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]

Staff Member INRIA

Jacques Noyé [CR1 INRIA on leave until August]

Faculty Members from EMN

Rémi Douence [Associate Professor]
Thomas Ledoux [Associate Professor]
Jean-Marc Menaud [Associate Professor]
Gilles Muller [Professor]
Jean-Claude Royer [Professor]
Mario Südholt [Associate Professor, on leave at INRIA since September]

Administrative Assistant

Sylvie Poizac [Part-time (50%)]

Ph. D. Students

Christoph Augier [Cifre grant with JANULA, since September]
Gustavo Bobeff [MINES contract]
Pierre-Charles David [MESR grant]
Simon Denier [EMN grant]
Hervé Duchesne [MINES grant]
Nicolas Loriant [EMN grant, since September]
Florian Minjat [MESR grant, since September]
Sebastian Pavel [MESR grant, ACI DISPO]
Marc Ségura-Devillechaise [MINES grant]
Éric Tanter [Grant from UNIVERSITY OF CHILE until November]
Luc Teboul [MINES grant until September]
Richard Urunuela [REGIONAL grant, since September]

Visiting Scientist

Julia Lawall [DIKU, University of Copenhagen, September to December]

Temporary Faculty Members

Hervé Albin-Amiot [On leave from SODIFRANCE since September]
Hervé Grall [MINES position since September]

2. 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 [13]. 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. Our objective is the implementation of a Component Virtual Machine (CVM) taking into account the three following levels :

1. A model formalizing the principles underlying the implementation of components, that is dealing with encapsulation, composition, interaction and adaptation.

2. A language integrating aspect-oriented and component-oriented programming in order to obtain reasonably expressive executable architectural descriptions.
3. A Java infrastructure including a number of tools, dealing with program analysis and transformation, interpretation, monitoring, and execution, common to the object, aspect and component approaches. These tools will be integrated in an Integrated Development Environment such as Eclipse.

In practice, the CVM federates our work along four directions:

2.1.1. 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.

2.1.2. 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.

2.1.3. 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.

2.1.4. 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 [13]. 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]. 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 [73]).

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* [69].

Aspects: *Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways. Examples of aspects include memory access patterns and synchronization of concurrent objects* (page 226 of [62]).

Reflection: *A process's integral ability to represent, operate on, otherwise deal with itself in the same way that it represents, operates and deals with its primary subject matter* [68].

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* [69]. The basic idea, as initially explained by M.D. McIlroy in 1968 [63] is to industrialize software reuse by setting up both an industry and a market of interchangeable parts. This corresponds to a strong decoupling between component producers and consumers, with new stages in the life cycle of a component (*e.g.*, packaging, deployment). This also leads to a kind of layered programming *in the small/in the large* with standard object-oriented languages used to implement *primitive* components, and a *component-oriented* language used to implement *compound* components, which can also be seen as *software architectures*. The two main features that a component-oriented language should support are:

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

These properties raise new challenges in programming language design and implementation. They require an integration of ideas coming from module interconnection languages [66], architecture description languages (ADLs) [67][64] 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

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

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* [69] prototypes have been developed as Java extensions. These languages focus on component structure.

At the implementation level, an important issue is the exacerbated conflict between *early* and *late binding* due to, on the one hand, strong encapsulation and the need to address errors as early as possible in the life cycle, and, on the other hand, the possibility to adapt a component all along its life cycle. Software specialization (*e.g.*, partial evaluation [60]) 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 [72]. 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:

- *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:

3.2.2.1. 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 [57][61] to extend the base language with new mechanisms such as multiple inheritance, concurrency or metaclass composition [2].
- In the case of open middleware [7], the main usage of behavioral reflection is to control message sending by interposing a metaobject in charge of adding extra behaviors/services (such as transaction, caching, distribution) to its base object. Nevertheless, the introduction of such *interceptor/wrapper* metaobjects requires to instrument the base level with some *hooks* in charge of causally connecting the base object with its metaobject [9].

3.2.2.2. Model-View-Controller:

The *MVC* developed for Smalltalk [59] 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]. 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. 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 obviously more restricted than a general-purpose language, such as Java or even C, but encapsulates domain expertise making it easier to verify important safety properties. DSLs are interesting because they can be used as a model to describe specific and crosscutting aspects of a system.

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] [71].

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 investigating, in particular, the application of separation of concerns - 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 [56][65].

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).

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 EJB and Fractal components (see Section 6.1).

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 [29][31][33][32]. 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 [33].
- 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. More precisely, we have proposed to guide the event insertion by using a set of rules, amounting to an aspect, that describes the control-flow contexts in which each event should be generated. Only 38 rules are needed for the Linux 2.4 kernel so as to support Bossa [36].
- 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.

A prototype of Bossa is publicly available at http://www.emn.fr/x-info/bossa. 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 in teaching scheduling.

Bossa was initially developed in the context of a research contract between France Télécom R&D and the Compose INRIA project. It is currently supported by Microsoft Research (see Section 7.2) and developed jointly by EMN and the University of Copenhagen (DIKU). A proposal for a CIFRE grant with Jaluna (formerly Chorus Systems) is under evaluation by the ANRT to support the Ph.D. thesis of Christophe Augier. Its goal 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. Finally, we are extending Bossa to support energy management through grant of the Pays de Loire council for the Ph.D. of Richard Urunuela.

5.2. EAOP

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

Participants: Mario Südholt [correspondent], Rémi Douence, Luc Teboul.

The EAOP software provides a Java testbed for the definition of expressive aspect languages. It enables weaving of aspect behavior during program execution based on relations between execution events. It supports the definition of expressive aspect languages and a general notion of dynamic aspect composition.

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 (implemented by B. Lewis). 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, Jean-Marc Menaud [correspondent], Gilles Muller, Julia Lawall, Mario Südholt, Rémi Douence.

