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, École des Mines de Nantes (EMN)]

Staff Member INRIA

Mario Südholt [CR2 INRIA on leave from EMN]

Faculty Members from EMN

Rémi Douence [Associate Professor]
Thomas Ledoux [Associate Professor]
Jean-Marc Menaud [Associate Professor]
Gilles Muller [Professor]
Jacques Noyé [Associate Professor]
Jean-Claude Royer [Professor]

Administrative Assistant

Isabelle Gonzales [Part-time (50%), since September]

Ph. D. Students

Ali Assaf [MESR grant, since October]
Christophe Augier [Cifre grant with JANULA]
Luis Daniel Benavides Navarro [MINES grant, since October]
Gustavo Bobeff [MINES contract, until October]
Pierre-Charles David [MESR grant, until June]
Simon Denier [EMN grant]
Simplice Djoko Djoko [INRIA grant shared with the project PopArt, since October]
Ha Dong Nguyen [REGIONAL COUNCIL & AOSD-Europe grant, since April]
Hervé Duchesne [MINES grant, until June]
Nicolas Loriant [EMN grant]
Florian Minjat [MESR grant]
Sebastian Pavel [MESR grant, ACI DISPO]
Marc Ségura-Devillechaise [ATER Nantes University, until August]
Richard Urunuela [REGIONAL COUNCIL grant]

Visiting Scientist

Julia Lawall [DIKU, UNIVERSITY OF COPENHAGEN, June to July 2005]

Temporary Faculty Members

Hervé Albin-Amiot [On leave from SODIFRANCE until August]
Didier Le Botlan [CNRS & REGIONAL COUNCIL postdoctoral grant]
Hervé Grall [MINES grant]
Yoann Padioleau [MINES grant]

2. Overall Objectives

2.1. Overall Objectives

OBASCO addresses the general problem of adapting software to its uses by developing tools for building software architectures based on components and aspects [57]. We are (re)using techniques developed in the programming languages, in particular object-oriented languages, arena.

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 are working along four directions:

Component-Oriented Programming: Definition of a language making it possible (i) to program components by explicitly representing their composition both at the structural and behavioral level, (ii) to manage their adaptation all along their life cycle. To this aim, we are relying on reflection and specialization techniques. 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: Formalization of aspect-oriented programming based on the concepts of event, trace, and monitor. Implementation of a corresponding language using reflection, as well as program analysis and transformation techniques.

Post Object-Oriented Programming: Contribution to the evolution from an object model to a unified model supporting programming in the large and adaptation through reflection. Study of the problems resulting from integrating objects and aspects on the one hand, objects and components on the other hand.

Applications: 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, to middleware and business components.

3. Scientific Foundations

3.1. Introduction

The OBASCO project was created in 2003. Its primary goal is to investigate the possibility of a *continuum* between objects, aspects and components [57]. We plan to study formal models of components and aspects to reason about adaptable systems. A natural result of our research is the implementation of prototypes based on the investigation into new programming languages and paradigms suited to component-oriented systems with a particular emphasis on metaprogramming.

Historically the core members of OBASCO have a strong background in the design and implementation of (reflective) object-oriented languages [1], [3], [10], [14]. This background has been enriched by an expertise in operating systems and middleware [8], [7]. Our goal is to take advantage of this complementarity by developing a methodology and a set of tools covering in a uniform way the software process from “OS to applications”.

3.2. 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 [74]).

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* [70].

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

Reflection: *A process’s integral ability to represent, operate on, otherwise deal with itself in the same way that it represents, operates and deals with its primary subject matter* [69].

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 programming in the large, in turn, has revealed some 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

In spite of its successes, the generalization of object-oriented languages has failed[1] to greatly improve software reusability. This is due to the inherent difficulties of a *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...), becomes a fundamental issue. Taking these issues into account has led to *components* [70]. The basic idea, as initially explained by M.D. McIlroy in 1968 [64] 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*.

These properties raise new challenges in programming language design and implementation. They require an integration of ideas coming from module interconnection languages [67], architecture description languages (ADLs) [68], [65] and object-oriented languages. Modules provide an interesting support for component structure. In particular, recent proposals around so-called *mixin modules* combine parameterization, recursive module definitions, and late binding. ADLs address many of the above-mentioned issues although at a description, 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* [70] 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,

[1] See the discussions at http://www.dreamsongs.com/Essays.html and those at http://www-poleia.lip6.fr/~briot/colloque-JFP/, in particular [57].

and, on the other hand, the possibility to adapt a component all along its life cycle. Software specialization (*e.g.*, partial evaluation [61]) and reflection have a key role to play here.

3.2.2. From Objects to Aspects

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 [73][22]. Aspect-oriented programming as well as aspect-oriented modeling is an extremely competitive field of research where people try to go beyond the object model by providing:

- *abstractions for the modularization of crosscutting concerns:* These new units of independant 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 [56], [62] 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].

Model-View-Controller: The *MVC* developed for Smalltalk [60] 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 croscutting 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*.

To conclude we have identified the following main issues that OBASCO strives to address concerning the relationship between computational reflection and aspects [22]. 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], [43]. A second issue is to study the integration of aspects and objects to propose an alternative to inheritance as a mechanism for reuse. 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 [23]. A fourth issue is to apply domain-specific languages to the expression of aspects.

3.3. Domain-Specific Languages

DSL: A domain-specific language (DSL) is a programming language dedicated to a particular application domain, for instance the implementation of drivers.

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.* termination) [8] [72].

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.

4. Application Domains

4.1. Overview

Keywords: *enterprise information systems, telecommunication.*

The goal of our research is to develop new methodologies based on the use of aspect-oriented programming and components languages for developing adaptable and composable software architectures. We plan to apply those methodologies and the associated tools in a systematic and uniform way from the OS to the enterprise applications. We are currently working in the OS field to express process scheduling extension, Web caches to dynamically adapt cache prefetch strategies and middleware to dynamically adapt components behaviors to their execution context.

