INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team OBASCO

OBjects, ASpects, and COmponents

Rennes

THEME COM

Table of contents

1. Team

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

Head of Project-team

Pierre Cointe [Professor, Ecole des Mines de Nantes (EMN), HdR]

Staff Member INRIA

Jean-Marc Menaud [CR2 INRIA on leave from EMN 1/09/2006-30/08/2007]
Mario Südholt [CR2 INRIA on leave from EMN 1/09/2004-30/08/2006]

Faculty Members from EMN

Rémi Douence [Associate Professor]
Thomas Ledoux [Associate Professor]
Jean-Marc Menaud [Associate Professor -30/08/2006]
Gilles Muller [Professor, HdR]
Jacques Noyé [Associate Professor]
Jean-Claude Royer [Professor, HdR]
Mario Südholt [Associate Professor 01/09/2006-]

Administrative Assistant

Elodie Lize [Part-time (50%)]
Diana Gaudin [Part-time (50%)]

Temporary Faculty Members

Pierre-Charles David [Research engineer ANR grant 1/09/2006-]
Simon Denier [Assistant Professor 1/09/2006-30/08/2007]
Didier Le Botlan [CNRS & REGIONAL COUNCIL postdoctoral grant 1/01/2005-30/08/2006]
Yoann Padioleau [Postdoctoral position ANR grant 1/10/2005-]

Visiting Scientist

Julia Lawall [DIKU, UNIVERSITY OF COPENHAGEN, June to July 2006]

Ph. D. Students

Hugo F. Arboleda Jimenez [MINES & Los Andès (Bogota Colombia) University grant, 1/1/2006-]
Ali Assaf [MESR grant 3/10/2005-]
Christophe Augier [Cifre grant with JANULA 1/10/2004-]
Luis Daniel Benavides Navarro [MINES grant, 1/10/2005-]
Gustavo Bobeff [ProxiAD, 1/09/2001-14/12/2006]
Simon Denier [EMN grant 1/10/2003-30/08/2006]
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-]
Nicolas Loriant [EMN grant 1/10/2004-]
Florian Minjat [MESR grant 1/10/2004-]
Dong Ha Nguyen [REGIONAL COUNCIL & AOSD-Europe grant, 1/04/2005-]
Angel Núñez [MINES & STREP AMPLE grant 1/10/2006-]
Sebastian Pavel [MESR grant, ACI DISPO 1/10/2003-1/10/2006]
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 [4]. We are (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, (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 to 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, to develop a formal foundation for AOP including concurrency and distribution and to enable 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 Aspect Domain Specific Languages (ADSL) 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.

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 [4][13]. 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 (DSL).

Historically the core members of OBASCO have a strong background in the design and implementation programming languages (more particularly OOL) [1], [10] [50]. This background has been enriched by an expertise in middleware and operating systems [7], [8]. Our goal is to take advantage of this complementarity by developing a methodology and a set of technics 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 [78]).

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* [72].

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 [62]).

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* [71].

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.

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.

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* [72] as building blocks for today's software systems. The basic idea, as initially explained by M.D. McIlroy in 1968 [63], 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 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*.

[1] See the discussions at http://www.dreamsongs.com/Essays.html and those at http://www-poleia.lip6.fr/~briot/colloque-JFP/, in particular [4].

These properties raise new challenges in programming language design and implementation. They require an integration of ideas coming from module interconnection languages [69], architecture description languages (ADLs) [70], [64] 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 language* [72] prototypes 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 [59]) 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 certain values 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. In AspectJ, these are method-like constructs used to define additional behavior at join points.

Aspect: concern crosscutting a set of traditional modular units (classes, packages, modules, components...); defines some pointcuts and advice. In AspectJ, aspects are units of modular crosscutting implementation, composed of pointcuts and advice, and ordinary Java member declarations.

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 [77]. Aspect-oriented programming as well as aspect-oriented modeling is an extremely active field of research where people try to go beyond the object model by providing [3]:

- *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 and then woven with the associated base program in a non intrusive way.

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 [52], [53], [61] to extend the base language with new mechanisms such as multiple inheritance, concurrency or metaclass composition [2].

- In the case of open middleware [7], 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 [3]. 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], [75]. 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 [51]. A fourth issue is to apply domain-specific languages to the definition of aspects [45].

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 [8], [66].

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) [76].

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 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: *enterprise information systems*, *telecommunication.*

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 industrializing just in time such software components, it becomes strategic to define product lines for generating (producing out) components in an automatic way (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], [51]. 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 domain of autonomic computing we investigate how to dynamically adapt software components to their execution contexts; (ii) in the OS field we provide support for advanced crosscutting functionalities, such as process scheduling, drivers evolution and grid services; (iii) in the middleware area we are considering system configuration and aspectualisation of non-modular functionalities.

4.2. Autonomic Computing

Distributed systems are more and more complex due to their heterogeneous architectures (on the physical and software level) in which resources are numerous and where their availability evolves at runtime. This complexity and dynamicity requiring permanent adaptation, from the application to the system level, and need 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 a 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 defined in four functional areas [58], [60]:

- self-configuration: characteristics that enable systems to adapt to changing conditions by changing their own configurations and that allows 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-sonfiguration 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 [24], 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 [30].

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 [49], [67].

A first application direction is to use ADSLs 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 5.4).

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. Middleware and Enterprise Information Systems

Stimulated by the growth of network-based applications, middleware technologies are taking an increasing importance. They cover a wide range of software systems, including distributed objects and components, mobile applications and finally ubiquitous computing. Companies and organizations are now using middleware technologies to build enterprise-wide information systems by integrating previously independent applications, together with new developments.

