanks to sophisticated run-time control, which provides transactional semantics (ACID properties) to the reconfigurations [30].

6.1.3.3. A Declarative Approach for QoS-Aware in Web Service Orchestrations

A Web Service is a component accessible over the Web that aims to achieve loose coupling between heterogeneous platforms. When composing Web services, architects encounter several issues dealing with Quality of Service (QoS): (i) how to guarantee global QoS of the assembly; (ii) how to adapt a composition of Web Services to a specific context. In [20], [21], we propose a declarative approach aiming to provide the architect with adequate means to specify QoS requirements in Web Service orchestrations. To this end, we design a language, named QoSL4BP (Quality of Service Language for Business Process) that abstracts QoS concerns from the low-level details of Web Services compositions. Additionally, we propose a tool, named ORQOS (ORchestration Quality Of Service) that interprets QoSL4BP and that produces an orchestration enhanced with QoS concerns. ORQOS is a non-intrusive platform and performs QoS adaptation both at pre-deployment time and at runtime.

6.1.4. Application to Software Product Lines

In order to apply our results in the STREP AMPLE project we are investigating a model-driven configuration process for software product lines, this was done in collaboration with Rubby Casallas from Los Andes University (Bogota, Columbia).

In a first attempt we compared two implementations of a MDA approach for managing variability in a software product line [18]. The implementations correspond to two representative frameworks based on the Model Driven Engineering (MDE) principles. These frameworks are the Graphical Modeling Framework (GMF) and the Generic Model Environment (GME). We built the core assets of the product line and we generated applications using the two different frameworks. The core assets that we built are: feature models, metamodels, mapping models, and three different types of transformation rules. We built the transformation rules using two different languages: the ATLAS Transformation Language (ATL) in the context of GMF and, the Embedded Constraint Language (ECL) in GME. In a second step we are developing an approach to create Model-Driven Software Product Lines by means of successive model refinement, guided by configuration of features [19]. Each refinement uses model-to-model transformation until arriving at the executable code with technological platform details included. During this process, users select features at each stage taking into account their preferences and requirements. The selection of features can be performed for each element of the model. Thus, the selection is constrained by many facts, for example, a mandatory selection element-feature because some structural model relationships that has to be preserved. To deal with model transformations while satisfying the constraints, we introduce the concept of constraint-model to restraint the possible feature configurations a user can specify. Then, we propose the construction of transformations by composing several rules that facilitate, from a single source, the generation of different targets according to a given feature configuration.

6.2. Aspects

Keywords: *aspectualization*, *concurrency*, *distribution*, *expressive aspect languages*, *properties*, *semantics*.

Participants: Mario Südholt, Pierre Cointe, Rémi Douence, Jean-Marc Menaud, Jacques Noyé, Ali Assaf, Luis Daniel Benavides Navarro, Simon Denier, Simplice Djoko Djoko, Nicolas Loriant, Dong Ha Nguyen, Angel Núñez.

OBASCO's work on aspect-oriented programming is targeted towards the development of, support for, and application of expressive aspect languages. Recently, we have mainly pursued work on instantiations of the model of Event-Based AOP for distributed and concurrent applications, which allows expressive aspects to be defined in terms of relations over sequences of execution events in distributed and concurrent systems. The resulting two aspect systems fill in an important blank of AOP. Until now almost all approaches to AOP for distributed and concurrent systems use sequential aspect systems to manipulate distributed or concurrent object-oriented frameworks. Such approaches do not allow to express crosscutting concerns to be modularized directly in terms of abstractions relevant to the domains of distribution and concurrency. In contrast, our systems allow to express distribution and concurrency issues directly at the aspect level.

Furthermore, we have introduced different expressive aspect languages for sequential systems featuring non-regular, logic-based, or sequence-based pointcuts. These aspect languages are geared towards enhancement of the expressiveness of aspects for general-purpose imperative and object-oriented base languages, in particular for Java and the modularization of concerns of C/C++ languages by means of the Arachne system.

We have also continued our work on the foundations of AOP by developing a comprehensive semantic basis for arbitrary aspect languages. Moreover, this semantic basis can be used to define aspect languages for a large range of base programming languages that belong to different programming paradigm (such as object-oriented and functional programming languages).

Finally, these formal results have been implemented as part of different publicly available software systems: the AWED prototype for aspects with explicit distribution (see Sec. 5.5), the BATON prototype that implements concurrent aspects in the CEAOP sense (see Sec. 5.6), and a new version of the Arachne weaver for aspects in C/C++ (see Sec. 5.1).

6.2.1. Formal models for AOP

We have further developed two approaches, Concurrent EAOP and VPA-based aspects, that are based on the formal properties of Event-based AOP, that is triggering actions on the occurrence of sequences of related events. The expressive power of EAOP makes it possible to reason about events patterns, thus supporting (temporal) reasoning over AO programs. Furthermore, we have developed a comprehensive approach for the definition of the semantics of aspect languages, the Common Aspect Semantics Base (CASB) and started work on a type system for advice in the presence of generic data structures for AspectJ-like languages.

6.2.1.1. Concurrent EAOP

The initial model of EAOP relied on a monolithic entity, the monitor, which observes the execution of the base program and executes the actions associated to the matching pointcut. This model favored a sequential point of view. However, crosscutting concerns are also present in numerous concurrent applications.

Then we developed a model for concurrent aspects, the model of *Concurrent EAOP (CEAOP)*. CEAOP uses Finite State Processes (FSP) and their representation as Labeled Transition Systems (LTS) for modeling aspects, base programs and their concurrent composition, thus enabling the use of the Labeled Transition System Analyzer (LTSA) for formal property verification. The initial work on CEAOP did not provide an implementation of its concepts, restricting the study of concurrent aspects to the study of a model. We have worked on the implementation of CEAOP as a Domain-Specific Aspect Language (DSA), which we called Baton [34].

Baton is very close to FSP, and can be easily compiled into Java. As an intermediate layer, we have developed a Java library which makes it possible to associate a Java implementation to a finite state process. The compilation process consists of translating both the Baton aspects and the Java base program into Java finite state processes. This translation relies on Metaborg/SDF to extend Java with Baton and Reflex to instrument the base program.

6.2.1.2. VPA-based Aspects

A second instantiation of the EAOP model features more expressive sequence pointcuts, the so-called *VPA-based aspects*, which are based on visibly pushdown automata (VPA). These automata define a class of languages that is a proper superset of regular languages and a proper subset of context-free languages.

Concretely, VPAs allow to match certain well-balanced context (without fixing the maximum depth of such expressions, as required by regular expressions). Since VPA preserve all common closure properties of regular languages, they are amenable to analysis techniques similar to those available in the regular case. We have defined an aspect language featuring VPA-based pointcuts and shown how to analyze interactions among such aspects.

In recent work, we have extended our aspect language by additional operators for the manipulation of non-regular interaction protocols [33], [32]. The pointcut operators provide more flexible means for the matching of non-regular sequences, e.g., by excluding certain events from being interleaved between two other matched events. An advice operator enables open matching contexts to be closed, p.ex. for error handling purposes. Furthermore, we have shown that this aspect language permits to define common evolutions of software components and formally proof their correctness, see Section 6.1.

6.2.1.3. Common Aspect Semantics Base (CASB) and Preserved Properties

As part of the European Network of Excellence in AOSD we have defined a common aspect semantics base (CASB) as a small step semantics that allows the modular introduction of formal semantic descriptions of different aspect mechanisms. The semantics only relies on minimal requirements on the base language semantics and can therefore be specialized to arbitrary base language paradigms. We have shown how to define general aspect mechanisms and covered features from different aspect languages, such as our EAOP model or the CaesarJ language developed at TU Darmstadt. As an illustration of our technique, we have described the semantics of an AspectJ-like core aspect language for a core Java language. Finally, we have formally proved the correctness of aspects transformations using this semantics [42]. These transformations implement an optimization: they replace dynamic pointcuts that require a test at run time by static pointcuts that can be inlined at compile time.