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 producing out components in an automatic way. With other researchers in the domain of generative programming we are investigating new methodologies, tools and applications [58].

We are applying, in particular, separation of concerns techniques - as a unified methodology - to reengineer or dynamically evolve existing complex legacy software. Our main goal is to develop new tools/methodologies based on the coupling of aspect-oriented design and domain-specific languages for the structuring of an OS kernel, an OS itself, Web caches and middleware.

4.2. Operating Systems and Networks

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 Domain-Specific Languages is a promising approach to address these issues [55], [66].

A first application direction is to use DSLs 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. The combination of aspect-oriented programming and domain-specific languages has been validated in the context of Bossa (see Section 5.1).

As a second application, we are investigating in this arena is the applicability of AOP for re-engineering or dynamically evolve existing complex system software such OS kernels and Web caches (see Section 5.3).

4.3. 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. Since an increasing number of devices are participating in a global information network, mobility and dynamic reconfiguration will be dominant features, requiring permanent adaptation of the applications. For example, component-based applications working in highly dynamic environments, where resource availability can evolve at runtime, have to fit their dynamic environment.

To adress this challenge, we propose that these applications must be self-adaptive, that is adapt themselves to their environment and its evolutions. We consider adaptation to a specific execution context and its evolutions as a aspect which should be treated separately from the rest of an application and should be expressed with a DSL. A first application is the expresssion of adaptation policies to adapt Fractal components (see Section 5.5).

We have also explored the application of dynamic aspect weaving of the image generation process of medical scanners of Siemens AG, Munich. [31], [30]. The adaptation of such software systems, which are implemented using C++, is subject to very similar problems, such as the difficulty of declarative definitions of dynamic adaptations, as typical middleware applications.

5. Software

5.1. Bossa

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

Participants: Gilles Muller [correspondent], Christophe Augier, Hervé Duchesne, 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. 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 [39], [40], [32], [51]. 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 [32].
- 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 Jaluna (formerly Chorus Systems). 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 [44].

5.2. EAOP

Keywords: *CPS, Java, Javassist, Recoder.*

Participants: Mario Südholt [correspondent], Rémi Douence.

Definition of expressive aspect languages is a major research issue of AOP. However, none of the common approaches to AOP allows for the flexible and powerful definition of new language constructs.

We have developed the Event-based Aspect-Oriented Programming model (EAOP) as a general-purpose, well-defined support for AOP with the following characteristics:

- Execution events (*e.g.*, method calls) represent points of interest of an application. Crosscuts, defined as *sequences* of events, explicitly represent relationships between points of interest.
- Aspect weaving is defined by an execution monitor, that triggers an action when a crosscut (*i.e.*, sequence of events) is detected. EAOP aims at a simple and intuitive semantics for applications, so we use a fully synchronous event model. On event emission the base program suspends its execution, yields control to the monitor, which in turn yields it to the aspects. After all actions have been applied, control returns to the base program.
- Two aspects interact when two actions are triggered at the same point of interest. In order to support conflict resolution, our model makes composition explicit through a tree whose nodes are composition operators and leaves are aspects. Such operators realize aspect compositions by controlling event propagation in the tree of aspects.
- EAOP supports an arbitrary number of aspect instances at run time. Aspect instances may be created and composed dynamically. Composition operators are responsible for dynamically creating aspects and inserting (*i.e.*, composing) them in the aspect tree.
- Aspects may be applied to other aspects and not only the base program: the monitor is re-entrant. In this case, composition operators can filter events to be propagated and thus define scope of aspects.

We have implemented in Java the EAOP tool as a testbed for the definition of expressive aspect languages. Aspect composition can be performed in EAOP by means of expressive and powerful composition operators. In particular, composition operators can be used for resolution of aspect interactions, for aspect instantiation, and for definition of aspects of aspects (see also Section 6.2).

The base program to be woven by our EAOP tool can be either Java source code (by instrumentation with the transformation tool *gv Recoder* http://sourceforge.net/projects/recoder) or Java byte code (by instrumentation using *Reflex*). Conceptually, aspects run in parallel with the base program. We have investigated two implementations: a general one based on threads and an optimized one based on continuations. The distribution of the EAOP tool is publicly available at http://www.emn.fr/x-info/eaop. EAOP is used for master lectures at EMN.

5.3. Arachne

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

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

We have have developed Arachne, an AOP-based software that permits to dynamically evolve a system at runtime without interrupting servicing. It is developed toward changing prefetching policies in Web caches and security update 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. That's why we have implemented Arachne, principally as part of Marc Ségura's PhD thesis [13], as an AOP-based software that permits to dynamically evolve a system at runtime without interrupting servicing.

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 [71], 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 three directions : i) supporting the introduction of new pointcut [28], [16], ii) moving from C to C++ weavers [31], iii) mixing aspects and CCM components for Grid Computing [37]. We also have further explored applications of Arachne to web caching [19] and have applied Arachne to the adaptation of medical image processing for scanners from Siemens AG, Munich [31], [30].

A prototype of Arachne is publicly available at http://www.emn.fr/x-info/arachne/.

5.4. 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

5.5. 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. It was designed and implemented by Pierre-Charles David during his PhD thesis [12]. 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 is a simple scripting language used to program the component reconfigurations which will adapt the application. Its design and implementation offer 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 [26]. 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.

A prototype of Safran is available at http://www.emn.fr/x-info/obasco/tools/safran.

6. New Results

6.1. Components

Keywords: *adaptation, communications, components, composition, encapsulation, interaction, interfaces, life cycle, modules, objects, protocols, services, specialization.*

Participants: Gustavo Bobeff, Pierre-Charles David, Hervé Grall, Thomas Ledoux, Jacques Noyé, Sebastian Pavel, Jean-Claude Royer, Mario Südholt, Marc Léger.