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 JBossCache. We have shown how aspects with explicit abstractions for distribution allow to improve the structure of such applications and therefore facilitate their evolution [34]. 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 [35]. 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 system 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).

5. Software

5.1. Arachne

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

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

We transposed the EAOP model into Arachne, our aspect dynamic weaver for C legacy applications. Arachne has been developed to dynamically evolve a system at runtime without interrupting servicing. for instance to changing 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 [73], thus enabling several improvements. 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 where an aspect developer can write it's own joinpoint or pointcut.

We are extending Arachne in two directions : supporting the introduction of new pointcut [18], and supporting the introduction of reflection [33]. We also have further explored applications of Arachne to web caching [19].

Arachne, is presented in M. Ségura 's PhD thesis [74]. and is publicly available at:http://www.emn.fr/x-info/arachne.

5.2. Awed

Keywords: *AOP*, *distributed programming*.

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

The system Aspects With Explicit Distribution (AWED) [34], [35] 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.

The AWED model 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. 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, the AWED system has been applied to replicated caching infrastructures of distributed component platforms and web services (see 5.2)

5.3. Baton

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

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

Baton [47] 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.

5.4. Bossa

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

Participants: Gilles Muller [correspondent], Christophe Augier, Julia Lawall, 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 [68]. 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). Additionaly, 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 [41].

5.5. Reflex

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

Participants: Jacques Noyé [correspondent], Ali Assaf.

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 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 in the context of the OSCAR project

5.6. Safran

Keywords: *Fractal*, *context-awareness*, *dynamic adaptation*, *self-adaptive components*.

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

Safran is an extension of the Fractal component model (http://fractal.objectweb.org/) to support the development of self-adaptive components, i.e. autonomous software components which adapt themselves to the evolutions of their execution context [23], [16]. It was designed and implemented by Pierre-Charles David in 2005 during his PhD thesis. Safran provides (i) a simple domain-specific language (also named Safran) to program reactive adaptation policies and (ii) a mechanism to dynamically attach and detach these policies to the Fractal components of an application.

Safran is composed of three sub-systems on top of Fractal:

1. FScript [24] is a domain-specific language used to program the component reconfigurations which will adapt the application. It provides a custom notation which makes it easy to navigate in the components Fractal architecture and offers certain guarantees on the changes applied to the target application, for example the atomicity of the reconfigurations.
2. WildCAT is a generic toolkit to build context-aware applications. It is used by Safran policies to detect the changes in the application's execution context which should trigger adaptations.
3. Finally, an adaptation Fractal controller binds FScript and WildCAT through the reactive rules of adaptation policies. These rules follow the Event-Condition-Action pattern, where the events are detected by WildCAT and the actions are FScript reconfigurations. The adaptation controller allows the dynamic attachment of Safran policies to individual Fractal components and is responsible for their execution.

In 2006, we decided to focus on and only support the first two sub-systems FScript and WildCAT. To this end, we proposed FScript as an extension of Fractal to the Fractal Community and we submitted the WildCAT project to the ObjectWeb College of Architects. FScript is now available in the core Fractal project (http://fractal.objectweb.org/); ObjectWeb accepted the hosting of the WildCAT project (http://wildcat.objectweb.org/). The development of FScript and WildCAT will continue inside the Selfware project for autonomic computing.

6. New Results

6.1. Components

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

Participants: Thomas Ledoux, Jacques Noyé, Jean-Claude Royer, Mario Südholt, Marc Léger, Fabien Baligand, Pierre-Charles David.

Our work on components has focused on two groups of related results. First, several results consider how certain composition properties of components can be analyzed and ensured in the presence of component adaption based on notions of explicit protocols that are part of component interfaces. Second, we have presented a range of results on conceptual and implementation support for the adaptation of software components.

Some of these results strongly link OBASCO's work on CBSE with that on AOP because the concern properties of component-based systems in the presence of modifications expressed in terms of aspects. Furthermore, have been partially integrated into two publicly available software systems, in particular the AWED system for aspects with explicit distribution (see Sec. 5.2) and the Safran system for adaptation of Fractal components (see Sec. 5.6). 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. Explicit protocols to support composition properties

This year we have investigated two classes of protocols to support composition interfaces: protocols defined using visibly-pushdown automata and symbolic transition systems. We have shown that these two approaches, albeit strictly more expressive than finite-state based protocols, support reasoning about interesting component properties.

6.1.1.1. VPA-based protocols

In the context the PhD thesis of Dong Ha Nguyen we have started to investigate 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 defined a language for the definition of such protocols [36] and started a study of their properties in the context of component-based software development. Furthermore, we have defined an aspect language for the manipulation of VPA-based protocols.

6.1.1.2. Property Checking using Symbolic Transition Systems

We are also interested in ways to check properties about components and architectures. We focus on availability properties for components, in order to avoid, for example, denials of services. The boundedness of Symbolic Transition System (STS) is a crucial point in the context of resources or services availability. We extend a first approach based on dictionary of services for asynchronously communicating systems. This leads to a notion of counter STS and a boundedness decision procedure for such counters systems. This boundedness decision procedure may prevent the state-explosion problems existing in STS. We have defined and formalised a notion of bounded decomposition which allows us to exhibit a finite state simulation of an STS. Since it is a simulation of the original system it may be used to prove safety properties. This approach complements model-checking since there are situations in which it succeeds while model-checking fails. In [40], we propose the notions of *bounded analysis* and *bounded decomposition*. This approach tests boundedness of a possibly infinite system, and then generates a finite simulation for it. Afterward, standard model-checking techniques can be used for verification purposes. This is illustrated through a mutual protocol exclusion and two versions of a resource allocator.

6.1.2. 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.