We used the CASB to prove some aspect categories preserve some classes of properties [29]. Indeed, AOP can arbitrarily distort the semantics of programs. In particular, weaving can invalidate crucial safety and liveness properties of the base program. We have identified categories of aspects (observers, aborters, confiners...) that preserve some classes of properties defined as subsets of the temporal logic LTL. It is then sufficient to check that an aspect belongs to a specific category to know which properties will remain satisfied by woven programs.

6.2.2. Language Design

6.2.2.1. Type Systems for Aspect Languages

It is well-known that AspectJ's type system is unsafe in certain (common) use-cases for around-advice that uses the proceed statement. Such problems have first noted for non-generic AspectJ (and Java) by Wand et al. [82] and the resulting severe restrictions in generic contexts have been analyzed and partially resolved by Jagadeesan et al. [59].

We have performed a complete analysis of the typing problems involving non/generic around-advice with proceed that is used in non/generic base contexts. More importantly, we have provided the first comprehensive solution to this problem in form of a type system that enables such advice to be typed safely in all contexts [28]. This type system is based on the notion of a *separate type signature for proceed* and *type ranges* that allow to precisely characterize the contexts in which advice can be correctly applied. The type system is type-safe in that it enjoys type preservation and progress properties. Furthermore, we have shown how to practically integrate it in two compilation and execution environments typically used for AspectJ-like aspect languages: (i) integration into the AspectJ language by an extension to the abc aspect compiler generation system and (ii) type-safe integration of framework-based approaches to AOP, such as Spring AOP, that do not use a dedicated aspect language. The complete type system and the corresponding soundness and progress proofs have been published as a companion technical report [41].

6.2.2.2. Aspects With Explicit Distribution (AWED)

Distributed applications abound of crosscutting concerns, e.g., persistence and transactions in component-based enterprise information systems. As part of Daniel Benavides's on-going PhD thesis, we have defined an aspect language for explicit distributed programming. Starting from the evaluation of crosscutting functionalities in the framework for distributed caching JBoss Cache, we have defined an aspect language named “Aspects with Explicit Distribution” (AWED) allowing the modularization of these functionalities using explicit references to hosts on which caches are executed. Concretely, we have realized three contributions: remote sequence pointcuts, remotely synchronously or asynchronously executed advice, and distributed aspects with corresponding deployment, instantiation and data sharing mechanisms.

We have shown that this language can serve as a basis for a new general notion of invasive patterns for distributed programming [23] and have applied AWED to the distribution of an industrial case study, an automatic road toll system [22].

6.2.3. Aspects in Practice

6.2.3.1. Design Pattern Aspectualization : AspectJ and JHotDraw

Design patterns are a powerful means to understand, model and implement OO micro-architectures. Pattern solutions, which we called *motifs*, appear with an increasing density in libraries and frameworks. Their composition leads to architectures with interesting properties in terms of variability and evolvability. However, patterns are difficult to track, modularize and reuse as their constituting elements tend not to be explicit anymore in source code. Following the two PhD theses of H. Albin and Y-G. Guéhéneuc dedicated to the reification and analysis of design patterns, we have experimented AOP modularization technology to deal with scattering and tangling of motif implementations.

S. Denier’s thesis addresses the problems of density, implementation and composition of motifs with AspectJ[14]. Based on a case study with the JHotDraw framework, this thesis illustrates how a high density of motifs weakens modularity and adaptation of code. Denier presents transformation of motifs with the help of aspects and describes AspectJ idioms supporting their modularization. He inspects modularity and reuse of motifs composition defined with aspects. He demonstrates how aspects crosscutting languages help solving motifs interactions. Finally, he develops a programming model for AspectJ based on the joint use of classes and aspects. This thesis enlightens how aspects facilitate studying and handling a high density of motifs. Moreover, this work opens perspectives for the improvement of aspects languages.

6.2.3.2. Invasive Patterns for Distributed System

We have introduced the new notion of *invasive patterns for distributed programming* [23], [24] that augment standard communication and computation patterns patterns, such as farming out computations, with two contributions: (i) means for the modularization of invasive accesses to execution state that are needed to enable the application of the communication patterns and (ii) expressive topology definitions for the quantification over large sets of hosts involved in a pattern application. We have defined the notion of invasive patterns by developing some of the abstractions introduced in the AWED language and shown that invasive patterns can be efficiently implemented using AWED. More information on invasive patterns and their validation in the context of the J2EE platform can be found in Section 6.1.

6.2.3.3. Aspects for C Applications

We have pursued different approaches for expressive aspect languages in the context of system-level applications. We have integrated EAOP-based techniques into Arachne, our aspect system for dynamic weaving of aspects into C applications (see Section 5.1).

Finding the root cause of bugs and performance problems in large applications is a difficult task. The main reason for this difficulty is that the understanding of such applications crosscuts the boundaries of a single process: the concurrent nature of large applications requires insights into multiple threads and process and even sometimes of the kernel. Currently, most existing tools lack support for simultaneous kernel and applications analysis. In order to address this point, we have extend Arachne. With this extension, it enables safe runtime injection of probes inside the Linux kernel and user space applications at both occurrences of function calls and variable accesses. It features an Aspect-Oriented language that permits access to execution contexts and to compose primitive probes (for example sequences of function calls). We show in [31] how Arachne allows to easily analyze problems such as race conditions which involve complex interactions between multiple processes, how Arachne is fast enough to analyze high performance applications such as the Squid web cache.

6.2.3.4. Distribution of an Automatic Toll System

The AWED language and system support the distribution of sequential programs in a simple manner because an aspect may define and/or modify the behavior of many distributed or to-be-distributed entities at once.

In the context of the ATOLL project with Siemens AG, Munich, Germany, we have validated this claim by applying AWED to the distribution of a sequential version of an automatic toll system, a medium-sized abstraction of an industrial-strength tolling system [22]. We have been successful in providing evidence for the improvements in the architecture and implementation of the toll system compared to a pure OO solution. Furthermore, AWED has proven valuable in permitting to flexibly distribute the sequential application, without any changes to the base code, between a server backend that performs accounting tasks and the on-board unit that are hosted in the trucks.

6.3. DSLs

Keywords: *Linux*, *OS*, *device drivers*, *energy management*, *process schedulers*, *software evolution*.

Participants: Gilles Muller, Julia Lawall, Yoann Padioleau, Christophe Augier, Richard Urunuela.

6.3.1. Process scheduling and energy management

The Bossa framework (DSL, compiler, run-time system) has been publicly available for 5 years and is used in several universities for both research and teaching. It is fully compatible with the Linux 2.4 and 2.6 kernels and can be used as a direct replacement. Our recent work on Bossa, is on its extension to energy management. This led us to design CPU frequency adaptation policies for improving energy management in the context of a video player in embedded systems [39].

6.3.2. Automating Collateral Evolutions in Linux Drivers

In a modern OS, device drivers can make up over 70% of the source code. Driver code is also heavily dependent on the rest of the OS, for functions and data structure defined in the kernel and driver support libraries. These two properties together pose a significant problem for OS evolution, as any changes in the interfaces exported by the kernel and driver support libraries can trigger a large number of adjustments in dependent drivers. These adjustments, which we refer to as collateral evolutions, may be complex, entailing substantial code reorganizations. Collateral evolution of device drivers is thus time consuming and error prone. We have previously performed a qualitative and quantitative assessment of the collateral evolution problem in Linux device driver code. This study has shown that from one version of Linux to the next, collateral evolutions can account for up to 35% of the lines modified in such code.

These issues clearly call for a formal means of describing collateral evolutions and automated assistance in applying them. In the context of the ANR Coccinelle project, we are developing a language-based infrastructure with the goal of documenting and automating the kinds of collateral evolutions that are required in device driver code. By comparison to patches that are the current solution for transmitting evolutions in Linux, our specifications are generic and can be applied to all files involved in a collateral evolution. Therefore, our specifications rely on the semantics of the previous and new versions of the code, that motivates the name “Semantic Patches” for our specifications.