At the theoretical level, we study the introduction of explicit interaction protocols, and property checking. On a more practical side, we work on Java extensions to support these features, and on techniques to better adapt component-based applications to their environment or to dynamically configure them in a safe way.

J. Noyé, R. Douence, and M. Südholt have published a comprehensive presentation of the current state-of-the-art of the issues raised by crosscutting functionalities in component-based models [18]. This survey presents, in particular, (i) a detailed introduction to aspect-oriented notions, (ii) the descriptive means for the definition of aspects (persistence, transactions and security) in industrial-strength component frameworks, and (iii) current research approaches for the integration of aspects and components.

6.1.1. Explicit Protocols

Component-Based Software Engineering (CBSE) has now emerged as a discipline for system development. Explicit behavioural protocols are now recognized as a mandatory feature of components to address architectural analysis. An important issue is to fill the gap between high-level models (needed for analysis) and

implementation. S. Pavel has started work on a Java library to support the use of STS and asynchronous communications. In [41] we describe a component model with explicit symbolic protocols based on Symbolic Transition Systems (STSs), and its implementation in Java. This implementation relies on controllers that encapsulate protocols and channels devoted to (possibly remote) communications between components.

M. Südholt has investigated how to use a notion of non-regular protocols originally developed by Franz Puntigam in the context of software compnents [42]. Concretely, he has defined a protocol language for this class of protocols which allows the expression of interesting component collaborations, in particular, in the context of peer-to-peer applications. Furthermore, he has discussed to what extent common compositional properties are preserved by this notion of protocols.

6.1.2. Property Checking

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. H. Grall has studied an extension of security automata, as defined by Schneider. A security automaton monitors the execution of a program: an event generated by the program is either accepted by the automaton and then executed, or refused, which entails exiting the program. In this standard form, a security automaton enables to enforce a safety property. Since availability properties typically have a liveness part, this class of security automata is too restricted. We have therefore defined an extension of this class, by allowing automata to perform edit actions and transactions. This work is a generalization of the recent work of Bauer, Ligatti and Walker about "edit automata".

The boundedness of Symbolic Transition System 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. We have also investigated the PVS specification generation from our STS.

6.1.3. Dynamic Adaptation

The increasingly diverse and dynamic contexts in which current applications are run imposes them to adapt and to become more autonomous. To this end we propose Safran, an extension of the Fractal component model enabling dynamic association of adaptation policies to the components of an application [12]. The adaptation policies execution harnesses WildCAT [26], a context-awareness system which can detect changes in the execution context (when to adapt?), and FScript, a language dedicated to dynamic and consistent reconfigurations of Fractal components (how to adapt?). Finally, Safran provides an infrastructure to develop self-adaptive Fractal components.

6.1.4. 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 [50].

6.2. Aspects

Keywords: *Arachne*, *EAOP*, *Reflex*, *expressive aspect languages*, *separation of concerns*.
Participants: Mario Südholt, Rémi Douence, Pierre Cointe, Jean-Marc Menaud, Jacques Noyé, Nicolas Loriant, Marc Ségura-Devillechaise, Simon Denier, Thomas Ledoux, Pierre-Charles David, Daniel Benavides.

Join points: well defined points in the execution of a program.

Pointcuts: means of referring to collections of join points and certain values at those join points (in order to executed associated advice at these join points).

Crosscut: specification of (sequences of) join points where aspect actions should be woven in a base program.

Advices/actions: specification of what an aspect computes. 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...); it defines some crosscuts and actions. 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 at join points where the crosscuts of both aspects match, *i.e.*, apply at the same time.

OBASCO's work on aspect-oriented programming is targeted towards the development, support for, and application of expressive aspect languages. This year we pursued work on the model of Event-Based AOP, which allows expressive aspects to be defined in terms of relations over sequences of execution events. As to general implementation support for aspect languages, we developed a versatile kernel based on our reflective infrastructure Reflex suitable for the realization of a wide range aspect languages. Finally, we applied several expressive aspect specific languages to different system-level applications, in particular to solve software evolution and adaptation problems.

Complementary to these technical contributions, we studied characteristics of a (still missing) comprehensive model for AOP, in particular, the relationship between generative programming and AOP [23].

6.2.1. 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 [6]. The expressive power of EAOP makes it possible to reason about events patterns, thus supporting (temporal) reasoning over AO programs. This model supports a wide range of languages used to define crosscuts and actions [5]. This year, we have started to extend EAOP for concurrent applications.

The Event-based Aspect-Oriented Programming model (EAOP) makes it possible to define pointcuts in terms of sequences of events emitted by the base program. The current formalization of the model [59] relies 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 is not intrinsically sequential but its current formalization favors a sequential point of view. This year, we have presented a new formalization of EAOP as finite state processes [29]. This new formalization paves the way to reasoning about aspects in a concurrent setting and to the definition and implementation of concurrent EAOP languages.

6.2.2. A versatile kernel for multi-language AOP

Reflex, initially a reflective extension of Java, has been evolved into a fully operational kernel for multi-language AOP. This experiment gives a first picture of what an AOP kernel may look like and of its benefits. It raises in a practical manner, the issue of determining what the building blocks of AOP are and how they can be combined in a flexible and manageable way. The architecture of Reflex consists of three layers: a transformation layer, based on reflection, in charge of basic weaving, supporting both structural and behavioral modifications of the base program; a composition layer, for load-time detection and resolution of interactions; and a language layer, for modular definition of aspect languages. This architecture is described in [43] based on a running example mixing an aspect written using AspectJ together with two other aspects written using small domain-specific aspect languages.