6.1.2.1. 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. The study has clearly shown strong dependencies between the two functionalities and resulted in concrete examples which types of refactorings and evolutive actions are hindered or inhibited by such dependencies [34]. We have also proposed a solution to such modularization properties by means of an aspect language with means for explicit distributed programming.

6.1.2.2. Communication Integrity in Fractal/Julia

Ensuring conformity between the specification of a component-based software architecture and its implementation requires to respect some structural constraints. Communication integrity, one of these constraints, is guaranteed statically at compile-time in ArchJava. However, for a more open model such as Fractal, most reconfigurations in the system are dynamic. We have worked on the possible violations of communication integrity in Julia, an implementation of Fractal in Java, and we have proposed a dynamic mechanism to guarantee this integrity property in Julia [31].

6.1.2.3. Safe Dynamic Reconfigurations of Fractal Architectures

Safe dynamic reconfigurations is an open issue in the building of autonomic computing applications. We propose FScript [24], a dedicated language used to program structural reconfigurations of Fractal architectures. Compared to the use of the standard Fractal APIs in a general purpose language, FScript offers better syntactic support for navigation, more dynamicity, and guarantees on the reconfigurations (termination, atomicity, consistency, and isolation). FScript introduces a new notation, called FPath, which is designed to express queries on Fractal architectures, navigating inside them and selecting elements according to predicates. It also provides access to all the primitive reconfiguration actions available in Fractal, and enables the user to define custom reconfigurations using a simple procedural language.

6.1.2.4. QoS problems in Web Services 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], we propose a new approach aiming to provide the architect with adequate means to specify QoS requirements in Web Service compositions. 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.

6.1.3. Miscellaneous

Furthermore, Mario Südholt has co-edited a summary of current research topics and the corresponding open issues that has been published as part of the proceedings of the 5th international symposium on software composition [12].

6.2. Aspects

Keywords: *aspectualisation*, *concurrency*, *distribution*, *events*, *expressive aspect languages*, *semantics*.

Participants: Mario Südholt, Pierre Cointe, Rémi Douence, Thomas Ledoux, Jacques Noyé, Luis Daniel Benavides Navarro, Pierre-Charles David, Simon Denier, Simplice Djoko Djoko, Dong Ha Nguyen, Angel Núñez, Nicolas Loriant.

OBASCO's work on aspect-oriented programming is targeted towards the development of, support for, and application of expressive aspect languages. This year 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 enhancment 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 conceptual results have been implemented as part of different publicly available software systems: the AWED system for aspects with explicit distribution (see Sec. 5.2), the BATON system that implements concurrent aspects in the CEAOP sense (see Sec. 5.3), a new version of the Arachne system for dynamic weaving of aspects in C/C++ systems (see Sec. 5.1), and the Safran tool for component adaptation using aspects (see Sec. 5.6). They have also been applied to two of OBASCO's main application domains: distributed operating systems and enterprise information systems, in particular, web services.

6.2.1. Formalization

6.2.1.1. Event-based AOP (EAOP)

Event-based AOP is an approach to aspect-oriented programming, based on the concept of 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.

The initial model 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. We have therefore developed a model for concurrent aspects, the model of *Concurrent EAOP (CEAOP)* [28], [29], [47], [43]. This model allows to express the concurrency relationships between aspects and a base program as well as among aspects themselves directly in terms of entities of aspects and the base program: parts of advice, for instance, may be executed asynchronously with the base program, while other parts are specified to executed synchronously. This is in contrast to current approaches for the modularization of concurrent applications using AOP because these approaches only allow to manipulate concurrent frameworks using sequential aspect languages, such as using the new Java5 concurrency library to handle concurrency using the AspectJ system. The CEAOP model is formally based on Finite State Processes, which enables static analyses using the model checker LTSA (e.g., deadlock detection but also analysis of more general refinement properties). In order to facilitate concurrent aspect-oriented programming we have defined different composition operators for concurrent aspects. Finally, we have implemented the CEAOP model in Java and provided a library of aspect composition operators.

A second new instantiation of the EAOP model that we have introduced features more expressive sequence pointcuts, the so-called *VPA-based aspects* [36], which are based on visibly pushdown automata (VPA). These automata define a class of languages which 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.

Finally, we have explored another instantiation of the EAOP model: *relational aspects* [27]. In this instance, an event adds a tuple of objects (e.g., the receiver of the method and its arguments) to a Datalog data base. An event can also query the data base in order to infer relations between objects. This enables context passing properties that go far beyond stack inspection to be expressed using aspects (in particular, extending AspectJ's `cflow` mechanism and runtime graph inspection). Moreover Datalog offers a good trade-off between power of expression (e.g., recursive definitions of relations are possible) and safety/efficiency (e.g., all queries are guaranteed to terminate). Recently [48], we have compared the performance of a specialized compilation scheme for Datalog-based queries with a solution embedding a full Datalog engine.

6.2.1.2. Common Aspect Semantics Base (CASB)

As part of the European Network of Excellence in AOSD we have defined a common aspect semantics base (CASB) [42]. Existing semantics for aspects provide a semantics as a whole but do not isolate the different features of aspect languages. They are typically based on a specific programming paradigm (e.g., object oriented, functional, process based) and model specific aspect systems. We have defined a common aspect semantics base 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. Finally, we have shown how to define general aspect mechanisms and covered very 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.

6.2.2. Language Design and Applications

6.2.2.1. Aspects With Explicit Distribution (AWED)

Distributed applications abound of crosscutting concerns, e.g., persistence and transactions in component-based enterprise information systems. However, currently there are very few approaches that address such distributed crosscutting concerns. Instead, it is common practice to use sequential aspect languages, in particular AspectJ, SpringAOP and JBoss AOP to manipulate entities provided by a distributed infrastructure, such as EJBs. This raises the important problem that aspect definitions are not declarative because they are formulated using rather low-level abstractions and make extensive use of the base-program and aspect-internal state.