We have designed the SmPL[2] DSL [36] for writing semantic patches and a prototype tool for applying them to Linux device drivers. We have evaluated our approach on over 60 representative collateral evolutions that were previously performed manually in Linux 2.5 and 2.6. On a test suite of over 5800 relevant driver files, the semantic patches for these collateral evolutions update over 93cases, the user is typically alerted to a partial match against the driver code, identifying the files that must be considered manually. We have additionally identified over 150 driver files where the maintainer made an error in performing the collateral evolution, but Coccinelle transforms the code correctly. Finally, several patches derived from the use of Coccinelle have been accepted into the Linux kernel.

[2]SmPL is the acronym for Semantic Patch Language and is pronounced either “sample” or “simple”

A first release of the Cocinelle tool is now available publicly from http://www.emn.fr/x-info/coccinelle. Coccinelle has been presented at the OLS *Ottawa Linux Symposium* in June and at the STIC 2007 symposium in November in Paris. Finally, we are currently investigating the usage of Smpl for introducing workaround code to help bug detection and diagnosis [38].

6.3.3. Zebu: a DSL for Improving the Robustness of Network Application Protocols

The secure and robust functioning of a network relies on the defect-free implementation of network applications. As network protocols have become increasingly complex, however, hand-writing network message processing code has become increasingly error-prone.

In collaboration with the INRIA Phoenix project team, we have designed a domain-specific language, Zebu, for describing protocol message formats and related processing constraints. From a Zebu specification, a compiler automatically generates stubs to be used by an application to parse network messages. Zebu is easy to use, as it builds on notations used in RFCs to describe protocol grammars. Zebu-based applications are also efficient, as the memory usage is tailored to application needs and message fragments can be specified to be processed on demand. Finally, Zebu-based applications are robust, as the Zebu compiler automatically checks specification consistency and generates parsing stubs that include validation of the message structure. Using a message torture suite in the context of SIP and RTSP, we show that Zebu-generated code reliably accepts valid messages and rejects invalid ones. Results of experiments on a real trace for SIP show that Zebu-generated code has performance comparable to that of most existing parsers for this protocol, including that of the widely used SER server [25].

7. Contracts and Grants with Industry

7.1. Siemens AG, Munich

Participants: Mario Südholt, Jean-Marc Menaud, Luis Daniel Benavides Navarro.

In the context of the *Aspects for Toll Systems (ATOLL)* contract, we have explored the use of aspects with explicit distribution (AWED, described in 6.2, 5.5) for the restructuring and dynamic adaptation of client-server based satellite-guided toll systems. Siemens AG from Munich, Germany, is in particular interested whether aspects enable functionality between the central servers and toll units installed on tracks to be shifted dynamically. Siemens supports this project by 10 KEUR, which has resulted in a flexible distribution strategy for a sequential toll implementation that has been integrated in the main industrial demonstrator of the European Network of Excellence in AOSD, AOSD-Europe.

7.2. VirtualLogix Cifre grant

Participants: Gilles Muller, Christophe Augier.

Our work on the development of Bossa (see Section 5.2) is supported in part by VirtualLogix (formerly Jaluna) in the context of the PhD of Christophe Augier, in particular, through a supervision fee of 12 KEUR per year. The goal of this work is to apply the Bossa approach to a virtual machine monitor (a.k.a nano-kernel) that simultaneously supports multiple OSes on a single machine.

7.3. France Télécom R&D PhD about Web Services

Participants: Thomas Ledoux, Fabien Baligand.

Web Services and Service-Oriented Architectures aim to deliver agile service infrastructures. While BPEL language has emerged to allow the specification of Web Services compositions from a functional point of view, it is still left to the architects to find proper means to handle the Quality of Service (QoS) concerns of their compositions.

F. Baligand's PhD work aims to overcome this shortcoming by introducing both a new language and a platform for QoS specification and implementation in service compositions. More specifically, we propose a policy-based language aiming to provide expressivity for QoS behavioural logic specification (e.g. QoS constraints processing, QoS mechanisms injection) in Web Service orchestrations, as well as a non-intrusive platform in charge of its execution both at pre-deployment time and at runtime.

This work is supported by France Telecom R&D (MAPS/AMS) amounting to 22.5 KEUR.

7.4. France Télécom R&D PhD about Fractal Architectures

Participants: Thomas Ledoux, Marc Léger.

Dynamic reconfigurations in component-based software applications are central to promising approaches like autonomic computing. M. Leger's PhD work aims to ensure the reliability of dynamic reconfigurations in the Fractal component model (but it can be generalized to other models), and can be applied to concurrent and distributed reconfigurations. We will show how ACID properties in the context of reconfigurations can improve reliability by making systems fault-tolerant i.e., compliant with the specification in spite of faults due to dynamic reconfigurations. The results will be used in the Selfware national project (see further 8.2) as a self-healing foundation of autonomic distributed applications.

This work is supported by France Telecom R&D (MAPS/AMS) amounting to 21 KEUR.

8. Other Grants and Activities

8.1. Regional Actions

8.1.1. MILES project / Software Engineering Cluster

This new two years project funded by the *Pays de la Loire Council* aims at applying models engineering, aspect oriented programming and domain specific languages to the field of real time systems. This is a join work with LINA and IRCCyN teams: ATLAS, COLOSS, MODAL and STR. See also the FLFS ANR related project in section 8.2.

P. Cointe is the cluster coordinator and OBASCO funding is 3 KEUR and half a PhD thesis grant.

8.1.2. Arantèle project

Participants: Jean-Marc Menaud, Pierre Cointe, Nicolas Loriant.

The Arantèle project is also funded by the *Pays de la Loire* council. It started in October 2004 for 30 months and a budget of 78 KEUR. Its objective is to explore and design new development tools for programming computational grids. These grids promise to be the next generation of high-performance computing resources and programming such computing infrastructures will be extremely challenging.

This project addresses the design of an aspect-based software infrastructure for computational grids based on our EAOP model and our current prototype of Arachne (see section 5.1). This work is done in close collaboration with Subatech (IN2P3) and the CERN since our future experiments and evaluations will focus on a legacy application : AliRoot, the main software used by physicians in their ALICE experiment. This work is also the first opportunity to collaborate with the PARIS project-team (C. Perez) on using AOP to design software components for Grid computing [68].

8.2. National Projects

8.2.1. ARC INRIA VeriTLA+

Participants: Gilles Muller, Julia Lawall.

The goal of this project is to develop a verification environment for TLA+, a specification language for distributed algorithms and reactive systems. Our aim is to validate this environment in the context of telecommunication services and OS kernel. We are mainly interested in formalizing and verifying Bossa properties using TLA+.

Our partners are the INRIA project-teams Cristal, Mosel, Cassis and Phoenix. Other partners are DIKU and Microsoft Research.

8.2.2. ANR/RNTL Selfware

Participants: Thomas Ledoux, Jean-Marc Menaud, Pierre-Charles David, Nicolas Loriant.

The Selfware project is an ANR/RNTL project running for 30 months which has been submitted and accepted in 2005 for funding amounting to 222 KEUR from June 2006.

Selfware goal is to propose a software infrastructure enabling the building of distributed applications under *autonomic* administration. Historically, Autonomic Computing is an initiative started by IBM Research in 2001 where the word *autonomic* is borrowed from physiology; as a human body knows when it needs to breathe, software is being developed to enable a computer system to know when it needs to repair itself, configure itself, and so on.

In the Selfware project, we are interested by autonomic administration of computing systems which involve the following characteristics: self-configuring, self-healing and self-optimizing of distributed applications. We will focus on two types of server administration: (i) J2EE application servers with Jonas; (ii) asynchronous Message-Oriented Middleware with Joram.

Experiments will be realized in the OW2 context with the Fractal component model. The web page is: http://www.ow2.org).

The project federates work between six partners: France Télécom R&D, Bull, Scalagent, INRIA Rhône-Alpes (the SARDES project-team), IRIT-ENSEEIHT and OBASCO.

8.2.3. ANR/RNTL SADAJ

Participant: Gilles Muller.