The mapping of AspectJ, actually a subset restricted to behavioral crosscutting, onto Reflex is presented in [17], together with benchmarks showing that the performance of the resulting system, although not as fast as a native implementation, is still reasonable. Two Master theses have also explored the use of Reflex for implementing EAOP [45] and better structuring aspects [49], using collaborations à la Caesar (http://www.caesarj.org/). This confirms the generality of the approach and provides new building blocks for implementing aspect languages using Reflex.

6.2.3. Aspects for system-level applications

We have pursued different approaches for expressive aspect languages in the context of system-level applications. We have, in particular, introduced sequence pointcuts into a dynamic weaver for C and C++ applications, have applied the resulting system to the adaptation of image generation tasks for medical devices, have introduced an aspect language for explicit distributed programming as well as context-aware applications.

Aspects to support runtime evolution 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 5.3). Concretely, we have extended Arachne's aspect language by sequences over C function calls, accesses to global variables and local aliases [28]. With 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. We have also considered two different formal semantics for sequence aspects, an abstract one defined in terms of a process calculus [28] and another transformational one which defines in detail how sequence aspects are implemented as part of the Arachne system [16], see also T. Fritz's MSc thesis [48].

We have also extended Arachne to support the deploymentof critical security updates on the fly [34], [36].

Adapting medical image generation using dynamic aspects.

In cooperation with Siemens AG, Munich, we have investigated the application of Arachne's sequence aspects (cf. the previous paragraph) to the adaptation of image generation algorithms for medical purposes. We have shown that the graph defining the valid sequences of basic image generation functors can be dynamically adapted using transformations which, e.g., insert or add new functor sequences to existing ones. Since Siemens AG's image generation library is realized as a C++-framework, we have in a first step provided a proof of concept system for the manipulation using Arachne-C [30]. In a second step, we have started to extend Arachne to C++ in order to realize the image manipulations directly on the Siemens AG's C++ legacy framework [31].

Aspects for explicit distributed programming

As part of Daniel Benavides's MSc thesis [46], we have begun work on the definition of 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 allowing the modularization of these functionalities using explicit references to hosts on which caches are executed. Concretely, we have considered three contributions: remote sequence pointcuts, remotely executed advice, and distributed aspects with corresponding deployment, instantiation and data sharing mechanisms.

Definition of the adaptation logic as an aspect for context-aware applications.

As part of the PhD thesis of Pierre-Charles David [12], we have developed an aspect-oriented approach for the development of self-adaptive applications where the adaptation logic is well modularized, both spatially and temporally. Concretely, we have proposed Safran, an extension of the Fractal component model for the development of reactive adaptation policies using *adaptation aspects*. These policies detect the evolutions of the execution context and adapt the base program by reconfiguring it. This way, SAFRAN allows the development of the adaptation aspect in a modular way and its dynamic weaving into applications [25].

6.3. Post-Objects

Keywords: *AspectJ, Eclipse, Java, Reflex, Squeak, classboxes, design patterns, generative programming, inter-type declarations, open systems, specialization, traits.*

Participants: Pierre Cointe, Jacques Noyé, Thomas Ledoux, Hervé Albin-Amiot, Simon Denier, Florian Minjat.

to reify: providing an explicit representation for a structural or runtime property of an otherwise implicit language entity.

to reflect: applying to the actual runtime system changes made to a reified representation.

spatial selection: consists in selecting what will be reified in an application.

temporal selection: consists in selecting when reifications are made (or are active).

hook: the base level piece of code responsible for performing a reification and giving control to the metaobject.

hookset: set of hooks to gather execution scattered in various objects.

We are investigating some new directions in the field of Object-Oriented languages. Our general aim is to open these languages in order to introduce extra mechanisms such as revisited encapsulation and inheritance, reflection, crosscutting aspects, design patterns reification, objects interaction and composition. This year contribution deals on : (i) mixing two models of class extensions, (ii) reporting first results in Java aspect mining, (iii) providing a better understanding of the relationship between metaprogramming, generative programming and AOP, (iiii) testing generative progamming in general and AOP in particular as new approaches to developing DSL.

6.3.1. Hybridation of Traits and Classboxes

The TRAIT model as defined in [54][15] is complementary to class inheritance and allows collections of methods to be reused by several classes. The CLASSBOX model as defined in [53] allows a collection of classes to be locally extended with variables and/or methods addition. We proposes a symbiosis of these two models for which classes can be locally extended by using a trait. It is illustrated by an efficient implementation of the collaboration model where a collaboration is represented by a classbox and a role by a trait [38].

This work was developed jointly by OBASCO and the Software Composition group at the University of Bern.

6.3.2. 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 evolutivity. However they are difficult to track, modularize and reuse as there elements tend to vanish in the code. Following the two PhD theses of H. Albin and Y-G. Guéhéneuc dedicated to the reification and analysis of design patterns, we experiment AOP modular technology to give new insights on the expression of design patterns. We first take a look at singular features of some well known design patterns to see how aspects deal with their representation. Then we study some cases of composition in the JHotDraw framework, where we analyze various interactions and the way aspects help to express them [27].

This work was done in cooperation with Sodifrance (see also Section 7.1).

6.3.3. From (meta) objects to aspects

We developed a guided tour of AspectJ illustrating by examples the new concepts of pointcuts, advices and inter-type declarations. This tour is the opportunity to discuss how the AspectJ model answers some of the issues raised by post-object oriented programming but also to enforce the relationship between reflective and aspect-oriented languages [22].

6.3.4. Towards Generative Programming

Generative Programming (GP) is an attempt to manufacture software components in an automated way by developing programs that synthesize other programs. Our purpose is to introduce the what and the how of the GP approach from a programming language point of view. For the what we discuss the lessons learned from object-oriented languages seen as general purpose languages to develop software factories. For the how we compare a variety of approaches and techniques based on program transformation and generation. On the one hand, we present the evolution of open-ended languages from metalevel programming to aspect-oriented programming. On the other hand, we introduce domain-specific languages as a way to bridge the gap between conceptual models and programming languages [23].