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. The language has been implemented as part of a cooperation with SSEL group from Vrije Universiteit Brussel as a distributed extension of JAsCo, a dynamic weaver for sequential Java programs [34], [44].

In a second step we have extended the language by mechanisms for the composition of synchronous or asynchronous remote advice and different parameter passing modes. Dependencies between different advice that are part of one composition are managed using futures, while parameter passing modes are implemented using AWED aspects [35].

6.2.2.2. Aspect Mining and Desin Patterns Aspectualisation

Design patterns are a powerful means to understand, model and implement OO micro-architecures. Their composition leads to architectures with interesting properties in terms of variability and evolvability. However, patterns are difficult to track, modularize and reuse as thir 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 provide new insights on the expression of design patterns. We have first investigated characteristic features of some well-known design patterns to see how aspects deal with their representation. Then we have studied concrete pattern compositions in the JHotDraw framework, how AspectJ' aspects help to express them and how interactions can be analyzed [17], [26].

6.2.2.3. Aspects for System-Level 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).

Concretely, we have extended Arachne's aspect language by sequences over C function calls, accesses to global variables and local aliases of global variables [18]. Using this aspect language sequence aspects allow matching of steps in a sequence to depend on predicates over values from previous steps. Furthermore, advice may be executed at arbitrary steps of the sequences. We have demonstrated that sequence aspects enable the modularization of different typical crosscutting functionalities, in particular, protocol transformations, bug correction (e.g., for security purposes), and prefetching introduction in web caches. We have implemented this aspect language as part of the Arachne system and shown that this implementation typically results in non-perceptible to negligible performance overhead [19].

If aspect weaving at run time as provided by Arachne has proven to be an effective means of implementing evolution of C-based software systems, it often happens that the execution of the base program prior to the aspect injection influences the aspect code to be applied later. In such situation, the developer has to "bullet proof" his code. Such system development results in complex aspect code that is often poorly modularized and not reusable. We have presented a reflective extension of Arachne's aspect language to tackle this problem. As illustrated by means of a deadlock detection aspect, this extension improves the modularization of crosscutting concerns and the reusability of aspects [33].

Furthermore, Mario Südholt has co-edited an overview of AOP for systems software and middleware and the corresponding open research issues that has been published as part of a special edition of the journal Transactions of AOSD [11].

6.2.2.4. Aspects for Grid Computing

Developing concurrent, parallel and distributed services is a tedious task, even more so in the context of grid computing. Besides our work on aspects for general distributed systems, we have investigated the use of virtualization techniques for the modularization of two complex services for grid computing: execution replay and power management.

Debugging grid systems is complex, mainly because of the probe effect (i.e., monitoring a system changes the behaviour of that system) and non-reproducible executions (due to non-determinism inherent in grid-based applications). In [32], we have proposed to design a replay execution debugger using a virtual machine approach.

The direct and indirect costs induced by a grid's electric consumption constitute a factor of increasing importance that limits the feasability of grid-based solutions. However, we have observed that grids typically do not constantly run at peak performance, thus paving the way for more power-efficient scheduling of activities on grids. To reduce overall grid power consumption, we have proposed a workload concentration strategy to shutdown nodes that are unused [30]. Technically, we have implemented a power management policy that allows to transparently and dynamically dispatch applications in the grid using the Xen virtual machine migration technology.

Because system performance is critical to grids, it is necessary to add or retract dynamically such services. For this reason, we have started experiments to apply the Arachne aspect system to grid computing services, thus leveraging to grids our results on the dynamic adaptation of SQUID — the most popular free software web cache — for modification of its internal cache strategies, such as prefetching [19], [18] and the robust Arachne prototype (publicly available at:http://www.emn.fr/x-info/arachne [73]).

6.2.2.5. Aspect for Self-Adaptive Applications.

In order to support self-adaptive applications, we have developed an aspect-oriented approach to support spatial and temporal modularization of adaptation logics in component-based applications [23], [16]. Concretely, we have proposed *Safran*, an extension of the Fractal component model for the development of adaptation aspects as reactive adaptation policies. These policies detect evolutions of the execution context and adapt the base application by reconfiguring it. This way, Safran allows the development of adaptation aspects in a modular way and supports their dynamic weaving into applications.

Moreover, we have also presented the relevance of AOSD for current industrial component platforms, in particular in dynamically adapting contexts, to the general public [45].

6.2.3. Miscellaneous

To conclude the presentation of our research results on AOSD, let us mention that many of these results have been integrated in the MSc program "European Masters in Object-Oriented Software Engineering". Concretely, our recent results on EAOP and distributed aspects with AWED have been presented as part of a curriculum developed within the European Network of Excellence in AOSD, AOSD-Europe. This curriculum has been accepted for publication in a special issue on software engineering curriculum development of the journal IEEE Software [15].

6.3. DSL

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

Participants: Gilles Muller, Jean-Marc Menaud, Julia Lawall, Yoann Padioleau, Christophe Augier, Nicolas Loriant, Richard Urunuela, Fabien Hermenier.

6.3.1. Scheduler Policies

The Bossa framework (DSL, compiler, run-time system) has been publicly available for 4 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 has been done in the context of the CORSS ACI in cooperation with M. Filali and J.P. Bodeveix (project FERIA/SVC, Toulouse). We are verifying Bossa properties using formal tools such as B and FMona [21], [22].

Finally, we have investigated the extension of Bossa 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 [41].