Arachne is 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 Aspect-Oriented Programming 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 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 [70], 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.

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

5.4. Reflex

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

Participants: Eric Tanter, Jacques Noyé [correspondent], Pierre Cointe, Simon Denier.

Reflex is an open behavioral reflective extension of Java. Compared to the Java reflection library, which only offers a collection of low-level primitives for building reflective systems, Reflex provides naive users of reflection with a ready-to-use and expressive metaobject protocol (MOP) and experienced users with an architecture and building blocks for implementing and using application-specific MOPs. Reflex is currently being evolved into a versatile AOP kernel for Java.

Behavioral reflection makes it possible to control at the metalevel specific base-level operations (*e.g.*, message send, instantiation, cast...). A great strength of behavioral reflection is to provide the means to achieve a clean separation of concerns, including dynamic concerns, and hence to offer a modular support for adaptation in software systems. This strength has already been exercised in a wide range of domains including distribution, mobile objects, and fault-tolerance. The applicability of a naive implementation of behavioral reflection is, however, limited by a number of issues:

- The definition of a unique, hardwired, metaobject protocol (MOP) unable to take into account the specific requirements of various target applications and set the appropriate trade-off between performance, expressiveness, and flexibility.
- A simplistic view of the link between the base level and the metalevel making it difficult to appropriately structure real-life applications.
- The cost of reifying base-level operations.

Reflex provides a powerful Java infrastructure on which to implement reflective applications by solving the above-mentioned issues as follows:

- Reflex is an *open* behavioral reflective extension of Java. As opposed to other reflective extensions, it does not impose any specific MOP, thanks to a layered architecture. Indeed, Reflex allows *metalevel architects* to define their own MOP, based on the framework provided by *Core Reflex*, possibly reusing parts of a *standard MOP* library. This library, built on top of *Core Reflex*, provides a standard MOP and turns Java into a ready-to-use behavioral reflective system. All this is is done without compromising portability: Reflex relies on load-time bytecode transformation, using Javassist, and can therefore be delivered as a portable Java library.
- Core Reflex provides the means to independently define *hooksets*, *i.e.*, sets of execution points to be reified, *metaobjects*, and *links*, associating hooksets and metaobjects. All these elements can be defined and configured either statically, using configuration classes or XML configuration files, or dynamically. This makes it possible to relate several execution points and handle crosscutting concerns in a very flexible way.
- Configuration also includes the possibility to define where and when reflection is useful, through *spatial* and *temporal selection*. *Spatial selection* deals with selecting the operation of interest, as well as their occurrences of interest, based on their location in the code. *Temporal selection* refers to the possibility of a lazy creation of reflective objects, and to the notion of *link activation*, which makes it possible to connect a hookset to the metalevel only when needed. This implements *partial* reflection.

A prototype of Reflex is available at http://www.dcc.uchile.cl/~etanter/Reflex/index.html. This prototype is currently being evolved into a versatile AOP kernel for Java (see Section 6.2). 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.

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, Thomas Ledoux, Jacques Noyé, Sebastian Pavel, Jean-Claude Royer, Mario Südholt.

At the theoretical level, we study two component model extensions: explicit interaction protocols, and asynchronous and synchronous communications with 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.

6.1.1. Explicit Protocols

We argue that explicit protocols are of great interest in architectural description languages but in component programming languages too [39]. In our current approach the dynamic behavior of a component is represented

as a symbolic transition system (STS). We have defined a general technique to formally describe the data part and the dynamic behavior of such a concurrent and communicating architecture. One resulting issue was to be able to write and check temporal properties for such systems. We defined such an approach based on temporal algebraic operators expressing a temporal partial ordering on defined execution paths. Temporal properties, such as deadlock freeness, are written as first-order formulas and may be checked in the algebraic context. This approach allows us to write safety, liveness or fairness properties. We also define CTL*Data, a natural extension of CTL* with first-order data formulas, which matches our temporal logic [42].

On a more practical side we are investigating the implementation of some of these ideas over Java. A first approach was done during the Master thesis of A. Sedkaoui [53]. S. Pavel continues these experiments and proposes a Java library to support the use of STS and asynchronous communications.

Finally, we have investigated how the framework for interaction analysis among aspects we have previously developed [5] can be applied to software components. To this end, we have integrated it with the component model with explicit finite-state protocols defined as part of Andrés Farías's PhD thesis. This on-going work has resulted in a set of formal conditions ensuring semantic compatibility of the two approaches and an evaluation of the preservation of composition properties of such software components in the context of protocol-modifying aspects [45].

6.1.2. Property Checking

We are also interested in ways to check properties about components and architectures. We compute the global dynamic behavior of an architecture using the principle of the synchronous product. Property checking is possible with proof assistants based on first-order logic.

To fit distributed systems, it seems relevant to consider asynchronous communications. It provides a more primitive communication protocol and maximize the concurrency. To take into account asynchronous communications we distinguish message receipt from message execution and we add mailboxes in our symbolic systems.

Asynchronous communications lead to more complex dynamic behavior, it increases the need for verification tools. Especially one point is to analyse the boundedness of set of communicating components. We advocate for the use of Symbolic Transition Systems as a protocol language which may deal also with this kind of communication. We then study how this generic formalism, specialized with different mailbox protocols, may be used to address verification issues related to the component mailboxes. The general problem with FIFO mailboxes is not decidable, but it is with dictionary mailboxes. Furthermore the boundedness of dictionary mailboxes is a necessary condition for FIFO boundedness [51][35].