6.3.5. A Generative Programming Approach to Developing DSL

Domain-Speciffic Languages (DSLs) represent a proven approach to raising the abstraction level of programming. They offer highlevel constructs and notations dedicated to a domain, structuring program design, easing program writing, masking the intricacies of underlying software layers, and guaranteeing critical properties. On the one hand, DSLs facilitate a straightforward mapping between a conceptual model and a solution expressed in a specific programming language. On the other hand, DSLs complicate the compilation process because of the gap in the abstraction level between the source and target language. The nature of DSLs make their compilation very different from the compilation of common General-Purpose Languages (GPLs). In fact, a DSL compiler generally produces code written in a GPL; low-level compilation is left to the compiler of the target GPL. In essence, a DSL compiler defines some mapping of the high-level information and features of a DSL into the target GPL and underlying layers (e.g., middleware, protocols, objects, . . .). We present a methodology to develop DSL compilers, centered around the use of generative programming tools. Our approach enables the development of a DSL compiler to be structured on facets that represent dimensions of compilation. Each facet can then be implemented in a modular way, using aspects, annotations and specialization. Because these tools are high level, they match the needs of a DSL, facilitating the development of the DSL compiler, and making it modular and re-targetable [24].

This work was developed in cooperation with the project-team Phoenix (Inria Futurs).

6.4. AOP and DSLs for OS Kernels

Keywords: *C language*, *OS*, *Web caches*, *process schedulers*, *proxies*.

Participants: Gilles Muller, Jean-Marc Menaud, Julia Lawall, Hervé Duchesne, Richard Urunuela, Christophe Augier, Marc Ségura-Devillechaise, Nicolas Loriant.

6.4.1. Scheduler Policies

The Bossa framework has been publicly available for three years. It is fully compatible with the Linux 2.4 and 2.6 kernels and can be used as a direct replacement. Most of our work has been done this year on the design of a modular version of the Bossa DSL so as to express in a simpler way families of real-time scheduling policies [32], and on an evaluation of the complete framework on Linux 2.4 [51], [39], [40].

Additionaly, we have investigated the extension of Bossa to energy management [44] and the support of the multimedia application in the context of a personnal video recorder [47].

Finally, in the context of the CORSS ACI, we are verifying Bossa properties using formal tools such as B and FMona [20], [21]. This work is done in cooperation with M. Filali and J.P. Bodeveix (project FERIA/SVC, Toulouse).

6.4.2. Understanding Evolution 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. 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 [33].

6.4.3. Extensible caches and security updates

Our work on Arachne has focused on the design and implementation of a robust prototype that has been made publicly available:http://www.emn.fr/x-info/arachne [71]. We had experimenting with applying Arachne to SQUID - the most used free-software Web cache - so as to dynamically change its internal cache strategies, such as prefetching. We use also Arachne in system administration to deploy critical security updates on the fly on applications running remotely. For that Arachne takes a patch produced by diff and builds an aspect to produce a dynamic patch that can later be woven to update the application on the fly [34], [35], [36].

7. Contracts and Grants with Industry

7.1. Sodifrance/Softmaint

Participants: Pierre Cointe, Hervé Albin-Amiot, Simon Denier.

The purpose of this contract is to apply the aspect technologies to the field of reengineering legacy applications. In the continuity of Hervé Albin-Amiot's thesis our idea is to automate the detection and application of some well defined aspects such as persistency, errors handling and user interfaces. At that time we have chosen the JHotDraw framework as a first testbed to aspectualize some design-patterns when experimenting with AspectJ [27].

7.2. Jaluna Cifre grant

Participants: Gilles Muller, Christophe Augier.

Our work on the development of Bossa (see Section 5.1) is supported in part by 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 thesis grant

Participants: Thomas Ledoux, Fabien Baligand.

As Web Services spread more and more widely in industrial applications, composition and adaptation of services still raise several important issues that hinder Web Service applications use.

Fabien Baligand PhD work aims at providing business process designers with AOP-based mechanisms that would allow separating the structural concerns from the Quality of Service concerns when composing web services. Developers should be able to specify the QoS properties of web service workflows in an efficient and reusable way, using appropriate aspect and domain-specific languages. This work is supported by FT amounting to 23 KEUR.

8. Other Grants and Activities

8.1. Regional Actions

8.1.1. COM project

The OBASCO team participates in the COM project funded 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.

8.1.2. Arantèle project

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

The Arantèle project is also funded by the *Pays de la Loire* council. It started in September 2004 for 30 months and a budget of 78 Keuros. 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 5.3). 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 and C. Perez on the application of AOP to the domain of software components for Grid computing [37].

8.2. National Projects

8.2.1. ANVAR Componentifying Multi-Agent Libraries

Participants: Jean-Claude Royer, Pierre Cointe, Jacques Noyé, Thomas Ledoux, Gustavo Bobeff, Pierre-Charles David, Sebastian Pavel.

This project is funded by ANVAR via ARMINES for an amount of 19 Keuros. The participants come from the group of Écoles des Mines (Alès, Douai, Nantes and Saint-Etienne). The goal is to evaluate the "semantic gap" between components and agents. To reach it, we are investigating a common model integrating components and agents. The model will be integrated in the Eclipse IDE and used as a test-bed to re-implement a library for multi-agent previously developed by the team from Saint-Etienne. The project started in 2003 with a general state of the art about component and agent technologies. On this basis, we elaborate the general principles of a component model with asynchronous communications, hierarchical components and explicit dynamic behavior (see also Section 6.1).

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 [20], [21].

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, Sebastian Pavel, Jean-Claude Royer, Hervé Grall, Mario Südholt.

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, Jacque Noyé, Jean-Claude Royer, Pierre Cointe.