The aim of the SADAJ project is to address technology locks that prevent to use Java in the domain of automotive real-time embedded systems. This application domain mainly uses low-cost 8-bits micro-controllers. Therefore, the challenge is to produce safe code that is both equivalent in size and speed to C programs. The precise technology contributions of the project are:

- Application-specific Java-like virtual machines developed by IST Nantes, a startup created by F. Rivard, a past OBASCO PhD student,
- Application-specific schedulers for easing the management of real-time aspects in the automotive applications. This contribution is made by the OBASCO team and relies on Bossa,
- Enforced security protection of the software in the micro-controller. This contribution is made by the micro-controller vendor, Atmel.

Two other partners, SiemensVDO and Ayrton Technology whose domains are automotive and embedded systems Développement will define the precise needs of the targeted applications and will evaluate the productivity gain of the solution.

The duration of the project is 24 months, starting December 2006. The OBASCO funding part amounts to 200 KEUR.

8.2.4. ANR non thématique Coccinelle

Participants: Gilles Muller, Julia Lawall, Yoann Padioleau.

One of the main challenges in the Linux operating system (OS) is to manage evolution. Linux is evolving rapidly to improve performance and to provide new features. This evolution, however, makes it difficult to maintain platform-specific modules such as device drivers. Indeed, an evolution in a generic OS module often triggers the need for multiple collateral evolutions in dependent device drivers. As these collateral evolutions are often poorly documented, the resulting maintenance is difficult and costly, frequently introducing errors. If a driver maintainer becomes unavailable, the driver quickly falls behind the rest of the OS.

The aim of this 3 year research project, which has been submitted and accepted in 2005 for funding amounting to 200 KEUR from January 2006 by the French ministry of research, is to propose a language-based approach to address the problem of collateral evolution in drivers. Specifically, we plan to create a development environment, Coccinelle, that provides a transformation language for precisely expressing collateral evolutions and an interactive transformation tool for applying them. The key idea of Coccinelle is to shift the burden of collateral evolution from the driver maintainer to the OS developer who performs the original OS evolution, and who thus understands this evolution best. In our vision, the OS developer first uses the Coccinelle transformation language to write a semantic patch describing the required collateral evolution in device drivers. He then uses the Coccinelle transformation tool to validate the semantic patch on the drivers in the Linux source distribution. Coccinelle will provide a means for formally documenting collateral evolutions and for easing the application of these evolutions to driver code. The primary result of this project will be the development of the Coccinelle environment. As part of the development of this environment, we will identify, classify, and implement collateral evolutions performed during the last five years. This work should lead to a more robust set of Linux drivers. More generally, our work should be helpful to companies using Linux who make specialized devices, such as in the area of consumer electronics.

Our partner is the DIKU laboratory from the University of Copenhagen.

8.2.5. ANR non thématique FLFS (Languages Family for Systems Family)

Participants: Pierre Cointe, Simon Denier, Kelly Garcés.

Traditionally, software development does not rely on an in-depth knowledge of the target domain. Instead, domain-specific knowledge is integrated in software development process in an ad hoc and partial fashion, without much formal basis or tools. In doing so, software systems are tackled in isolation, making conceptual or implementation factorization difficult. Yet, it is fundamental to observe that programs always belong to a family. In this family, they share commonalities and expose specific variations.

From a software development viewpoint, a program family represents a domain of expertise, that is, a vocabulary, notations, rules and protocols that are specific to a domain. For example, the telephony domain consists of a set of concepts, rules, protocols and interfaces that represent a precise framework to be used for the development of telephony services.

Our goal is to place domain expertise at the centre of the software development process. It is aimed to lift the current limitations of software engineering regarding large scale software production, robustness, reliability, maintenance and evolution of software components. Our key innovation is to introduce a software development process parametrized with respect to a specific domain of expertise. This process covers all the stages of software development and combines the following three emerging approaches:

- Domain-specific modelling, also known as model engineering;
- Domain-specific languages, in contrast with general-purpose languages;
- Generative programming and in particular aspect-oriented programming as a means to transform models and programs.

Our partners are the Atlas (J. Bévivin) and Phoenix (C. Consel) INRIA teams. The duration of the project is 36 months, starting December 2006. The OBASCO funding part amounts to 70 KEUR. The Web page is :http://flfs.com.fr).

8.3. European Projects

8.3.1. NoE AOSD-Europe

Participants: Pierre Cointe, Rémi Douence, Jacques Noyé, Mario Südholt, Ali Assaf, Simplice Djoko Djoko.

OBASCO participates in the European Network of Excellence in Aspect-Oriented Software Development (NoE AOSD) since September 2004. This network is meant to federate the essential part of the European research community in AOSD over 4 years. The network is coordinated by Lancaster University (UK) and includes 10 other partners: Technische Univ. Darmstadt (Germany), Univ. of Twente (The Netherlands), INRIA (project teams OBASCO, JACQUARD, TRISKELL and POP-ART), Vrije Univ. Brussels (Belgium), Trinity College Dublin (Irland), Univ. of Malaga (Spain), KU Leuven (Belgium), Technion (Israel), Siemens (Germany) and IBM UK.

With regard to technical integration work, the network is structured in four “laboratories:” a Language Lab, a Formal Methods Lab, an Analysis and Design Lab and an Applications Lab. OBASCO essentially takes part in the first two labs whose main goal is a comprehensive meta-model and correspond implementation platform for the Language Lab as well as a comprehensive semantic model and corresponding proof/analysis tools for the Formal Methods Lab. Furthermore, OBASCO coordinates the work of the four participating INRIA groups including Jacquard (Lille), Triskell (Rennes) and PopArt (Grenoble).

Overall funding of the network by the EU is 4.4 MEUR. OBASCO’s share amounts to 200 KEUR. The web page is: http://www.aosd-europe.net.

This year we have mainly worked on an industrial-strength demonstrator showing the flexible distribution of a sequential satellite-based toll system implementation. Furthermore, we have developed new means for the formal definition of aspects and the formal manipulation of components using aspects that modify component interaction protocols. Finally, we have worked on the metamodel-based definition and classification of aspect languages.

8.3.2. STREP AMPLE

Participants: Jean-Claude Royer, Pierre Cointe, Jacques Noyé, Mario Südholt, Joost Noppen, Hugo Arboleda, Fabricio Fernandes, Angel Nũnez.

The Aspect-Oriented, Model-Driven Product Line Engineering (AMPLE) project started on October 1, 2006 and will finish September 30, 2009. It involves the following partners: University of Lancaster (UK), Universidade Nova de Lisboa (Po), École des Mines de Nantes (Fr), Darmstadt University (De), Universiteit Twente (Nl), Universidad de Màlaga (Es), HOLOS (Po), SAP AG (De), Siemens (De).

The aim of this project is to provide a Software Product Line (SPL) development methodology that offers improved modularisation of variations, their holistic treatment across the software lifecycle and maintenance of their (forward and backward) traceability during SPL evolution. Aspect-Oriented Software Development (AOSD) can improve the way in which software is modularised, localising its variability in independent aspects as well as improving the definition of complex configuration logic to customise SPLs. Model-Driven Development (MDD) can help to express concerns as a set of models without technical details and support traceability of the high-level requirements and variations through model transformations.

AMPLE will combine AOSD and MDD techniques to not only address variability at each stage in the SPL engineering lifecycle but also manage variations in associated artifacts such as requirements documents. Furthermore, it aims to bind the variation points in various development stages and dimensions into a coherent variability framework across the life cycle thus providing effective forward and backward traceability of variations and their impact. This makes it possible to develop resilient yet adaptable SPL architectures for exploitation in industrial SPL engineering processes.

This first year OBASCO mainly participates in work-package 3 and 4. Work-package 3 is related to implementation of software product lines and the team wrote the part of the D3.1 deliverable related to software product lines tools and criteria to analyze these tools. Work-package 4 focus on traceability and we collaborate on the D4.1 milestone which is a state of the art of traceability in software product lines, model-driven approaches and aspect-oriented programming.

Overall funding of the network by the EU is 3.78 million euros. OBASCO's share amounts to 370 KEUR. The web page is: http://www.ample-project.net for more details.

8.4. Collaboration with Foreign Research Groups

8.4.1. ECO-NET Project # 16293RG

Participants: Jean-Claude Royer, Jacques Noyé, Mario Südholt, Fabricio Fernandes.