When we tried to experiment proofs in systems with asynchronous communication a difficulty was the presence of buffers and the fact that the receipt instant is distinct from the execution instant. This complicates the specifications and also the proofs. The main problem is that the logic instant to receive a message is not simply linked with the execution instant. We propose to use an algorithm which decides if the system has bounded mailboxes and computes the reachable mailbox contents of the system. This algorithm gives constraints which are used to specialise the dynamic behaviour of the components according to the current system configuration. Then we are able to generate a PVS specification coping with dynamic behaviour and data type which is simpler since it removes the need for some mailboxes [37].

Architecture diagrams represents components with their communication links, UML may be used to describe such architectures. We introduce a structural and object-oriented formal model and applications of this model to the checking of some ill-formed diagrams. We focus on static diagrams mixing inheritance and composition relations. We consider an approach based on the usefulness notion for a class or a component type. It means that a type description is finitely generated and with at least one defined value. We defined a general process to check the notion of usefulness and a static algorithm which applies to the UML language [16].

6.1.3. Adaptation

Component-Based Software Development is an attractive way to deliver generic executable pieces of program, ready to be reused in many different contexts. Component reuse is based on a black-box model that frees component consumers from diving into implementation details. Adapting a generic component to a particular context of use is then based on a parameterized interface that becomes a specific component wrapper at runtime. This *shallow* adaptation, which keeps the component implementation unchanged, is a major source of inefficiency. By building on top of well-known specialization techniques, it is possible to take advantage of the genericity of components and adapt their implementation to their usage context without breaking the black-box model. In [17], we illustrate these ideas on a simple Java-compatible component model, considering dual specialization techniques, partial evaluation and slicing. A key to not breaking encapsulation is to use *specialization scenarios* extended with *assumptions* on the required services and to package components as *component generators*. Based on the idea presented in the paper mentioned above, we are currently working on a prototype development tool based on Eclipse and REQS, the recursive equation solver developed at Irisa by the Lande project.

This first kind of adaptation is automatic and takes place at assembly time. We are also interested in a second, very different, kind of adaptation that is programmed and takes place at runtime.

In [23], we propose to build self-adaptive component-based applications to fit their evolving environment. We present the general approach following the separation of concerns principle where the adaptation logic of an application is developed separately from the rest of it. The proposed framework is mainly based on a context-awareness service and an adaptation policies development framework. The context-awareness service provides information about the execution context (network connection, available memory...). This information is used by adaptation policies which capture the adaptation concern and allow the dynamic reconfiguration of component-based applications. These policies are Event/Condition/Action rules expressing: the event of interest, the needed condition to satisfy before realizing the associated action (i.e., a reconfiguration). The adaptation mechanism is based on a weaving process that dynamically binds the adaptation policies to the components, making them self-adaptive. This framework has been used to adapt Java objects (RAM), Entreprise Java Beans (Jonas) and more recently Fractal components. The next step, presented in the body of P.-C. David's Ph.D. thesis, will be the expression of adaptation policies with a DSL.

6.1.4. Dynamic Configuration

We consider the use of a state-of-the-art, general-purpose, component programming language, specifically ArchJava, to implement software product lines. Component programming languages provide a more straightforward mapping between components as assets and components as implementation artifacts. However, guaranteeing that the implementation conforms to the architecture raise new issues with respect to dynamic configuration. We show how this can be solved in ArchJava by making the components auto-configurable, which corresponds to replacing components by component generators. Such a scheme can be implemented in various ways, in particular with a two-stage generator. This solution goes beyond the initial technical ArchJava issue and complements the standard static generative approach to software product-line implementation [38].

6.2. Aspects

Keywords: *Arachne, Bossa, EAOP, Reflex, expressive aspect languages, separation of concerns.*

Participants: Mario Südholt, Rémi Douence, Pierre Cointe, Jean-Marc Menaud, Gilles Muller, Julia Lawall, Marc Ségura-Devillechaise, Luc Teboul, Éric Tanter, Simon Denier.

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

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

Advices/actions: A specification of what an aspect computes. In AspectJ, these are method-like constructs used to define additional behavior at join points.

Aspect: a 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 advice, and ordinary Java member declarations.

Weaver: a 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.

OBASCO pursued the definition and application of expressive aspect languages, investigated the relationship between AOP and reflection, and applied expressive aspects to problems arising in the context of system-level applications.

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. While separation of concerns was identified a long time ago as a basic problem in software engineering, language-oriented solutions to crosscutting concerns are much more recent. Today, most general definitions of AOP remain vague and broad. In different position papers, we have reviewed several such approaches and provided means for a better understanding of AOP [26][22][21][20].

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 [15]. This year we have extended a framework for the static analysis of aspect interactions, considered its adaptation to software components as well as an extension accounting for control-flow relationships.

Based on our previous work [5], we have extended the language of crosscut definitions while keeping static analysis capabilities. We still focus on sequences (of execution events) defined as regular expressions, but we allow equality constraints between several events to be expressed using variables. We have also developed more expressive support for conflict resolution and proposed a notion of enriched interfaces that make explicit the family of base programs which can safely be woven with an aspect. This extension provides support for reuse of aspects [27].

The EAOP model has also been used to extend the Ph.D. thesis of Andrés Farías in order to integrate aspects with software components defined using explicit protocols [45] (see Section 6.1 for details).

Finally, we have defined general means for the definition of pointcuts in terms of control flow relationships. AspectJ provides an operator `cflow` to define pointcuts in the control flow of the base application and to pass contextual information between two points in a control flow. This mechanism has been found useful for several applications, in particular, optimization of the refresh in graphical applications and file prefetching in operating systems. We have proposed a more general pointcut language for control flow [28]. It enables context to be passed downward in the control flow, but also upward with a new operator `path`, and from left to right with another new operator `from`. We have provided a formal definition of this language and implemented it on top of AspectJ. A static alphabet analysis has been defined in order to optimize its implementation.