The Selfware project is an ANR/RNTL project running for 30 months which has been submitted and accepted in 2005 for funding amounting to 212 KEUR from January 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 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 265 KEUR from Januar 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.

8.3. European Projects

8.3.1. NoE AOSD

Participants: Pierre Cointe, Mario Südholt, Jacques Noyé, Rémi Douence, 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 million euros. OBASCO's share amounts to 200 Keuros.

8.4. Associated Teams

8.4.1. OSCAR project

Participants: Pierre Cointe, Jacques Noyé, Jean-Marc Menaud, 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 & distributed programming and verification of distributed systems. A second wokshop has been held in Sophia Antipolis on october 2005 around the CoreGrid week.

9. Dissemination

9.1. Animation of the community

9.1.1. Animation

ACM/Sigops: G. Muller is the vice-chair of the ACM/Sigops, and was the chair of the French Sigops Chapter (ASF) until April 2005.

ECOOP 2006: OBASCO has started co-organizing the 20th European Conference on Object-Oriented Programming (ECOOP) in Nantes in 2006, see http://ecoop2006.emn.fr/. ECOOP is one of the two main scientific events in the domain of object orientation (the other being its North American counterpart OOPSLA). The organization committee includes, among others, Pierre Cointe (general co-chair), Mario Südholt (workshops co-chair), Thomas Ledoux (tutorials co-chair) and Gilles Muller (sponsors and industrial chair).

Software Composition 2006: Mario Südholt is preparing as a general co-chair, the 5th International Symposium on Software Composition 2006, a two-day satellite event of the ETAPS multi-conferences. Software Composition 2006 will take place in Vienna in March 2006.

Rencontres Francophones en Parallélisme, Architecture, Système et Composant: This event (http://www.emn.fr/x-info/renpar2005/) colocates together four conferences on parallelism (RenPar), machine architecture (SympAA), systems (CFSE), and component programming (JC). It has beeen organized by Gilles Muller (general chair), Jean-Marc Menaud and Thomas Ledoux in Le Croisic.

EIWAS 2005: Rémi Douence has co-organized the European Interactive Workshop on Aspects (EIWAS'05). This 2-day forum organized at Vrije Universiteit Brussel has welcomed around 25 participants in September. See http://prog.vub.ac.be/events/eiwas2005/ for more details.

AOMD 2005: Rémi Douence co-organized the First Workshop on Aspect-Oriented Middleware Development (AOMD'05). This forum will be held as a satellite event to Middleware'05 in Grenoble in November. See http://www.lifl.fr/~pawlak/aomd/ for more details.

Eclipse Day 2005: Obasco has organized under the auspice of the AOSD network the first French Eclipse Day (Nantes, March 17th). This industrial conference dedicated to the presentation of the Eclipse platform welcomed 150 participants. Erich Gamma (IBM Zurich), Krysztof Czarnecki (Waterloo University) gave the two keynotes while Philippe Mulet (IBM St Nazaire), Jean Bézivin (INRIA Atlas), Helen Hawkins and Catherine Griffin (IBM Hursley) presented Java 5, AMMA, AspectJ and EMF. For more details, see http://www.emn.fr/x-di/eclipse-day/

Les jeudis de l'objet: This bimonthly industrial seminar organized by our group is now eight 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.

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 is a program committee member of the OOPS 2005 and 2006 special technical tracks at the ACM Symposium on Applied Computing. He is a program committee member of the Software Composition workshops organized in the context of ETAPS 2005 in Edinburgh and 2006 in Vienna. He was a program committee member of the JFDLPA 2005 workshop and the editor of a selection of papers presented at JFLDPA 2004 [11].

R. Douence: He was on the program committees of the JFDLPA 2005, EIWAS 2005, and AOMD 2005 workshops.

T. Ledoux: He has been the program chairman of JC 2005 (*Journées Composants*, Le Croisic, April 2005).

G. Muller: Gilles Muller was the publicity chair of Middleware 2005.

He is a program committee member of ICDCS 2005 (IEEE International Conference on Distributed Systems, June 2005 Columbus, Ohio, USA), ISAS 2005 (Second International Service Availability Symposium, April 2005, Berlin), ICPP 2005 (34th Annual Conference on Parallel Processing, June 2005, Oslo, Norway), HASE 2005 (9th Intl. Symposium on High Assurance Systems Engineering, October 2005, Heidelberg, Germany), CFSE'4 (4th French Conference on Operating Systems, April 2005, Le Croisic). He was the program chair of RENPAR 2005 (16ème édition des Rencontres Francophones du Parallélisme, April 2005, Le Croisic).

He is 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) and DSN/DCCCS 2006 (International Conference on Dependable Systems and Networks, June, 2006, Philadelphia, USA).

J. Noyé: Jacques Noyé has been a program committee member of JC 2005 (*Journées Composants*, Le Croisic, April 2005).

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 2005 (*Langages et Modèles à Objets*, Berne, March 2005) and JC 2005 (*Journées Composants*, Le Croisic, April 2005).

M. Südholt: Mario Südholt is on the steering committee of "Software Composition", a series of ETAPS satellite events. He is general co-chair of Software Composition 2006, which will take place 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 AOSD 2005 (Aspect Oriented Software Design, Boston, March 2005) and will serve on that for AOSD 2006. He also served on the program committee of NetObject'Days 2005. He served as a program committee member of the workshops DAW at AOSD 2004, and SC at ETAPS 2005. He was the general chair of the scientific workshop of AOSD-Europe, the European Network of Excellence in AOSD, which took place in July 2005 in Glasgow.

9.1.3. Thesis committees

P. Cointe: He is a member of the PhD jury of D. Boinnot (Ecole des Mines de Paris, 18/11/05).

T. Ledoux: He was the scientific advisor of Pierre-Charles David [12] (University of Nantes, 01/07/05). He is a member of the PhD jury of O. Barais (University of Lille, 29/11/05).