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 done a qualitative and quantitative assessment of the collateral evolution problem in Linux device driver code [39]. We have provide a taxonomy of evolutions and collateral evolutions, and have 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 contex 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 comparision to patches that are the current solution for transmitting evolutions in Linux, our specifications are generic and can be applied to all files envolved 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 recently designed the SmPL[2] DSL [38], [37] for writing semantic patches and a prototype tool for applying them to Linux device drivers. In the current state of our work, SmPL allows us to write semantics patches for about 2/3 of the collateral evolutions that we have studied in detail in [39]. Our prototype tool allows a real scale evaluation of our approach on Linux drivers. Our first results demonstrate the conciseness of semantic patches which for our examples are up to 343 times smaller than the total size of equivalent driver patches. This shows that collateral evolutions can be documented in short and comprehensive specifications.

7. Contracts and Grants with Industry

7.1. Siemens AG, Munich

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

[2] SmPL is the acronym for Semantic Patch Language and is pronounced "sample" in Danish, and "simple" in French.

The *Aspects for toll systems* contract explores the use of aspects with explicit distribution (AWED, described in 6.2, 5.2) 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 and has included it 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 5.4) 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.

Service Oriented Architectures aim to deliver agile service infrastructures. In this context, solutions to specify service compositions (mostly BPEL language) and Quality of Service (QoS) of individual services have emerged. However, architects still lack adapted means to specify and implement QoS in service compositions.

F. Baligand's PhD work aims to overcome this shortcoming by introducing both a new language and tool for QoS specification and implementation in service compositions. More specifically, our language is a declarative DSLthat allows the architect to specify QoS constraints and mechanisms in Web Service orchestrations. Our tool is responsible for the QoS constraints processing and for QoS mechanisms injection into the orchestration.

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

7.4. France Télécom R & D PhD about DSL and the Fractal Architecture

Participants: Thomas Ledoux, Marc Léger.

M. Leger's PhD work aims to propose a model and a DSL allowing reliable reconfiguration of Fractal components architecture. The results will be used in the Selfware national project (see further 8.2) as a self-healing foundation of 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. COM project

The OBASCO team participates in the COM project funded from 2000 to 2006 by the *Pays de la Loire Council* to promote research in computer science in the region in particular via the creation of LINA (*Laboratoire d'Infomatique de Nantes Atlantique*), a common laboratory (CNRS FRE 2729) between University of Nantes and École des Mines de Nantes. The main purpose of this project was to launch the software engineering cluster grouping together the Coloss, Modal and Obasco teams.

8.1.2. 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 to fhe 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.

8.1.3. 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 September 2004 for 30 months and a budget of 78 KEUR. Its objective is to explore and design new development tools for programming computationalgrids. 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 futur 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 [65].

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 telecomunication 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. Action incitative CORSS

Participants: Gilles Muller, Julia Lawall.

The aim of this first research action, funded by the French ministry of research and started in October 2003, is to establish a cooperation between research groups working in the domains of operating systems and formal methods. Our goal is to study methods and tools for developing OS services that guarantee by design safety and liveness properties. Targeted applications are phone systems, kernel services, and composition of middleware services. Our specific interest is in verifying Bossa properties using formal tools such as B and FMona [21], [22].

Our partners are the FERIA/SVF project at the University Paul Sabatier (coordinator), the INRIA Arles project, the INRIA's Phoenix team, and the LORIA Mosel project.

8.2.3. Action incitative DISPO

Participants: Jacques Noyé, Hervé Grall, Jean-Claude Royer, Mario Südholt, Sebastian Pavel.

The aim of this second research action, funded by the French ministry of research and started in October 2003, is to contribute to the design and implementation of better component-based software in terms of security and more precisely service availability. This will be based, on the one hand, on formalizing security policies using modal logic (*e.g.*, temporal logic or deontic logic), and, on the other hand, on modular program analysis and program transformation techniques making it possible to enforce these possibilities. We are in particular interested in considering a security policy as an aspect and using aspect-oriented techniques to inject security into components implemented without taking security into account (at least in a programmatic way).

Our partners are the FERIA/SVF project at the University Paul Sabatier (Toulouse), the INRIA Lande team (coordinator), and the RSM team of the ENSTB (*École Nationale Supérieure de Télécommunications de Bretagne*).

8.2.4. ANR/RNTL Selfware

Participants: Thomas Ledoux, Pierre Cointe, Pierre-Charles David.

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 ObjectWeb context with the Fractal component model (see http://www.objectweb.org).

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

8.2.5. 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 developped 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 developpement 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.

8.2.6. ANR non thématique Coccinelle

Participants: Gilles Muller, Julia Lawall, Yoann Padioleau, Pierre Cointe.

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 Copenhaguen.

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

Participants: Pierre Cointe, Gilles Muller.

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 parameterized 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.

8.3. European Projects

8.3.1. NoE AOSD-Europe

Participants: Pierre Cointe, Rémi Douence, Jacques Noyé, Mario Südholt, Didier Le Botlan.

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.

8.3.2. STREP AMPLE

Participants: Pierre Cointe, Jacques Noyé, Jean-Claude Royer, Mario Südholt.

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 artefacts 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.

The main contribution of OBASCO is to provide its expertise in aspect oriented languages, in particular the AWED and Baton prototypes.

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. Associated Teams

8.4.1. OSCAR project

Participants: Pierre Cointe, Jacques Noyé, Eric Tanter.

The collaboration OSCAR (*Objets et Sémantique, Concurrence, Aspects et Réflexion*) project (see http://www-sop.inria.fr/oasis/oscar) groups together the DCC (Universidad de Chilem Santiago), the OASIS project-team (Sophia) and OBASCO. Its aim is to share the know-how of the members in the subjects of meta-object protocols, concurrent and distributed programming, verification of distributed systems. After Santiago (2004), Sophia Antipolis (2005), a third wokshop has been held in Valparaiso on december 2006 around J. Bustos' thesis defence (P. Cointe beeing one of the reviewer).