6.2.2. A reflective versatile kernel for AOP

AOP is a promising approach to modularizing software in presence of crosscutting concerns. Numerous proposals for AOP have been formulated, some of them generic, others specific to particular concerns. There are commonalities and variabilities among these approaches, which are worth exploring. Unfortunately, in practice, these various approaches are hard to combine and to extend. This results from the fact that the corresponding tools, such as aspect weavers, have not been designed to be used along with others, although they usually perform very similar low-level tasks.

In [47][43], we suggest to include common functionality into a *versatile kernel for AOP*. Such a kernel alleviates the task of implementing an aspect-oriented approach by taking care of basic program alterations. It also lets several approaches coexist without breaking each other by automatically detecting interactions among aspects and offering expressive composition means. From a review of the main features of Aspect-Oriented Programming, we present the main issues that the design of such an AOP kernel should address: open support for aspect languages taking care of both behavior and structure, base language compliance, and aspect composition.

As part of this infrastructure was already provided by Reflex [47], we have studied how to evolve Reflex into such a kernel. [40] reports on an experiment to support a widely-accepted, general-purpose aspect language mechanism, namely AspectJ's dynamic crosscutting, with our model of partial behavioral reflection. We present a first approach to such a mapping, identify some extensions that can improve the effectiveness of the mapping, and validate our proposal through a revisited mapping. These extensions have been integrated into Reflex. Benchmarks have shown that the resulting performance of AspectJ's dynamic crosscutting on Reflex is promising [11]. Moreover, [11] addresses the integration within Reflex of a number of missing features : support for structural aspects, aspect composition, and the definition of (possibly domain-specific) aspect languages. Both structural cut and actions (including intercession) have been integrated in our model. Aspect composition distinguishes between inter-aspect collaboration and detection/resolution of aspect interactions. Inter-aspect collaboration allows an aspect to expose or hide its changes to other aspects. Detection/resolution of aspect interactions detects when two aspects directly affects a program point and use composition rules to deal with the interaction. Finally, a lightweight plugin architecture provides support for defining aspect languages. A case study based on the Sequential Object Monitors described in [18] shows the benefits of the approach.

6.2.3. Aspects for system-level applications

6.2.3.1. Aspects to support run time evolution of system-level applications.

In the context of Web caches (see Section 5.3), we have previously developed another strategy for dynamically changing cache prefetch strategies without interrupting request servicing [70]. This allows the evolution of web services performed at the level of binary code at run time. We used the same approach to deploy critical security updates on the fly on various proxies like web caches, ftp servers, ssh daemons, etc. This year, we have shown that security updates are crosscutting concerns and can be considered as aspects [34]. We are investigating how to unify both approaches.

Furthermore, we have started work on the application of EAOP-based techniques to system-level applications written in C. Concretely, we have defined two constructs for sequence-based pointcuts geared towards the declarative formulation of sequences of statements and dependencies between variable and argument values occurring in such sequences. Currently, we are working on the integration of these advanced pointcut language in the tool Arachne (see Section 5.3) and its application in the context of a medical imaging application from Siemens AG, Germany [49].

6.2.3.2. Definition of OS-level component interfaces using aspects.

Furthermore, we have shown how to use aspect-oriented techniques for the extraction of component interfaces of OS-level legacy code [36] in the context of the systems Bossa and Arachne (see Sections 5.1, 5.3).

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

In [23], we consider adaptation to a specific execution context and its evolutions as an aspect which should be treated separately from the rest of an application (see Section 6.1 for a complete description).

6.3. Post-Objects

Keywords: *AspectJ*, *Eclipse*, *Java*, *Reflex*, *Squeak*, *classboxes*, *inter-type declaration*, *java.lang.reflection*, *opening*, *specialization*, *static AOP*, *traits*.

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

to reify: prviding 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 being effectively active).

hook: the base level piece of code responsible for performing a reification and giving control to the metaobject. To be compared with an AOP jointpoint.

hookset: to gather execution scattered in various objects.

We are investigating some new directions in the field of Object-Oriented languages. Our general purpose is to open these languages in order to introduce extra mechanisms such as revisited encapsulation and inheritance, reflection, crosscutting aspects, objects interaction and composition. At that time we are prototyping with Java via the Reflex infrastructure.

6.3.1. Aspect Reifications

The two PhD theses of H. Albin and Y-G. Guéhéneuc were dedicated to the reification and analysis of design patterns. There goals were to fill in the gap between programming languages *à la* Java, and modeling languages à la UML by providing on the one hand automatic code generation from a given model and on the other hand detection of complete and distorted forms of design patterns in Java sources. One of the main contribution was consensual definitions of the binary class relationships (association, aggregation, and composition) with four properties and the definition of algorithms to automatically dectect them in source code [30].

Our current work on aspectualisation of legacy Java code done in cooperation with Sodifrance will reuse these previous results (see also Section 7.1).

6.3.2. Traits in Java

TRAITS as defined in [55] provide an interesting level of granularity intermediate between a method and a class. More precisely a trait is a set of requested and provided methods that can be used by a Smalltalk class as a mixin module. We have demonstrated in [24][25][21] that such a trait can be emulated in Java by using the AspectJ introduction mechanism, then establishing a relationship between structural AOP (static weaving) and traits composition.

6.3.3. Reflection, Aspects and Components in Smalltalk

In the tradition of ObjVlisp and ClassTalk [4][2], we have also investigated the use of reflection to support AOP in Smalltalk. In [12], T. Ledoux and N. Bouraqadi introduce the MetaclassTalk MOP and discuss one possible approach to isolate aspects as explicit metaclasses and then weave them together by using metaclass composition.