This project is funded by EGIDE and the partners are: the COLOSS team (LINA), the OBASCO team, the DSRG team (Charles University, Prague, Czech Republic) and the CSRL team (Universitatea Babes-Bolyai, Cluj-Napoca, Romania). The core project is to establish a link from component code to component specifications. The main direction is to extract some abstract descriptions from Java programs. Thus this paves the way for verification and compatibility checking. In a first step we expect to define a minimal meta-model for components with protocol descriptions. The second step will be to define tools to extract component descriptions from Java source code. A first workshop with the partners was held during the first week of September 2007.

8.4.2. CONICYT Chili - INRIA CORDIAL Project

Participants: Jacques Noyé, Mario Südholt, Luis Daniel Benavides Navarro, Angel Núñez.

CORDIAL stands for COncuRrency, Distribution, and Interactions in Aspect Languages. The objective of this 2-year project CONICYT/INRIA, starting in January 2008, is to advance the state of the art in concurrent and distributed aspect-oriented programming by leveraging the expertise of the two participating teams: OBASCO in Nantes and the newly-created PLEIAD laboratory, led by Éric Tanter, at *Universidad de Chile*. In particular, we will study different dimensions of concurrent and distributed aspect language mechanisms and consider their composition using Domain Specific Aspect Languages (DSALs). Experiments will be carried out with the prototypes developed by both groups, AWED, Baton, and ReflexD (a platform for implementing distributed aspect languages).

9. Dissemination

9.1. Animation of the Community

9.1.1. Animation

CNRS/GDR GPL: OBASCO is member of the *transformation* group of this new GDR about softare engineering and programming (see http://www-lsr.imag/GPL/).

JFDLPA 2007: OBASCO has co-organized the third edition of the *Journée Francophone sur le Développement de Logiciels Par Aspects* (JFDLPA (see http://pop-art.inrialpes.fr/~jfdlpa07/). This french workshop provides a forum for the community of researchers in AOP, and it promotes expression for young researchers, in particular PhD students.

Les jeudis de l'objet: This bimonthly industrial seminar organized by our group is now ten years old. Surviving the annual conferences Objet/OCM, it has become a great place for local industry to acquire knowledge about emerging technology, exchange ideas about state-of-the-art technologies, share experiences around the technologies associated with objects and components. Each seminar presents either a state of the art of an emerging technology (Spring, Ruby on Rails, Google Web Toolkit, etc.) or feedback on an industrial project in the field of large software architectures (mobility-based applications in a small enterprise, open source middleware...). For more details on the past/future agenda, go to http://www.emn.fr/jeudis-objet.

ACM/SIGOPS: G. Muller was the vice-chair of the ACM/Sigops until June 2007. J.-M. Menaud has been the treasurer of the French SIGOPS Chapter (ASF) since April 2005.

9.1.2. Steering, journal, conference committees

P. Cointe: He is a member of the ECOOP and LMO steering committees (http://www.ecoop.org). He served in the JFDLPA 2007, NOTERE 2007 and TOOLS 2007 committees.

R. Douence: He was a program committee member of AOSD 2007 (*Aspect-Oriented Software Development*) GPCE 2007 (*Generative Programming and Component Engineering*), LMO 2007 (*Langages et Modèles à Objets*) and the co-chair of the third edition of the JFDLPA (*Journée Francophone sur la Programmation Par Aspects*) workshop in Toulouse.

T. Ledoux: He was a program committee member of the Special Issue on Software Components for the journal Annals of Telecommunications. He was a referee for the ICWS 2007 (International Conference on Web Services) and TOOLS 2007.

J.-M. Menaud: He is a member of the (RenPar/CFSE/Sympa) steering committees. He was a program committee member of Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC'07) of the 13th International European Conference on Parallel and Distributed Computing (Euro- Par 2007), Workshop on Virtualization/Xen in High-Performance Cluster and Grid Computing (XHPC 07) of the 16th IEEE International Symposium on High Performance Distributed Computing (HPDC-16) and serves on the program committee of CFSE-6 (6th French Conference on Operating Systems, March 2008, Fribourg, Switzerland).

G. Muller: He is a member of the RENPAR steering committee. He served as a program committee member for Eurosys 2007 (2nd ACM SIGOPS/EuroSys chapter Conference, April 2007, Lisbon, Portugal), ISORC 2007 (10th IEEE International Symposium on Object-oriented Real-time distributed Computing, April 2007, Santorini Island, Greece), IEEE ICDCS 2007 (International Conference on Distributed Computing Systems, June 2007, Toronto, Canada), IEEE DSN/DCCCS 2007 (International Conference on Dependable Systems and Networks, June, 2007, Edinburg, UK), ICWS 2007 (IEEE International Conference on Web Services, July 2007, Salt Lake City, USA), ACM PLOS 2007 (4th Workshop on Programming Languages and Operating Systems, October 2007, Stevenson, Washington, USA), HASE 2007 (10th IEEE High Assurance Systems Engineering Symposium, November 2007, Dallas, Texas, USA), ACM ASPLOS 2008 (13th International Conference on Architectural Support for Programming Languages and Operating Systems, March 2008, Seattle, USA), EDCC-7 (Seventh European Dependable Computing Conference, May 2008, Kaunas, Lithuania), CFSE-6 (6th French Conference on Operating Systems, March 2008, Fribourg, Switzerland), IEEE DSN/PDS 2007 (International Conference on Dependable Systems and Networks, June, 2008, Anchorage, Alaska, USA).

He gave 5 seminars on the Coccinelle project at INRIA Rhône-Alpes, in April 2007, at the University of Paris 6 in July 2007, at INRIA Nancy in October 2007, at the STIC 2007 Symposium in November 2007 in Paris and at INSA Lyon in December 2007.

J. Noyé: He was a program committee member of PEPM 2007 (ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation), Nice, January 2007 and a co-organizer of DSAL 2007 (2nd Domain-Specific Aspect Languages Workshop, Vancouver, March 2007). He is the co-editor of a special issue on Domain-Specific Aspect Languages to be published by the IET Software journal. He is a program committee member of LMO 2008 (*Langages et Modèles à Objets*), Montreal, March 2008, and SC 2008 (Software Composition), Budapest, March 2008.

J.-C. Royer: He is an editor-in-chief of the *RSTI L'Objet* journal, a member of the editorial board of the Journal of Object Technology (JOT), and a member of the steering committee of RSTI (*Revue des Sciences et Technologies de l'Information*, Hermès-Lavoisier). He was co-workshop chair at the TOOLS Conference, Zurich, June 2007 and he has done a review for the Journal of Applied Logic.

M. Südholt: He is on the steering committee of *International Symposium on Software Composition*, a series of ETAPS satellite events. He has co-chaired the international workshops ADI'07 (Aspects, Dependencies and Interactions) and ACP4IS'07 (Aspects, Components, Programs for Infrastructure Software). He has served on the program committees of ECOOP'07 and SC'07 as well as the program committees of the workshops ProVeCS'07 (Programming and verification of component systems), RV'07 (Runtime Verification), and ACP4IS'07 (Aspects, Components, Programs for Infrastructure Software). He has co-edited and published in 2007 the proceedings of the workshops of ECOOP'06.

Finally, he serves as workshop chair of AOSD'08 as well as the program committees of the international conferences and symposia AOSD'08, ECOOP'08 and SC'08, as well as the international workshop on Runtime Verification'08.

9.1.3. Thesis committees

P. Cointe: He was the reviewer of S. Hong Tuan Ha (University of Rennes I, 30/01/07) and N. Pessemier (University of Lille, 27/06/07) PhD thesis as well as R. Marlet's HDR thesis (University of Bordeaux, 23/11/07).

G. Muller: He was a reviewer of the PhD committee of M. Grenier (University of Nancy, 10/2007) and Y. Royon (INSA Lyon, 12/2007). He was a member of the PhD committee of Vidal Martins (University of Nantes, 5/2007) and E. Donin (University of Rennes 1, 9/2007).