9. Dissemination

9.1. Animation of the community

9.1.1. Animation

ECOOP 2006: OBASCO has organized the 20th European Conference on Object-Oriented Programming (ECOOP) in Nantes[3]. ECOOP is one of the two main scientific events in the domain of object orientation (the other being its North American counterpart OOPSLA). We welcomed 450 attendees for a week (July 3-7). The organization committee included, among others, P. Cointe (general co-chair), M. Südholt (workshops co-chair), T. Ledoux (tutorials co-chair) and D. Le Botlan (Cyber Chair).

Software Composition 2006: Mario Südholt prepared as a general co-chair, the 5th International Symposium on Software Composition 2006 (Vienna, March 20060, a two-day satellite event of the ETAPS multi-conferences.

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 (XML, .NET, web services 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 is the vice-chair of the ACM/Sigops. J.-M. Menaud is 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 was the main organizer of ECOOP 2006 and serve(d) in the following program committee: OOPS 2006 (special technical track at the ACM Symposium on Applied Computing), SC 2006 and CARI 2006

R. Douence: He is a program committee member of the AOSD 2007 conference and the co-chair of the third edition of the JFDLPA workshop in Toulouse (see http://pop-art.inrialpes.fr/~jfdlpa07/). He served as a committee member of the workshop on *Aspects and Components Patterns for Infrastructure Software* (ACP4IS) 2006.

T. Ledoux: He was a program committee member of CFSE 2006 (*Conférence Française en Systèmes d'Exploitation, Perpignan October 2006*) and JC 2006 (*Journées Composants*, Perpignan October 2006). He was a program committee member of the TSI journal dedicated to *Adaptation et gestion du contexte*).

J.-M. Menaud: He is a member of the (RenPar/CFSE/Sympa) steering committees. He was a program committee member of CFSE 2006 *Conférence Française en Systèmes d'Exploitation, Perpignan October 2006*).

G. Muller: He was a program committee member of Eurosys 2006 (1rst ACM SIGOPS/EuroSys chapter Conference, April 2006, Leuven, Belgium), ISORC 2006 (*9th IEEE International Symposium on Object-oriented Real-time distributed Computing April 2006, Gyeongju, Korea*), DSN/DCCCS 2006 (*International Conference on Dependable Systems and Networks*, June, 2006, Philadelphia, USA), SRDS 2006 (*IEEE Symposium on reliable Distributed Systems*), and CFSE 2006 (*Conférence Française en Systèmes d'Exploitation, Perpignan October 2006*). He was a committee member for the jury for the 3rd best ASF French PhD in Systems. He gave two seminars on the Coccinelle project at the University of Texas in Austin, in February 2006 and at Microsoft Research in Redmond, in November 2006.

J. Noyé: He was a program committee member of JC 2006 (*Journées Composants*, Perpignan, October 2006), CAL 2006 (*Conférence francophone sur les Architectures Logicielles*, Nantes, October 2006) and DSAL 2006 (*Workshop on Domain-Specific Aspect Languages*, Portland).

[3] See http://ecoop2006.emn.fr/.

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 a program committee member of LMO 2006 (*Langages et Modèles à Objets*, Nimes), CAL 2006 (*Conférence francophone sur les Architectures Logicielles*, Nantes, October 2006) and JC 2006 (*Journées Composants*, Perpignan, October 2006).

M. Südholt: He is on the steering committee of "Software Composition", a series of ETAPS satellite events. He was the general co-chair of Software Composition 2006, organized in March 2006 in Vienna. He is co-editor of a special issue of Springer Verlag's journal *Transactions in Aspect-Oriented Software Development* on Aspects, Systems Software and Middleware Systems. He was a program committee member of the AOSD 2006 conference, of NetObject'Days 2006 and DAW 2006 (*Dynamic Aspects Workshop*, Bonn March 2006). He will serve as PC member of ECOOP 2007.

9.1.3. Thesis committees

P. Cointe: He was the reviewer of the following PhDs: A. Reilles (Institut National Polytechnique de Lorainne, Nancy, 27/11/06), S. Chantit (Université de Paris 6, 28/11/06), J. Bustos (University of Santiago du Chili and University of Nice Sophia Antipolis, Santiago du Chili, 18/12/06), S. Hong Tuan Ha (Université de Rennes,30/01/07).

T. Ledoux: He was a member of the PhD committee of R. Rouvoy (Université de Lille, 08/12/06).

G. Muller: He was a member of the PhD committee of S. Bathia (Universite de Bordeaux I, 06/06).

J.-C. Royer: He was the reviewer of the PhD of H. Fakih (Uuniversité de Lille, 12/12/06).

M. Südholt: He was a member of the PhD committee of B, Verheecke (Vrije Universiteit Brussel, 30/08/06).

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 STIC (Software Engeenering) cooperation.

G. Muller was an expert for the CNRS evaluation of the Verimag Laboratory and also for the RNTL 2006.

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 eighth promotion graduated in August 2006 while the ninth promotion was about to start their first semester. See also: http://www.emn.fr/emoose.

OBASCO (along with its partners from VUB) was mandated by the Network of Excellence in AOSD to extend EMOOSE by the year 2006 by an AOSD-centric specialization giving rise to an "AOSD minor" qualification within EMOOSE. In 2006, OBASCO members supervised one EMOOSE MSc theses by Agel Núñez [47].

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.

In 2006, the OBASCO team welcomed student interns from the ALD master for their master thesis: F. Hermenier [46] and A. Ozsemerciyan [48].

9.3. Collective Duties

P. Cointe: He is chairman of the Computer Science Department at EMN, the co-chairman of the *Laboratoire Informatique de Nantes Atlantique* (LINA-CNRS FRE 2729) 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*.