Taking advantage of the great malleability of this dynamically typed language, we have started the investigation of TRAITS as defined in [55] to provide a clean Smalltak/Squeak MOP, which would then be used in turn to integrate traits, classboxes (class extensions), and aspects in a uniform way [52].

6.3.4. Reflex: Partial Behavioral Reflection in Java

Reflection is a powerful approach for adapting the behavior of running applications and to provide the means to achieve a clean separation of concern. Nevertheless the applicability of (behavioral) reflection is limited by the lack of a widely accepted appropriate infrastructure on which to implement reflective applications. Our main results presented in [10][11] are as follows:

- a comprehensive approach to reflection based on the model of hooksets. The idea consists of grouping reified execution points into composable sets, possibly crosscutting object decomposition and attaching at the metalevel some behavior to these sets through an explicit configurable link,
- in the context of Java, the implementation of the Reflex open architecture supporting this approach and making it possible to combine static and dynamic configuration of reflection (see also Section 5.4).

6.4. AOP 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 two years. It is fully compatible with the Linux 2.4 kernel and can be used as a direct replacement. Most of our work has been done to add functionalities such as the support of a hierarchy of schedulers which allows an application to customize the global behavior of the OS to specific needs [32][29][33]. As an example we have designed policies for multimedia applications and teaching labs.

Our first Bossa kernels were manually re-engineered. This process is tedious, error prone and requires a lot of work (the Linux kernel currently amounts to over 100MB of source code). We have experimented an AOP-based approach to transform the Linux kernel (see [36] and Section 5.1).

Additionally, we have investigated the extension of Bossa to energy management [54] and the support of the network [48]. Finally, we are investigating in designing a modular version of the Bossa DSL so as to express in a simpler way families of real-time scheduling policies [31].

6.4.2. Extensible caches and security updates

Our work on Arachne has focused on the design and implementation of a robust prototype that can be made publicly available [70]. 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].

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 [30], 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 Eclipse/AspectJ.

7.2. Microsoft Research

Participants: Gilles Muller, Jean-Marc Menaud, Mario Südholt, Hervé Duchesne.

Our work on the development of Bossa (see Section 5.1) is currently supported by Microsoft Research via two grants. The first grant of 30 000 euros is related to our AOP-based approach for re-engineering existing kernel [36] and the integration of Bossa within the .NET framework.

The second grant of 25 000 US $ is an award resulting from a world-wide Embedded Systems Request for Proposal (RFP) that MSR has run. The topic of this grant is a port of Bossa to Windows XP embedded.

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 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 a Pays de Loire project started in September 2004 for 30 months, with 78 Keuros of fundings. Its objective is to explore and design new development tools for programming computational grids. These grids promise to be the next generation of high-performance computing resources and programming such computing infrastructures will be extremely challenging.

This project addresses the design of an aspect-based software infrastructure for computational grids based on our EAOP model and our current prototype of Arachne (See section 5.3)). 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 will be also the first opportunity to collaborate with the PARIS team on the application of AOP to the domain of Grid computing.

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 joint project is funded by ANVAR via ARMINES for an amount of 19 Keuros (2003/2004). 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. RNTL ARCAD

Participants: Thomas Ledoux, Pierre-Charles David, Pierre Cointe, Jacques Noyé, Jean-Marc Menaud, Eric Tanter.

The ARCAD project was an RNTL project running from December 2000 to June 2004 with a 107 Keuros funding for our team. Its goal was to propose a component-based software architecture enabling the building of distributed adaptable applications. Experiments were realized in the ObjectWeb context with JOnAS and Fractal (see http://www.objectweb.org).

The project has federated work between five partners: France Télécom R&D, INRIA Rhône-Alpes (Sardes project), INRIA Sophia-Antipolis (Oasis project), laboratoire I3S (CNRS et Université de Nice, Rainbow team) and ourself.

In 2004, our team was in charge of two deliverables. The first one deals with the optimisation issue in adaptive component-based architecture [46]. This deliverable emphasizes the cost introduced by the adaptation and proposes several technics to reduce this cost (and then to increase efficiency such architectures). The second one presents the ARCAD final architecture [44] based on a Fractal/Julia kernel.

8.2.3. Action incitative CORSS

Participant: Gilles Muller.

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 generalizing the Bossa framework to multiple types of OS resources such as energy, disks and networks.

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

8.2.4. 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*) Bretagne.

8.3. European Projects

8.3.1. NoE AOSD

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

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-Claude Royer, 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 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 first two days workshop has been held in Santiago on november "around" Eric Tanter's Ph.D. defence [11].

9. Dissemination

9.1. Animation of the community

9.1.1. Animation

9.1.1.1. ACM/Sigops:

G. Muller is the vice-chair of the ACM/Sigops, and the chair of the French Sigops Chapter (ASF).

9.1.1.2. CNRS/RTP Distributed System:

G. Muller is a member of the board of the CNRS *Réseau Thématique Prioritaire* on Distributed Systems.

9.1.1.3. CNRS/GDR ALP:

The team is a member of the CNRS *Groupe de Recherche* "Algorithms, Languages and Programming" (see http://www.liafa.jussieu.fr/~alp). As such, we participated to the special day organized in conjunction with the LMO conference in Lille.

9.1.1.4. EJCP 2004:

The summer school *École des jeunes chercheurs en programmation du GDR ALP* has been co-organized by Rémi Douence and Mario Südholt and partially hosted at École des Mines de Nantes. EJCP'04 has provided advanced courses on computer languages and programming to around 40 PhD students during two weeks (May/June).

9.1.1.5. JFDLPA 2004:

Under the auspice of the NoE AOSD we have organized this first french workshop related to aspect-oriented design and programming. This forum organized at IRCAM in october has welcomed around 70 participants [19].