J.M. Menaud: He was the scientific advisor of Marc Ségura-Devillechaise [13] (University of Nantes, 01/07/05).

G. Muller: He was the reviewer of the PhD of Arnaud Albinet (Institut National Polytechnique de Toulouse), Damien Deville (U. of Lille), Gaël Thomas (U. of Paris 6) and Yoann Padioleau (U. of Rennes 1). He is a member of the PhD jury of V. Quema (U. of Grenoble).

M. Südholt: He was a reviewer of the PhD of Wim Vanderperren (Vrije Universiteit Brussel).

9.1.4. Evaluation committees and expertise

Pierre Cointe has been 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.

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 seventh promotion graduated in August 2005 while the eighth promotion was about to start their first semester. See also: http://www.emn.fr/x-info/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 2005, OBASCO members supervised two EMOOSE MSc theses by Luis Daniel Benavides Navarro [46] and Richa Gupta [49].

ALD Master The faculty members of the team participate to this master program and give lectures about new trends in the field of component-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. Gilles Muller was the co-chair together with José Martinez from Polytech Nantes until August 2005 when he chairs this formation.

In 2005, the OBASCO team welcomed student interns from the ALD master for their master thesis: Ali Assaf [45], Pierre Lavoix and Richa Gupta. Furthermore, Mario Südholt has supervised an MSc-level internship of Thomas Fritz, a student from *Ludwig-Maximilians Universität* of Munich [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 co-chairman of the *pôle informatique* associated to the CPER 2000-2006.

G. Muller: He is a member of the board of the “Ecole Doctorale STIM”.

J.-M. Menaud: He is treasurer of ASF, the French chapter of ACM Sigops.

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° 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° 10, ACM Press, ACM SIGPLAN, October 1998, p. 84–96.

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

[4] 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° 12, ACM Press, October 1987, p. 156–167.

[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. STEELE, JR. (editors). , vol. 38, n° 11, ACM Press, October 2003, p. 27–46.

Books and Monographs

[11] P. COINTE (editor). *Programmation par aspects*, vol. 11, n° 3, Hermès, 2005.

Doctoral dissertations and Habilitation theses

[12] P.-C. DAVID. *Développement de composants Fractal adaptatifs: un langage dédié à l'aspect d'adaptation*, Ph. D. Thesis, École des Mines de Nantes and Université de Nantes, July 2005.

[13] 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.

Articles in refereed journals and book chapters

[14] P. COINTE. *Les langages à objets*, chap. "Langages à objets" de l'Encyclopédie de l'informatique et des systèmes d'information, Vuibert, December 2005.

[15] S. DENIER. *Traits Programming with AspectJ*, in "RSTI - L'objet", vol. 11, n° 3, 2005, p. 69–86.

[16] 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", to appear, 2006.

[17] R. LEONARDO, É. TANTER, J. NOYÉ. *La réflexion comportementale partielle comme infrastructure de programmation par aspects - Étude du cas des coupes dynamiques*, in "RSTI L'Objet, Programmation par aspects", November 2005, p. 31-52.

[18] J. NOYÉ, R. DOUENCE, M. SÜDHOLT. *Composants et aspects*, in "Ingénierie des composants : Concepts, techniques et outils", M. OUSSALAH (editor). , chap. 6, Vuibert, 2005, p. 169-195.

[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, to appear, 2006.

Publications in Conferences and Workshops

[20] J.-P. BODEVEIX, M. FILALI, J. LAWALL, G. MULLER. *Applying the B formal method to the Bossa domain-specific language*, in "The 17th Nordic Workshop on Programming Theory (NWPT'05), Copenhagen, Denmark", October 2005.

[21] J.-P. BODEVEIX, M. FILALI, J. LAWALL, G. MULLER. *Formal Methods Meet Domain Specific Languages*, in "Fifth International Conference on Integrated Formal Methods, Eindhoven, The Netherlands", November 2005.

[22] 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, FCMO 2005, Leiden, The Netherlands", F. DE BOER (editor). , Lecture Notes in Computer Science, vol. 3657, Springer-Verlag,

November 2005, p. 70–94.

[23] 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.

[24] 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.

[25] P.-C. DAVID, T. LEDOUX. *Une approche par aspects pour le développement de composants Fractal adaptatifs*, in "2ème Journée Francophone sur le Développement de Logiciels Par Aspects (JFDLPA'05), Lille, France", N°3 RSTI L'objet, vol. 11, Hermès, September 2005.

[26] P.-C. DAVID, T. LEDOUX. *WildCAT: a generic framework for context-aware applications*, in "3rd International Workshops on Middleware for Pervasive and Ad-hoc Computing (MPAC)", ACM Digital Library, November 2005.

[27] S. DENIER, H. ALBIN-AMIOT, P. COINTE. *Expression and Composition of Design Patterns with Aspects*, in "2ème Journée Francophone sur le Développement de Logiciels Par Aspects (JFDLPA 2005), Lille, France", N°3 RSTI L'objet, vol. 11, Hermès, sept 2005.

[28] R. DOUENCE, T. FRITZ, N. LORIANT, J.-M. MENAUD, M. SÉGURA-DEVILLECHAISE, M. SÜDHOLT. *An expressive aspect language for system applications with Arachne*, in "Proc. of 4th International Conference on Aspect-Oriented Software Development (AOSD'05)", ACM Press, March 2005, p. 27–28.

[29] R. DOUENCE, J. NOYÉ. *Towards a Concurrent Model of Event-based Aspect-Oriented Programming*, in "European Interactive Workshop on Aspects in Software (EIWAS 2005), Brussels, Belgium", September 2005.

[30] T. FRITZ, M. SÉGURA-DEVILLECHAISE, M. SÜDHOLT, E. WUCHNER, J.-M. MENAUD. *An application of dynamic AOP to medical image generation*, in "International Workshop on Dynamic Aspects at AOSD (DAW'05)", March 2005.