10. Bibliography

Major publications by the team in recent years

[1] M. AKSIT, A. BLACK, L. CARDELLI, P. COINTE, ET AL.. *Strategic Research Directions in Object Oriented Programming*, in "ACM Computing Surveys", vol. 28, n^{o} 4, December 1996, p. 691-700.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[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] 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.

[8] F. MÉRILLON, L. RÉVEILLÈRE, C. CONSEL, R. MARLET, G. MULLER. *Devil: An IDL for Hardware Programming*, in "Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, San Diego, California", USENIX Association, October 2000, p. 17–30.

[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 programing, systems, languages, and applications (OOPSLA 2003), Anaheim, California, USA", R. CROCKER, G. L. J. STEELE (editors). , vol. 38, n^{o} 11, ACM Press, October 2003, p. 27–46.

Year Publications

Books and Monographs

[11] Y. COADY, H.-A. JACOBSEN, M. SÜDHOLT (editors). *Special issue on AOP for systems software and middleware*, Transactions on AOSD, vol. II, Springer Verlag, September 2006.

[12] W. LÖWE, M. SUDHOLT (editors). *Software Composition*, Lecture Notes in Computer Science, 5th International Symposium, SC 2006, vol. 4089, Springer-Verlag, Vienna, Austria, 2006.

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

Doctoral dissertations and Habilitation theses

[14] G. BOBEFF. *Spécialisation de composants*, Ph. D. Thesis, École des Mines de Nantes and Université de Nantes, 14 décembre 2006.

Articles in refereed journals and book chapters

[15] J. BRICHAU, R. CHITCHYAN, S. CLARKE, E. D'HONDT, A. GARCIA, M. HAUPT, W. JOOSEN, S. KATZ, J. NOYÉ, A. RASHID, M. SÜDHOLT. *A Model Curriculum for Aspect-Oriented Software Development*, in "IEEE Software", Special issue on Software Engineering Curriculum Development, nov/dec 2006, http://www.computer.org/portal/pages/software/content/edcal.html.

[16] P.-C. DAVID, T. LEDOUX. *Une approche par aspects pour le développement de composants Fractal adaptatifs*, in "RSTI - L'objet", JFDLPA 2005, vol. 12, n^{o} 2-3, 2006, p. 113-132.

[17] S. DENIER, P. COINTE. *Expression and Composition of Design Patterns with AspectJ*, in "RSTI - L'objet", JFDLPA 2005, vol. 12, n^{o} 2-3, 2006, p. 41-61.

[18] R. DOUENCE, T. FRITZ, N. LORIANT, J.-M. MENAUD, M. SÉGURA-DEVILLECHAISE, M. SÜDHOLT. *An expressive aspect language for system applications with Arachne*, in "Transactions on Aspect-Oriented Software Development", vol. 9, 2006, p. 174-213.

[19] M. SÉGURA-DEVILLECHAISE, J.-M. MENAUD, N. LORIANT, T. FRITZ, R. DOUENCE, M. SÜDHOLT. *Dynamic Adaptation of the Squid Web Cache with Arachne*, in "IEEE Software", Special Issue on Aspect-Oriented Programming, January 2006, p. 34-41.

Publications in Conferences and Workshops

[20] F. BALIGAND, D. LEBOTLAN, T. LEDOUX, P. COMBES. *A Language for Quality of Service Requirements Specification in Web Services Orchestrations*, in "Proceeding of Second International Workshop on Engineering Service-Oriented Applications: Design and Composition, ICSOC 2006, Chicago, USA", December 2006.

[21] J.-P. BODEVEIX, M. FILALI, J. LAWALL, G. MULLER. *Automatic Verification of Bossa Scheduler Properties*, in "Sixth International Workshop on Automated Verification of Critical Systems, Nancy, France", September 2006.

[22] J.-P. BODEVEIX, M. FILALI, J. LAWALL, G. MULLER. *Vérification automatique de propriétés d'ordonnanceurs Bossa*, in "Approches Formelles dans l'Assistance au Développement de Logiciels, Paris, France", March 2006, p. 95–109.

[23] P.-C. DAVID, T. LEDOUX. *An Aspect-Oriented Approach for Developing Self-Adaptive Fractal Components*, in "International Workshop on Software Composition (SC), Vienna, Austria", W. LÖWE, M. SÜDHOLT (editors). , Lecture Notes in Computer Science, 5th International Symposium, SC 2006, vol. 4089, Springer Verlag, March 2006, p. 82-97.

[24] P.-C. DAVID, T. LEDOUX. *Safe Dynamic Reconfigurations of Fractal Architectures with FScript*, in "Proceeding of Fractal CBSE Workshop, ECOOP'06, Nantes, France", July 2006.

[25] P.-C. DAVID, T. LEDOUX. *Safe Dynamic Reconfigurations of Fractal Architectures with FScript*, in "Proceeding of Fractal CBSE Workshop, ECOOP'06, Nantes, France", July 2006.

[26] S. DENIER, P. COINTE. *Understanding Design Patterns Density with Aspects. a Case Study in JHotDraw using AspectJ*, in "International Workshop on Software Composition (SC), Vienna, Austria", W. LÖWE, M. SÜDHOLT (editors). , Lecture Notes in Computer Science, 5th International Symposium, SC 2006, vol. 4089, Springer-Verlag, March 2006, p. 243-258.

[27] R. DOUENCE. *Relational Aspects for Context passing Beyond Stack Inspection*, in "International Workshop on Software Engineering Properties of Languages and Aspect Technologies (SPLAT'06)", March 2006.

[28] R. DOUENCE, D. LEBOTLAN, J. NOYE, M. SUDHOLT. *Concurrent Aspects*, in "Generative Programming and Component Engineering (GPCE)", ACM Press, October 2006, p. 79-88.