9.1.1.6. JMAC 2004:

This one day conference is dedicated to multi-agent and component systems. It was organized in link with the ANVAR project and was hosted at Ecole des Mines de Paris. See http://csl.ensm-douai.fr/MAAC/3 for more details.

9.1.1.7. 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 will be organized by Gilles Muller (General chair), Jean-Marc Menaud and Thomas Ledoux in Le Croisic.

9.1.1.8. 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

9.1.2.1. P. Cointe:

He is a member of the ECOOP and LMO steering committees (http://www.ecoop.org). He has reviewed papers for TOPLAS and TSI.

He is a program committee member of the Software Composition workshops organized in the context of ETAPS 2004 in Barcelone and 2005 in Edinburgh . He is a program committee member of the OOPS special technical track at the 19th and 20 th ACM Symposium on Applied Computing (see http://oops.disi.unige.it/OOPS).

He has served as a program committee member of the eTX (eclipse Technology Exchange) workshop at ETAPS (Barcelona, March/April 2004, see http://www.lsi.upc.es/etaps04).

He has been a track leader, on Generative Programming, at the Unconventional Programming Paradigms workshop (Saint-Malo, September 2004, see http://upp.lami.univ-evry.fr/ and [22]).

He has been invited speaker at the third FMOC (*symposium on Formal Methods for Components and Objects*) in Leiden [21], and the 7th CARI (*African Conference On Research In Computer Science*) in Hammamet [20].

9.1.2.2. T. Ledoux:

He is the program chairman of JC 2005 (*Journées Composants*, Le Croisic, April 2005).

9.1.2.3. G. Muller:

He was a program committee member of DSN 2004 (IEEE Conference on Dependable Systems and Networks, Firenze, June 2004), FTDCS 2004 (International Workshop on Future Trends of Distributed Computing Systems, Suzhou, China, May 2004), EW04 (ACM SIGOPS European Workshop, Leuven, September 2004), PLOS04 (ECOOP Workshop on Programming Languages and Operating Systems, June 2004, Oslo, Norway), DECOR04 (1ère Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels, October 2004, Grenoble)

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 is the program chair of RENPAR 2005 (16ème édition des Rencontres Francophones du Parallélisme, April 2005, Le Croisic).

9.1.2.4. J. Noyé:

He was a program committee member of LMO 2004 (*Langages et Modèles à Objets*, Lille, March 2004) and JMAC 2004 (*Journée Multi-Agent et Composant*, Paris November 2004). He is a program committee member of JC 2005 (*Journées Composants*, Le Croisic, April 2005).

9.1.2.5. 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 program chair of JMAC 2004 (*Journée Multi-Agent et Composant*, Paris November 2004), and a program committee member OCM-SI (*Objets, Composants et Modèles dans les Systèmes d'Information*, Biarritz May 2004), IASSE 2004 (*Intelligent and Adaptative Systems and Software*

Engineering, Nice July 2004), LMO 2005 (*Langages et Modèles à Objets*, Berne, March 2005) and JC 2005 (*Journées Composants*, Le Croisic, April 2005).

9.1.2.6. M. Südholt:

He was a program committee member of AOSD 2004 (Aspect Oriented Software Design, Boston, March 2003) and will serve on those for AOSD 2005 and 2006. He served as a program committee member of the workshops ACP4IS and FOAL at AOSD 2004, and SC at ETAPS 2004. He will serve at the programm committee of the workshop DAW at AOSD 2005.

9.1.3. Thesis committees

9.1.3.1. P. Cointe:

He reviewed the HDR of P. Lahire (U. de Nice - Sophia Antipolis, 10/12/04), and the PhD thesis of W. De Meuter (Vrije Universiteit Brussels, 10/9/04). He participated to the HDR committee of C. Agon (Ircam, 7/5/04).

9.1.3.2. G. Muller:

He reviewed the Ph.D. of Eric Marsden (U. de Toulouse/Laas, 26/2/2004), Damien Deville (U. of Lille), Benoît Poulot-Cazajous (U. of Paris 7/Jaluna).

9.1.3.3. J. Noyé:

He reviewed the Ph.D. of O. Nano (U. de Nice - Sophia Antipolis, 6/12/2004).

He has been the scientific advisor of E. Tanter (Université de Nantes and University of Chile, 8/11/2004) [11].

9.1.3.4. J-C. Royer:

He reviewed the Ph.D. of Antoine Rollet (U. de Reims - Champagne Ardenne/CRESTIC, December 2004) and F Seyler (U. de Pau/LIUPPA, December 2004).

9.1.3.5. M. Südholt:

He served on the PhD committee of W, Vanderperren at Vrije Universiteit Brussels.

9.1.4. Evaluation committees and expertise

9.1.4.1. P. Cointe

As a member of the MSTP (*Mission Scientifique Technique Pédagogique*), he was in charge of evaluating proposals for the ACI *Young Researchers*. As a member of the France-Maroc scientific committe he participated to the STIC workshop in Rabat (december) and the evaluation of PhD proposals.

9.2. Teaching

9.2.1. 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 sixth promotion graduated in September 2004 while the seventh promotion was about to start their first semester. See also: http://kgb.emn.fr:8099/.

OBASCO team (along with its patners 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.

9.2.2. DEA infomatique de Nantes and other MSC-level internships.

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 is the co-chair together with José Martinez from Polytech Nantes.

In 2004, the OBASCO team welcomed five student interns from the DEA for their master thesis: Christophe Augier, Nicolas Loriant, Olivier Maréchal, Florian Minjat, and Richard Urunuela [48][50][51][52][54]. Furthermore, Mario Südholt supervised a MSc-level internship of Thomas Fritz [49], a student from *Ludwig-Maximilians Universität* of Munich.