[31] T. FRITZ, M. SÉGURA-DEVILLECHAISE, M. SÜDHOLT, E. WUCHNER, J.-M. MENAUD. *Automating adaptive image generation for medical devices using Aspect-Oriented Programming*, in "Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05)", September 2005.

[32] J. LAWALL, H. DUSCHENE, G. MULLER, A.-F. L. MEUR. *Bossa Nova: Introducing Modularity into the Bossa Domain-Specific Language*, 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, p. 78–93.

[33] J. LAWALL, G. MULLER, R. URUNUELA. *Tarantula: Killing Driver Bugs Before They Hatch*, in "The 4th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), Chicago, IL", March 2005, p. 13–18.

[34] N. LORIANT, M. SÉGURA-DEVILLECHAISE, J.-M. MENAUD. *Server protection through dynamic patching*, in "Proceedings of the 11th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'05), Changsha, Hunan, China", IEEE Computer Society, December 2005.

[35] N. LORIANT, M. SÉGURA-DEVILLECHAISE, J.-M. MENAUD. *Software security patches – Audit, deployment and hot update*, in "Proceedings of the Fourth AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS'O5), Chicago, USA", March 2005, p. 25–29.

[36] N. LORIANT, M. SÉGURA-DEVILLECHAISE, J.-M. MENAUD. *Un bac à sable juste à temps – correctifs ciblés et injection à chaud*, in "5ème Conférence Française sur les Systèmes d'Exploitation (CFSE'05), Le Croisic, France", March 2005.

[37] 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.

[38] F. MINJAT, A. BERGEL, P. COINTE, S. DUCASSE. *Mise en symbiose des traits et des classboxes, Application à l'expression des collaborations*, in "LMO 2005 - Langages et modèles à objets", March 2005, p. 33–46.

[39] G. MULLER, J. LAWALL, H. DUSCHENE. *A Framework for Simplifying the Development of Kernel Schedulers: Design and Performance Evaluation*, in "Tenth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2005), Sedona, AZ", February 2005, p. 219–230.

[40] 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.

[41] S. PAVEL, J. NOYÉ, P. POIZAT, J.-C. ROYER. *Java Implementation of a Component Model with Explicit Symbolic Protocols*, in "Proceedings of the 4th International Workshop on Software Composition (SC'05)", T. GSCHWIND, U. ASSMAN, O. NIERSTRASZ (editors). , Lecture Notes in Computer Science, vol. 3628, Springer-Verlag, April 2005, p. 115–124.

[42] M. SÜDHOLT. *A model of components with non-regular protocols*, in "Proceedings of the 4th International Workshop on Software Composition (SC'05)", T. GSCHWIND, U. ASSMAN, O. NIERSTRASZ (editors). , Lecture Notes in Computer Science, vol. 3628, Springer-Verlag, April 2005, p. 99–114.

[43] É. TANTER, J. NOYÉ. *A Versatile Kernel for Multi-Language AOP*, 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, vol. 3676, Springer-Verlag, September/October 2005, p. 173-188.

[44] R. URUNUELA. *Application Coopérative et Gestion de l'Energie du Processeur.*, in "1ères Rencontres des Jeunes Chercheurs en Informatique Temps Réel 2005 (RJCITR'05) Conjointement à l'école d'été temps réel 2005 (ETR'05)", September 2005.

Miscellaneous

[45] A. ASSAF. *A Wrapper API for Reflex*, MSc Thesis, École des Mines de Nantes/Université de Nantes, September 2005.

[46] L. D. BENAVIDES NAVARRO. *Dhamaca: an aspect-oriented language for explicit distributed programming*, MSc Thesis, Vrije Universiteit Brussel/École des Mines de Nantes, August 2005.

[47] S. BERKANI. *Conception d'un système vidéo personnel garantissant la qualité de service*, MSc Thesis, École des Mines de Nantes/Université de Nantes, September 2005.

[48] T. FRITZ. *An Expressive Aspect Language with Arachne*, MSc Thesis, Ludwig-Maximilians-Universität München, April 2005.

[49] R. GUPTA. *A Wrapper API for Reflex*, MSc Thesis, Vrije Universiteit Brussel/École des Mines de Nantes, August 2005.

[50] M. LÉGER. *Intégrité structurelle dans les architectures à composants. Application à l'intégrité des communications dans la plateforme Fractal/Julia*, MSc Thesis, École des Mines de Nantes/Université de Nantes, 2005.

[51] G. MULLER, J. LAWALL. *The Bossa Framework for Scheduler DevelopmentÉcole d'été Temps Réel 2005, Nancy, France*, September 2005.

[52] S. PAVEL, J. NOYÉ, J.-C. ROYER. *Un modèle de composant avec protocole symboliqueJournée du groupe Objets, Composants et Modèles, Bern, Suisse*, March 2005.

Bibliography in notes

[53] A. BERGEL, S. D. ANS OSCAR NIERSTRASZ. *Classbox/J: Controlling the Scope of Change in Java*, in "Proceedings of the 20th ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications (OOPSLA 2005), San Diego, California, USA", R. JONHSON, R. GABRIEL (editors). , ACM Press, October 2005, p. 177–190.

[54] A. P. BLACK, N. SCHÄRLI, S. DUCASSE. *Applying Traits to the Smalltalk Collection Classes*, 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. STEELE, JR. (editors). , ACM Press, October 2003, p. 47–64.

[55] 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.

[56] 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.

[57] 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° 4, 2004, http://www.lip6.fr/colloque-JFP.

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

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

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

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

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

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

[64] 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.

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

[66] 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.

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

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

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

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

[71] 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.

[72] 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.

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

[74] 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° 12, ACM Press, October 1987, p. 168–182.