M. Südholt: He was a reviewer of the PhD of D. Stauch (INP Grenoble, 13/11/07).

9.1.4. Evaluation committees and expertise

P. Cointe: is a member of the MSTP (*Mission Scientifique Technique Pédagogique*) since March 2003. He is a member of the France-Maroc scientific committee in charge of the *Software Engeenering* cooperation. He is heading the software theme of the *Media and Networks* Cluster (see http://www.images-et-reseaux.com) and the INRIA representative at the *NESSI SRA*.

T. Ledoux: was an expert for the ANR *Technologies Logicielles 2007* call.

G. Muller: was an expert for the ANR *Technologies Logicielles 2007* call.

J.-M. Menaud: is an expert for the *Pays de la Loire Council* in charge of supervising the development of the high bandwidth networks.

9.2. Teaching

EMOOSE. In September 1998, the team set up, in collaboration with a network of partners, an international Master of Science program EMOOSE (European Master of Science on Object-Oriented and Software Engineering Technologies). This program is dedicated to object-oriented software engineering in a broad sense, including component-based and aspect-oriented software development.

The program is managed by the team in cooperation with the *Vrije Universiteit Brussel* (VUB) and the courses take place in Nantes. The students receive a Master of Science degree of the *Vrije Universiteit Brussel* and a *Certificat d'études spécialisées de l'École des Mines de Nantes*. The ninth promotion graduated in August 2007 and the tenth promotion started their first semester in October. See also: http://www.emn.fr/emoose.

AOSD-Europe Summer School. P. Cointe participated to this one week school in Genoa. He gave a lecture on *Reflection and AOP*.

Ecole des jeunes chercheurs du CNRS. R. Douence, J. Noyé and M. Südholt taught on AOP at the EJCP 2007 (*Ecole des jeunes chercheurs en Programmation*, Dinard, May 2007).

ALD Master The faculty members of the team participate to this master program and give lectures about new trends in the field of component and aspect oriented software engineering. 2004 and the implementation of the LMD was the opportunity to redesign the old *DEA informatique*. This lead us to the definition of the ALD master *Architectures Logicielles distribuées* mainly animated and chaired by the ATLAS and OBASCO teams. G. Muller was the co-chair together with J. Martinez from Polytech Nantes until August 2005 when he chairs this formation.

9.3. Collective Duties

P. Cointe: He is chairman of the Computer Science Department at EMN, the next chairman of the *Laboratoire Informatique de Nantes Atlantique* (UMR LINA CNRS) and the chairman of the *Software Engineering Cluster* associated to the new CPER 2007-2010 and its *MILES* project.

G. Muller: He is a member of the board of the *École Doctorale STIM* and the LINA laboratory.

T. Ledoux: He is a member of the board of the LINA laboratory.

10. Bibliography

Major publications by the team in recent years

[1] L. D. BENAVIDES NAVARRO, M. SÜDHOLT, W. VANDERPERREN, B. DE FRAINE, D. SUVÉE. *Explicitly distributed AOP using AWED*, in "Proceedings of the 5th ACM Int. Conf. on Aspect-Oriented Software Development (AOSD'06)", ACM Press, March 2006.

[2] P. COINTE, H. ALBIN-AMIOT, S. DENIER. *From (meta) objects to aspects : from Java to AspectJ*, in "Third International Symposium on Formal Methods for Components and Objects, FMCO 2004, Leiden, The Netherlands", F. DE BOER, ET AL (editors), Lecture Notes in Computer Science, vol. 3657, Springer-Verlag, November 2005, p. 70-94.

[3] R. DOUENCE, P. FRADET, M. SÜDHOLT. *Trace-Based Aspects*, in "Aspect-Oriented Software Development", M. AKSIT, S. CLARKE, T. ELRAD, R. E. FILMAN (editors), Addison-Wesley Professional, September 2004, p. 201-218.

[4] R. DOUENCE, P. FRADET, M. SÜDHOLT. *A framework for the detection and resolution of aspect interactions*, in "Generative Programming and Component Engineering: ACM SIGPLAN/SIGSOFT Conference, GPCE 2002 - Proceedings, Pittsburgh, PA, USA", D. BATORY, C. CONSEL, W. TAHA (editors), Lecture Notes in Computer Science, vol. 2487, Springer-Verlag, October 2002, p. 173–188.

[5] R. DOUENCE, T. FRITZ, N. LORIANT, J.-M. MENAUD, M. SÉGURA-DEVILLECHAISE, M. SÜDHOLT. *An expressive aspect language for system applications with Arachne*, in "Proc. of 4th International Conference on Aspect-Oriented Software Development (AOSD'05)", ACM Press, March 2005.

[6] R. DOUENCE, O. MOTELET, M. SÜDHOLT. *A formal definition of crosscuts*, in "Proceedings of the 3rd International Conference on Reflection 2001, Kyoto, Japan", A. YONEZAWA, S. MATSUOKA (editors), Lecture Notes in Computer Science, vol. 2192, Springer-Verlag, September 2001, p. 170–186.

[7] R. DOUENCE, D. LE BOTLAN, J. NOYÉ, M. SÜDHOLT. *Concurrent Aspects*, in "Proc. of the Int. ACM Conf. on Generative Programming and Component Engineering (GPCE)", ACM Press, October 2006.

[8] Y. PADIOLEAU, J. LAWALL, G. MULLER. *Understanding Collateral Evolution in Linux Device Drivers*, in "Proc. of the first ACM SIGOPS EuroSys conference (EuroSys 2006), Leuven, Belgium", April 2006, p. 59-71.

[9] E. TANTER, N. BOURAQADI-SAÂDANI, J. NOYÉ. *Reflex - Towards an Open Reflective Extension of Java*, in "Proceedings of the 3rd International Conference on Reflection 2001, Kyoto, Japan", A. YONEZAWA, S. MATSUOKA (editors), Lecture Notes in Computer Science, vol. 2192, Springer-Verlag, September 2001, p. 25-42.

[10] E. TANTER, J. NOYÉ, D. CAROMEL, P. COINTE. *Partial Behavioral Reflection: Spatial and Temporal Selection of Reification*, in "Proceedings of the 18th ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2003), Anaheim, California, USA", R. CROCKER, GUY L. JR. STEELE (editors), vol. 38, n^{o} 11, ACM Press, October 2003, p. 27–46.

Year Publications

Books and Monographs

[11] J. FABRY, D. REBERNAK, T. CLEENEWERCKAND, A.-F. LE MEUR, J. NOYÉ, É. TANTER (editors). *DSAL'07: Proceedings of the 2nd workshop on Domain-Specific Aspect Languages*, ACM Press, Vancouver, British Columbia, Canada, 2007.

[12] O. SPINCZYK, M. SÜDHOLT, C. GIBBS (editors). *ACP4IS'07: Proceedings of the 6th workshop on Aspects, components, and patterns for infrastructure software*, ACM Press, New York, NY, USA, 2007.

[13] M. SÜDHOLT, C. CONSEL (editors). *ECOOP 2006 Workshop Reader*, LNCS, vol. 4379, Springer Verlag, 2007.

Doctoral dissertations and Habilitation theses

[14] S. DENIER. *Expression et composition des motifs de conception avec les aspects*, Ph. D. Thesis, École des Mines de Nantes and Université de Nantes, July 2007.

[15] N. LORIANT. *Evolution dynamique des systèmes d'exploitation, une approche par la programmation par aspects*, Ph. D. Thesis, École des Mines de Nantes and Université de Nantes, December 2007.

[16] M. SÜDHOLT. *Towards expressive, well-founded and correct Aspect-Oriented Programming*, Mémoire d'habilitation à diriger des recherches, July 2007.

Articles in refereed journals and book chapters

[17] É. TANTER, R. TOLEDO, G. POTHIER, J. NOYÉ. *Flexible Metaprogramming and AOP in Java*, in "Science of Computer Programming - Special issue on Experimental Software and Toolkits", to appear, 2008.

Publications in Conferences and Workshops

[18] H. F. ARBOLEDA JIMENEZ, R. CASALLAS, J.-C. ROYER. *Comparing two Implementations of an Approach for Managing Variability in Product Line Construction Using the GMF and GME Frameworks*, in "Proceedings of the 5th Nordic Workshop on Model Driven Engineering", August 2007, p. 67 – 82.