9.3. Collective Duties

9.3.1. P. Cointe:

He is chairman of the Computer Science Department at EMN, the co-chairman of the new CNRS FRE LINA (*Laboratoire Informatique de Nantes Atlantique*) and the co-chairman of the *pôle informatique* associated to the CPER 2000-2006. He has been a member of the MSTP (*Mission Scientifique Technique Pédagogique*) since March 2003. Finally, he is a member of the France-Maroc scientific committee in charge of the STIC cooperation.

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.

Doctoral dissertations and Habilitation theses

[11] É. TANTER. *From Metaobject Protocols to Versatile Kernels for Aspect-Oriented Programming*, Ph. D. Thesis, École des Mines de Nantes, Université de Nantes, and University of Chile, November 2004.

Articles in referred journals and book chapters

[12] N. M. BOURAQADI-SAÂDANI, T. LEDOUX. *Supporting AOP Using Reflection*, in "Aspect-Oriented Software Development", R. E. FILMAN, T. ELRAD, S. CLARKE, M. AKSIT (editors)., Addison-Wesley Professional, 2004, p. 261-281.

[13] 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.

[14] R. DOUENCE, P. FRADET. *The next 700 Krivine machines*, in "Higher-Order and Symbolic Computation", 2004.

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

[16] J.-C. ROYER. *Checking Class Schema Usefulness*, in "Journal of Object Technology (JOT)", vol. 3, n° 1, January 2004, p. 157-176.

Publications in Conferences and Workshops

[17] G. BOBEFF, J. NOYÉ. *Component Specialization*, in "2004 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'04), Verona, Italy", ACM Press, August 2004.

[18] D. CAROMEL, L. MATEU, É. TANTER. *Sequential Object Monitors*, in "ECOOP 2004 - Object-Oriented Programming,18th European Conference, Oslo, Norway", M. ODERSKY (editor)., Lecture Notes in Computer Science, n° 3086, Springer-Verlag, June 2004, p. 316-340.

[19] P. COINTE. *Actes de la Premi‘ere Journée Francophone sur le Développement du Logiciel par Aspects (JFDLPA'04)*, P. COINTE, M. SÜDHOLT (editors)., October 2004, http://www.emn.fr/x-info/obasco/events/jfdlpa04/actes/actes-jfdlpa04.pdf.

[20] P. COINTE. *Comprendre la programmation par aspects*, in "Proceedings of the 7th African Conference in Computer Science, Hammamet, Tunisie", CARI, November 2004.

[21] P. COINTE. *From objects to aspects*, in "Proceedings of FMC0'2004, Leiden, The Netherlands", Invited talk, Springer-Verlag, Third International Symposium on Formal Methods for Components and Objects, November 2004.

[22] P. COINTE. *Understanding Generative Programming: from (meta)objects to aspects*, in "Pre-Proceedings Unconventional Programming Paradigms (UPP'04), Mont St-Michel, France", To be published in the hot topics subline of LNCS, Springer-Verlag, ERCIM/NSF, September 2004, p. 219-222.

[23] P.-C. DAVID, T. LEDOUX. *Pour un aspect d'adaptation dans le développement d'applications à base de composants*, in "Actes Journée de l'AS 150, Systèmes répartis et réseaux adaptatifs au contexte, Paris, France", April 2004.

[24] S. DENIER, P. COINTE. *A propos du modèle des traits et de sa transposition en Java à l'aide d'AspectJ*, in "Journée du groupe Objets, Composants et Modèles, Lille, France", March 2004, p. 13-18.

[25] S. DENIER. *Traits Programming with AspectJ*, in "Première Journée Francophone sur le Développement du Logiciel par Aspects", P. COINTE, M. SÜDHOLT (editors)., October 2004, p. 62-78, http://www.emn.fr/x-info/obasco/events/jfdlpa04/actes/actes-jfdlpa04.pdf.

[26] R. DOUENCE. *A restricted definition of AOP*, in "European Interactive Workshop on Aspects in Software (EIWAS'04), Berlin, Germany", September 2004.

[27] R. DOUENCE, P. FRADET, M. SÜDHOLT. *Composition, Reuse and Interaction Analysis of Stateful Aspects*, in "Proceedings of the 3rd Int. Conf. on Aspect-Oriented Software Development (AOSD'04)", ACM Press, ACM, March 2004.

[28] R. DOUENCE, L. TEBOUL. *A crosscut language for control-flow*, in "GPCE 2004 - 3rd ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering, Vancouver, Canada", Lecture Notes in Computer Science, vol. 3286, Springer-Verlag, October 2004, p. 95-114.

[29] H. DUCHESNE, C. AUGIER, R. URUNUELA. *Déploiement d'ordonnanceurs de processus spécifiques dans un système d'exploitation généraliste*, in "Conférence Francophone sur le Déploiement et la (Re) Configuration de Logiciels (DECOR'04), Grenoble, France", Octobre 2004, p. 193–198.

[30] Y.-G. GUÉHÉNEUC, H. ALBIN-AMIOT. *Recovering Binary Class Relationships : Putting Icing on the UML Cake*, in "Proceedings of OOPSLA'2004, Vancouver, Canada", ACM SIGPLAN, October 2004.

[31] J. LAWALL, A.-F. LE MEUR, G. MULLER. *Modularity for the Bossa Process-scheduling Language*, in "Proceedings of the ECOOP Workshop on Programming Languages and Operating Systems (PLOS'04), Oslo, Norway", June 2004.

[32] J. LAWALL, G. MULLER, H. DUCHESNE. *Language Design for Implementing Process Scheduling Hierarchies*, in "2004 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'04), Verona, Italy", Invited Paper, ACM Press, August 2004.

[33] J. LAWALL, G. MULLER, A.-F. LE MEUR. *On the design of a domain-specific language for OS process-scheduling extensions*, in "GPCE 2004 - 3rd ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering, Vancouver, Canada", Lecture Notes in Computer Science, Springer-Verlag, October 2004.