[29] R. DOUENCE, D. LEBOTLAN, J. NOYE, M. SUDHOLT. *Towards a model of concurrent AOP*, in "International Workshop on Software Engineering Properties of Languages and Aspect Technologies (SPLAT'06)", March 2006, http://aosd.net/workshops/splat/2006/papers/sudholt.pdf.

[30] F. HERMENIER, N. LORIANT, J.-M. MENAUD. *Power Management in Grid Computing with Xen*, in "Proceedings of 2006 on XEN in HPC Cluster and Grid Computing Environments (XHPC06), Sorento, Italy", LNCS, n^{O} 4331, Springer Verlag, December 2006, p. 407–416.

[31] M. LEGER, T. COUPAYE, T. LEDOUX. *Contrôle dynamique de l'intégrité des communications dans les architectures à composants*, in "Langages et Modèles à Objets, Nimes, France", R. ROUSSEAU, C. URTADO, S. VAUTTIER (editors). , Hermès-Lavoisier, March 2006, p. 21-36.

[32] N. LORIANT, J.-M. MENAUD. *The Case for Distributed Execution Replay using a Virtual Machine*, in "Proceedings of 2006 WETICE workshop on Emerging Technologies for Grid Computing (ETNGRID06), Manchester, England", June 2006.

[33] N. LORIANT, M. SEGURA, T. FRITZ, J.-M. MENAUD. *A Reflexive Extension to Arachne's Aspect Language*, in "Proceedings of 2006 AOSD workshop on Open and Dynamic Aspect Languages (ODAL'O6), Bonn, Germany", March 2006.

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

[35] L. D. B. NAVARRO, M. SÜDHOLT, W. VANDERPERREN, B. VERHEECKE. *Modularization of distributed web services using AWED*, in "Proceedings of the 8th Int. Symposium on Distributed Objects and Applications (DOA'06)", LNCS, Springer Verlag, Oct./Nov. 2006.

[36] D. H. NGUYEN, M. SÜDHOLT. *VPA-based aspects: better support for AOP over protocols*, in "4th IEEE International Conference on Software Engineering and Formal Methods (SEFM'06)", IEEE Press, September 2006.

[37] Y. PADIOLEAU, R. R. HANSEN, J. L. LAWALL, G. MULLER. *Semantic Patches for Documenting and Automating Collateral Evolutions in Linux Device Drivers*, in "PLOS 2006: Linguistic Support for Modern Operating Systems, San Jose, CA", October 2006.

[38] Y. PADIOLEAU, J. L. LAWALL, G. MULLER. *SmPL: A Domain-Specific Language for Specifying Collateral Evolutions in Linux Device Drivers*, in "International ERCIM Workshop on Software Evolution (2006), Lille, France", April 2006.

[39] Y. PADIOLEAU, J. L. LAWALL, G. MULLER. *Understanding Collateral Evolution in Linux Device Drivers*, in "The first ACM SIGOPS EuroSys conference (EuroSys 2006), Leuven, Belgium", Also available as INRIA Research Report RR-5769, April 2006, p. 59-71.

[40] P. POIZAT, J.-C. ROYER, G. SALAUN. *Bounded Analysis and Decomposition for Behavioural Description of Components*, in "Proceedings of FMOODS", Lecture Notes in Computer Science, vol. 4037, Springer-Verlag, 2006, p. 33-47.

[41] R. URUNUELA, J. LAWALL, G. MULLER. *Energy Adaptation for Multimedia Information Kiosks*, in "International Conference on Embedded Software, EMSOFT'06, Seoul, South Korea", S. MIN, W. YI (editors). , ACM, October 2006.

Internal Reports

[42] S. D. DJOKO, R. DOUENCE, P. FRADET, D. L. BOTLAN. *CASB: Common Aspect Semantics Base*, Research Report, n^{o} D54, Network of Excellence in AOSD (AOSD-Europe, August 2006.

[43] R. DOUENCE, D. LEBOTLAN, J. NOYE, M. SUDHOLT. *Concurrent aspects*, Research Report, n^{o} RR-5873, INRIA, March 2006, https://hal.inria.fr/inria-00071396.

[44] L. D. B. NAVARRO, M. SÜDHOLT, W. VANDERPERREN, B. DE FRAINE, D. SUVÉE. *Explicitly distributed AOP using AWED*, Slightly extended version of [Benavides et al., AOSD'06], Research Report, n^{o} 5882, INRIA, March 2006, https://hal.inria.fr/inria-00071386.

Miscellaneous

[45] 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.

[46] F. HERMENIER. *Gestion dynamique des grilles de calcul : Application à la gestion d'énergie.*, ALD master thesis, Technical report, EMN et Université de Nantes, September 2006.

[47] A. N. LÓPEZ. *Concurrent Aspects*, EMOOSE master thesis, Technical report, VUB and EMN, August 2006.

[48] A. OZSEMERCIYAN. *Passage de contexte par aspects*, ALD master thesis, Technical report, EMN et Université de Nantes, September 2006.

References in notes

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

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

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

[61] 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.

[62] G. KICZALES, J. LAMPING, A. MENDHEKAR, C. MAEDA, C. LOPES, J.-M. 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.

[63] 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.

[64] 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.

[65] 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.

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

[67] 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.

[68] 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.

[69] 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.

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

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

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

[73] 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.

[74] M. SÉGURA-DEVILLECHAISE. *Traitement par aspects des problèmes d'évolution logicielle dans les caches Webs*, Ph. D. Thesis, École des Mines de Nantes and Université de Nantes, July 2005.

[75] 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.

[76] 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.

[77] 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.

[78] 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.