[19] H. F. ARBOLEDA JIMENEZ, R. CASALLAS, J.-C. ROYER. *Dealing with Constraints during a Feature Configuration Process in a Model-Driven Software Product Line*, in "Proceedings of The 7th OOPSLA Workshop on Domain-Specific Modeling", October 2007.

[20] F. BALIGAND, T. LEDOUX, P. COMBES. *Une Approche pour Garantir la Qualité de Service dans les Orchestrations de Services Web*, in "7ème Conférence Internationale sur les NOuvelles TEchnologies de la REpartition (NOTERE 2007)", June 2007.

[21] F. BALIGAND, N. RIVIERRE, T. LEDOUX. *A Declarative Approach for QoS-Aware Web Service Compositions*, in "Proceedings of the 5th International Conference on Service Oriented Computing (ICSOC)", Springer Verlag, September 2007.

[22] L. D. BENAVIDES NAVARRO, C. SCHWANNINGER, R. SOBOTZIK, M. SÜDHOLT. *ATOLL: Aspect-Oriented Toll System*, in "Proc. 6th Int. Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS'06) at AOSD, New York, NY, USA", ACM Press, 2007.

[23] L. D. BENAVIDES NAVARRO, M. SÜDHOLT, R. DOUENCE, J.-M. MENAUD. *Invasive patterns for distributed programs*, in "Proc. of the 9th International Symposium on Distributed Objects, Middleware, and Applications (DOA'07)", LNCS, Springer Verlag, November 2007.

[24] L. D. BENAVIDES NAVARRO, M. SÜDHOLT, R. DOUENCE, J.-M. MENAUD. *Invasive patterns: aspect-based adaptation of distributed applications*, in "4th International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT´07) at the 21st European Conference on Object-Oriented Programming ECOOP'07", July 2007.

[25] L. BURGY, L. RÉVEILLÈRE, J. LAWALL, G. MULLER. *A Language-Based Approach for Improving the Robustness of Network Application Protocol Implementations*, in "26th IEEE International Symposium on Reliable Distributed Systems", October 2007.

[26] P. COINTE. *On the Evolution of Programming Languages*, in "Les paradigmes informatiques appliqués à la musique (Festival Agora), IRCAM, Paris - France", A. GERZO (editor), Conférence invitée, June 2007.

[27] P. COINTE. *Quelques réflexions sur la programmation post objet*, in "NOuvelles TEchnologies de la RÉpartition (NOTERE), Marrakech, Maroc", M. ERRADI (editor), Conférence invitée, June 2007, 391.

[28] B. DE FRAINE, M. SÜDHOLT, V. JONCKERS. *StrongAspectJ: Flexible and Safe Pointcut/Advice Bindings*, in "Proceedings of the 7th ACM Int. Conf. on Aspect-Oriented Software Development (AOSD'08)", M. MEZINI (editor), To appear, ACM Press, March 2008.

[29] S. DJOKO DJOKO, R. DOUENCE, P. FRADET. *Aspects Preserving Properties*, in "Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'08), San Francisco, CA, USA", to appear, ACM Press, January 2008.

[30] M. LEGER, T. LEDOUX, T. COUPAYE. *Reliable dynamic reconfigurations in the Fractal component model*, in "Proc. 6th Workshop on Adaptive and Reflective Middleware (ARM2007) at Middleware", ACM Digital Library, November 2007.

[31] N. LORIANT, J.-M. MENAUD. *Generalized Dynamic Probes for the Linux Kernel and Applications with Arachne*, in "Proceedings of the 2007 IADIS Applied Computing International Conference (AC'O7), Salamanca, Spain", February 2007.

[32] D. H. NGUYEN, M. SÜDHOLT. *Property-preserving evolution of components using VPA-based aspects*, in "4th ECOOP'2007 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE)", 2007.

[33] D. H. NGUYEN, M. SÜDHOLT. *Property-preserving evolution of components using VPA-based aspects*, in "Proceedings of the 9th International Symposium on Distributed Objects, Middleware, and Applications (DOA'07)", Springer-Verlag, November 2007.

[34] A. NÚÑEZ, J. NOYÉ. *A Domain-Specific Language for Coordinating Concurrent Aspects in Java*, in "3ème Journée Francophone sur le Développement de Logiciels Par Aspects (JFDLPA 2007), Toulouse, France", R. DOUENCE, P. FRADET (editors), March 2007.

[35] A. NÚÑEZ, J. NOYÉ. *A Seamless Extension of Components with Aspects using Protocols*, in "WCOP 2007 - Components beyond Reuse - 12th International ECOOP Workshop on Component-Oriented Programming, Berlin, Germany", R. REUSSNER, C. SZYPERSKI, W. WECK (editors), July 2007.

[36] Y. PADIOLEAU, J. LAWALL, G. MULLER. *Semantic Patches, Documenting and Automating Collateral Evolutions in Linux Device Drivers*, in "Ottawa Linux Symposium (OLS 2007)", June 2007.

[37] J. POLAKOVIC, S. MAZARÉ, J.-B. STEFANI, P.-C. DAVID. *Exprience with Implementing safe reconfigurations in component-based embedded systems*, in "The 10th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2007), Boston, MA, USA", LNCS, Springer Verlag, ACM, July 2007.

[38] H. STUART, R. R. HANSEN, J. LAWALL, J. ANDERSEN, Y. PADIOLEAU, G. MULLER. *Towards Easing the Diagnosis of Bugs in OS Code*, in "4th Workshop on Programming Languages and Operating Systems (PLOS 2007)", October 2007.

[39] R. URUNUELA, G. MULLER, J. LAWALL. *Towards Class-Based Dynamic Voltage Scaling for Multimedia Applications*, in "Power-aware Computing Systems", L. BENINI, N. CHANG, U. KREMER, C. W. PROBST (editors), Dagstuhl Seminar Proceedings, n^o 07041, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[40] F. DE ALEXANDRIA FERNANDES, J.-C. ROYER. *Components with Symbolic Transition Systems: A Java Implementation of Rendez-Vous*, in "Proceedings of the Communicating Process Architecture Conference 2007", Concurrent Systems Engineering, vol. 65, IOS Press, 2007, p. 89-107.

Internal Reports

[41] B. DE FRAINE, M. SÜDHOLT, V. JONCKERS. *A Formal Semantics of Flexible and Safe Pointcut/Advice Bindings*, Technical report, n^o SSEL 02/2007/a, Vrije Universiteit Brussel, October 2007.

[42] S. DJOKO DJOKO, R. DOUENCE, P. FRADET. *Proof of correctness of aspect transformations in the CASB*, Research Report, n^o D88, Network of Excellence in AOSD (AOSD-Europe), July 2007.

Miscellaneous

[43] F. DE ALEXANDRIA FERNANDES, J.-C. ROYER. *The STSLIB Project: Towards a Formal Component Model Based on STS*, Formal Aspects of Component Software to appear, 2007.

References in notes

[44] M. AKSIT, A. BLACK, L. CARDELLI, P. COINTE, J. COPLIEN, R. GUERRAOUI, G. KICZALES, D. LEA, O. MADSEN, B. MAGNUSSON, J. MESEGUER, H. MOESSENBOECK, J. PALSBERG, D. SCHMIDT. *Strategic Research Directions in Object Oriented Programming*, in "ACM Computing Surveys", vol. 28, n^o 4, December 1996, p. 691-700.

[45] N. BOURAQADI-SAÂDANI, T. LEDOUX, F. RIVARD. *Safe Metaclass Programming*, in "Proceedings of OOPSLA 1998, Vancouver, British Columbia, USA", C. CHAMBERS (editor), vol. 33, n^o 10, ACM Press, ACM SIGPLAN, October 1998, p. 84–96.

[46] S. CHANDRA, B. RICHARDS, J. LARUS. *Teapot: Language Support for Writing Memory Coherence Protocols*, in "Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation, Philadelphia, PA", ACM SIGPLAN Notices, 31(5), May 1996, p. 237–248.

[47] P. COINTE. *Les langages à objets*, in "Technique et Science Informatique", vol. 19, n^o 1-2-3, 2000, p. 139-146.