[34] N. LORIANT, M. SEGURA-DEVILLECHAISE, J.-M. MENAUD. *Des correctifs de sécurité à la mise à jour - Audit, déploiement distribué et injection à chaud*, in "Conférence Francophone sur le Déploiement et la (Re) Configuration de Logiciels (DECOR'04), Grenoble, France", Octobre 2004, p. 65–76.

[35] O. MARÉCHAL, P. POIZAT, J.-C. ROYER. *Checking Asynchronously Communicating Components Using Symbolic Transition Systems*, in "On The Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE", R. MEERSMAN, Z. TARI (editors)., Lecture Notes in Computer Science, n° 3291, Springer-Verlag, 2004, p. 1502-1519.

[36] G. MULLER, J. LAWALL, J.-M. MENAUD, M. SÜDHOLT. *Constructing Component-Based Extension Interfaces in Legacy Systems Code*, in "ACM SIGOPS European Workshop 2004 (EW'2004), Louvain, Belgique", September 2004.

[37] J. NOYÉ, S. PAVEL, J.-C. ROYER. *A PVS Experiment with Asynchronous Communicating Components*, in "17th Workshop on Algebraic Development Techniques, Barcelona, Spain", March 2004.

[38] S. PAVEL, J. NOYÉ, J.-C. ROYER. *Dynamic Configuration of Software Product Lines in ArchJava*, in "Software Product Lines : Third International Conference, SPLC 2004", R. L. NORD (editor)., Lecture Notes in Computer Science, n° 3154, Springer-Verlag, Software Engineering Institute, August 2004, p. 90-109.

[39] P. POIZAT, J.-C. ROYER, G. SALAÜN. *Formal Methods for Component Description, Coordination and Adaptation*, in "Proceedings of the ECOOP Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04), Oslo, Norway", June 2004.

[40] L. RODRÍGUEZ, É. TANTER, J. NOYÉ. *Supporting Dynamic Crosscutting with Partial Behavioral Reflection: a Case Study*, in "14th International Conference of the Chilean Computer Society, Arica, Chile", G. NAVARRO (editor)., IEEE Computer Society, November 2004.

[41] L. RODRÍGUEZ, É. TANTER, J. NOYÉ. *Supporting Dynamic Crosscutting with Partial Behavioral Reflection: a Case Study*, P. COINTE, M. SÜDHOLT (editors)., October 2004, p. 118-137, http://www.emn.fr/x-info/obasco/events/jfdlpa04/actes/actes-jfdlpa04.pdf.

[42] J.-C. ROYER. *A Framework for the GAT Temporal Logic*, in "Proceedings of the 13th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE'04), Nice, France", ISCA, July 2004, p. 275-280.

[43] É. TANTER, J. NOYÉ. *Motivation and Requirements for a Versatile AOP Kernel*, in "European Interactive Workshop on Aspects in Software (EIWAS'04), Berlin, Germany", September 2004.

Internal Reports

[44] F. BAUDE, E. BRUNETON, T. COUPAYE, P.-C. DAVID, T. LEDOUX, A. OCCELLO, M. RIVEILL. *PROJET RNTL ARCAD - D1.3 Document d'architecture*, Research Report, ARCAD project, July 2004, http://arcad.essi.fr/livrables/d1-3.pdf.

[45] A. FARÍAS, M. SÜDHOLT. *Integrating protocol aspects with software components to address dependability concerns*, Research Report, n° 04/6/INFO, École des Mines de Nantes, November 2004.

[46] J. NOYÉ, E. BRUNETON, T. COUPAYE, D. HAGIMONT, J.-M. MENAUD, É. TANTER. *PROJET RNTL ARCAD - D1.4 Coût et optimisation*, Research Report, ARCAD project, March 2004, http://arcad.essi.fr/livrables/d1-4.pdf.

[47] É. TANTER, J. NOYÉ. *Versatile Kernels for Aspect-Oriented Programming*, Research Report, n° RR-5275, INRIA, July 2004, http://www.inria.fr/rrrt/rr-5275.html.

Miscellaneous

[48] C. AUGIER. *Extension de Bossa à la gestion de la ressource réseau*, Master's Thesis, École des Mines de Nantes/Université de Nantes, September 2004.

[49] T. FRITZ. *Sequence- and dataflow pointcuts for expressive aspect languages*, To appear, Technical report, Ludwig-Maximilians Universität München, December 2004.

[50] N. LORIANT. *Evolution dynamique des systèmes d'exploitation, une approche par AOP*, Master's thesis, Université d'Evry Val d'Essonne, September 2004.

[51] O. MARÉCHAL. *Algorithmes d'analyse des communications asynchrones entre composants*, Master's thesis, École des Mines de Nantes/Université de Nantes, August 2004.

[52] F. MINJAT. *Vers une modélisation transverse et modulaire des collaborations par couplage des Traits et des Classboxes*, Master's thesis, École des Mines de Nantes/Université de Nantes, August 2004.

[53] A. SEDKAOUI. *Implantation d'un modèle à composants en Java*, Master's thesis, Université d'Evry Val d'Essonne, August 2004.

[54] R. URUNUELA. *Extension du langage Bossa à la problématique de facturation et de gestion de l'énergie du processeur*, Master's thesis, École des Mines de Nantes/Université de Nantes, September 2004.

Bibliography in notes

[55] 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.

[56] 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.

[57] 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.

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

[59] A. GOLDBERG, D. ROBSON. *SMALLTALK-80. The language and its implementation*, Addison Wesley, 1983.

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

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

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

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

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

[65] 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.

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

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

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

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

[70] 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.

[71] 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.

[72] 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.

[73] 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.