[48] P. COINTE. *Towards Generative Programming*, in "Unconventional Programming Paradigms, UPP 2005, Mont St Michel, France", J.-P. BANÂTRE, P. FRADET, J.-L. GIAVITTO, O. MICHEL (editors), Lecture Notes in Computer Science, vol. 3566, Springer-Verlag, September 2005, p. 302–312.

[49] P. COINTE. *La programmation par aspects relève le défi des usines logicielles*, Carte blanche à l'occasion d'ECOOP 2006, 01 Informatique, page 28, 30 juin 2006.

[50] P. COINTE. *Les langages à objets*, Chapitre de l'Encyclopédie de l'informatique et des systèmes d'information, Vuibert, October 2006.

[51] P. COINTE. *Metaclasses are First Class: The ObjVlisp Model*, in "Proceedings of the second ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications (OOPSLA 1987), Orlando, Florida, USA", J. L. ARCHIBALD (editor), vol. 22, n^o 12, ACM Press, October 1987, p. 156–167.

[52] P. COINTE. *CLOS and Smalltalk - A comparison*, in "Object-Oriented Programming: The CLOS Perspective", A. PAEPCKE (editor), chap. 9, MIT PRESS, 1993, p. 216-250.

[53] P. COINTE, J. NOYÉ, R. DOUENCE, T. LEDOUX, J.-M. MENAUD, G. MULLER, M. SÜDHOLT. *Programmation post-objets : des langages d'aspects aux langages de composants*, in "RSTI L'Objet", vol. 10, n^o 4, 2004, http://www.lip6.fr/colloque-JFP.

[54] C. CONSEL, F. LATRY, L. RÉVEILLÈRE, P. COINTE. *A Generative Programming Approach to Developing DSL Compilers*, in "Proceedings of the 4th International Conference on Generative Programming and Component Engineering (GPCE'05), Tallinn, Estonia", R. GLÜCK, M. LOWRY (editors), Lecture Notes in Computer Science, Springer-Verlag, September/October 2005.

[55] K. CZARNECKI, U. W. EISENECKER. *Generative Programming. Methods, Tools and Applications*, 2nd printing, Addison Wesley, 2000.

[56] DGE. *Technologies clés 2010*, Ministère de l'Économie, des Finances et de l'Industrie, NRF, 2006.

[57] A. GOLDBERG, D. ROBSON. *SMALLTALK-80. THE LANGUAGE AND ITS IMPLEMENTATION*, Addison Wesley, 1983.

[58] IBM. *Practical Autonomic Computing: Roadmap to Self Managing Technology*, Research Report, January 2006, http://www-03.ibm.com/autonomic/pdfs/AC_PracticalRoadmapWhitepaper_051906.pdf.

[59] R. JAGADEESAN, A. JEFFREY, J. RIELY. *Typed Parametric Polymorphism for Aspects*, in "Science of Computer Programming", Special issue on Foundations of AOP, vol. 63, n^o 3, 2006, p. 267–296.

[60] N. JONES, C. GOMARD, P. SESTOFT. *Partial Evaluation and Automatic Program Generation*, International Series in Computer Science, Prentice Hall, 1993.

[61] J. KEPHART, D. CHESS. *The Vision of Autonomic Computing*, in "IEEE Computer", vol. 36, n^{o} 1, January 2003, p. 41–50.

[62] G. KICZALES, J. M. ASHLEY, L. RODRIGUEZ, A. VAHDAT, D. BOBROW. *Metaobject protocols: Why we want them and what else they can do*, in "Object-Oriented Programming: The CLOS Perspective", A. PAEPCKE (editor), chap. 4, MIT PRESS, 1993, p. 101-118.

[63] G. KICZALES, J. LAMPING, A. MENDHEKAR, C. MAEDA, C. LOPES, J. LOINGTIER, J. IRWIN. *Aspect-Oriented Programming*, in "ECOOP'97 - Object-Oriented Programming - 11th European Conference, Jyväskylä, Finnland", M. AKSIT, S. MATSUOKA (editors), Lecture Notes in Computer Science, vol. 1241, Springer-Verlag, June 1997, p. 220–242.

[64] T. LEDOUX. *OpenCorba: a Reflective Open Broker*, in "ACM Meta-Level Architectures and Reflection, Second International Conference, Reflection'99, Saint-Malo, France", P. COINTE (editor), Lecture Notes in Computer Science, vol. 1616, Springer-Verlag, July 1999, p. 197–214.

[65] R. MARLET. *Spécialiser les programmes. Spécialiser les langages*, Mémoire d'habilitation à diriger des recherches, November 2007.

[66] M. MCILROY. *Mass produced software components*, in "Proceedings of the NATO Conference on Software Engineering, Garmish, Germany", P. NAUR, B. RANDELL (editors), NATO Science Committee, October 1968, p. 138-155.

[67] N. MEDVIDOVIC, R. TAYLOR. *A Classification and Comparison Framework for Software Architecture Description Languages*, in "IEEE Transactions on Software Engineering", vol. 26, n^{o} 1, January 2000, p. 70-93.

[68] J.-M. MENAUD, J. NOYÉ, P. COINTE, C. PEREZ. *Mixing Aspects and components for Grid Computing*, in "International Workshop on Grid Systems, Tools and Environments", October 2005.

[69] M. MERNIK, J. HEERING, A. SLOANE. *When and How to Develop Domain-Specific Languages*, in "ACM Computing Surveys (CSUR)", vol. 37, n^{o} 4, December 2005, p. 316-344, http://doi.acm.org/10.1145/1118890.1118892.

[70] 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, p. 19–24.

[71] G. MULLER, J. LAWALL, H. DUSCHENE. *A Framework for Simplifying the Development of Kernel Schedulers: Design and Performance Evaluation*, in "HASE 2005 - 9th IEEE International Symposium on High-Assurance Systems Engineering, Heidelberg, Germany", October 2005.

[72] 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", USENIX Association, October 2000, p. 17–30.

[73] P. POIZAT, J.-C. ROYER. *A Formal Architectural Description Language based on Symbolic Transition Systems and Modal Logic*, in "Journal of Universal Computer Science", vol. 12, n^{o} 12, 2006, p. 1741-1782.

[74] R. PRIETO-DIAZ, J. NEIGHBORS. *Module Interconnection Languages*, in "The Journal of Systems and Software", vol. 6, n^{o} 4, November 1986, p. 307-334.

[75] M. SHAW, D. GARLAN. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[76] B. SMITH. *Procedural reflection in programming languages*, Ph. D. Thesis, Massachusetts Institute of Technology, 1982.

[77] C. SZYPERSKI. *Component Software*, 2nd edition, ACM Press, 2003.

[78] M. SÉGURA-DEVILLECHAISE, J.-M. MENAUD, G. MULLER, J. LAWALL. *Web Cache Prefetching as an aspect: Towards a Dynamic-Weaving Based Solution*, in "Proceedings of the 2nd international conference on Aspect-oriented software development, Boston, Massachusetts, USA", ACM Press, March 2003, p. 110–119.

[79] E. TANTER, J. NOYÉ. *A Versatile Kernel for Multi-Language AOP*, in "Generative Programming and Component Engineering (GPCE), Tallinn, Estonia", R. GLÜCK, M. LOWRY (editors), Lecture Notes in Computer Science, vol. 3676, Springer-Verlag, September/October 2005, p. 173-188.

[80] 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.

[81] D. THOMAS. *Reflective Software Engineering - From MOPS to AOSD*, in "Journal Of Object Technology", vol. 1, n^{o} 4, October 2002, p. 17-26, http://www.jot.fm/issues/issue_2002_09/column1.

[82] M. WAND, G. KICZALES, C. DUTCHYN. *A Semantics for Advice and Dynamic Join Points in Aspect-Oriented Programming*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", vol. 26, n^{o} 5, 2004, p. 890–910.

[83] P. WEGNER. *Dimension of Object-Based Language Design*, in "Proceedings of the second ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications (OOPSLA 1987), Orlando, Florida, USA", J. L. ARCHIBALD (editor), vol. 22, n^{o} 12, ACM Press, October 1987, p. 168–182.