INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team OBASCO

OBjects, ASpects and COmponents

Rennes

THEME 2A

Table of contents

1. Team

Figure 1.

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]

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]

Administrative Assistant
Sylvie Poizac [part-time (50%)]

Ph. D. Student
Hervé Albin-Amiot [Cifre grant with SODIFRANCE until January 2003]
Gustavo Bobeff [MINES contract]
Pierre-Charles David [MESR grant]
Simon Denier [EPA grant since September]
Hervé Duchesne [MINES grant since October]
Andrés Farías [ATER UNIVERSITY OF NANTES]
Yann-Gaël Guéhéneuc [IBM grant until May]
Sebastian Pavel [MESR grant, ACI DISPO, since October]
Marc Segura-Devillechaise [MINES grant]
Éric Tanter [grant from UNIVERSITY OF CHILE]
Luc Teboul [MINES grant]

Visiting Scientist
Julia Lawall [DIKU, UNIVERSITY OF COPENHAGEN, May to August 2003]

Technical Staff
Bil Lewis [June to July 2003]
Patricio Salinas [March to December 2003]

Student Intern
Rickard A. Åberg [October 2002 - March 2003]

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 [15]. 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[1]. Its primary goal is to investigate the possibility of a *continuum* between objects, aspects and components. We plan to study formal models of components and aspects to reason about adaptable systems. A natural result of our research is the realization of prototypes based on the investigation into new programming languages and paradigms suited to component-oriented systems with a particular emphasis on meta-programming.

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 expertise in operating system, active networks and middleware [18][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”.

[1]Great thanks to the reviewers: Isabelle Attali, Gérard Boudol, Yves Caseau, Jacky Estublier, Rachid Guerraoui, Michel Mauny and Jan Vitek.

3.2. Object-Oriented Languages

Glossary

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 [61]).

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* [58].

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 [49]).

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* [57].

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[2] 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* [58]. The basic idea, as initially explained by M.D. McIlroy in 1968 [50] 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).

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

- *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 [55], architecture description languages (ADLs) [56][51] and object-oriented languages. Modules provide an interesting support for component structure. In particular, recent proposals around so-called *mixin modules* combine parameterization, recursive module definitions, and late binding. ADLs address many of the above-mentioned issues although at a description, rather than programming, level. Finally, object-oriented languages remain a major source of inspiration. Interesting extensions have indeed been worked out in this context like notions of explicit protocols that can be seen as finite state automata but also integrated within the language as types. Recently, a number of *connection-oriented language* [58] 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 [47]) 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 [60]. 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 accord to 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 [44][48] to extend the base language with new mechanisms such as multiple-inheritance, concurrency or metaclasses composition [2].

- In the case of an 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 *interceptors/wrappers* 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 [46] 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*.

A first issue is now to have a better understanding of how to use reflective tools to model aspects languages and their associated crosscutting languages and advice languages [31]. 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

Glossary

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

A DSL is obviously more restricted than a general-purpose language, such as Java or even C, but encapsulates domain expertise making 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] [59].

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, etc.

In the application domain of operating systems, network and middleware, we are investigating 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. Application Domains

4.1. Overview

Key words: *telecommunication*, *enterprise information systems*.

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 [45].

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 [43][53].

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

Participants: Gilles Muller [correspondant], Rickard A. Åberg, Hervé Duchesne, Jean-Marc Menaud, Julia Lawall.

Key words: *OS*, *process scheduling*, *Linux*, *DSL*, *AOP*.

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

Emerging applications, such as multimedia applications and real-time applications, have increasingly specialized scheduling requirements. Nevertheless, developing a new scheduling policy and integrating it into an existing OS is complex, because it requires understanding (often implicit) OS conventions. Bossa is a kernel-level event-based framework to facilitate the implementation and integration of new scheduling policies [39][38]. Advantages of Bossa are:

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

Concretely, a complete Bossa kernel comprises three parts:

- A standard kernel, in which all scheduling actions are replaced by Bossa event notifications. The process of re-engineering a kernel for use with Bossa can be almost fully automated using AOP. 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 2.4.18 kernel so as to support Bossa [33][32].
- 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). J.-F. Susini hosted by the project from January to August 2003 has participated to this development.

5.2. EAOP

Participants: Mario Südholt [correspondant], Rémi Douence, Bil Lewis.

Key words: *Java*, *Recoder*, *Javassist*, *CPS*.

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 [6]:

- 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 aspect.
- EAOP supports an arbitrary number of aspect instances at runtime. 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 [23] 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. MicroDyner

Participants: Marc Segura-Devillechaise, Jean-Marc Menaud [correspondant], Gilles Muller, Julia Lawall.

Key words: *Web cache*, *Dynamic system evolution*, *AOP*, *C language*.

MicroDyner 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.

Given the high proportion of HTTP traffic in the Internet, Web caches are crucial to reduce user access time, network latency, and bandwidth consumption. Prefetching in a Web cache can further enhance these benefits. For the best performance, however, the prefetching policy must match user and Web application characteristics. Thus, new prefetching policies must be loaded dynamically as needs change.

Most Web caches are large C programs, and thus adding one or more prefetching policies to an existing Web cache is a daunting task. The main problem is that prefetching concerns crosscut the cache structure. Aspect-oriented programming is a natural technique to address this issue. Nevertheless, existing approaches either do not provide dynamic weaving, incur a high overhead for invocation of dynamically loaded code, or do not target C applications. When using the C language[3], MicroDyner provides a low overhead for aspect invocation, that meets the performance needs of Web caches [29][28][27].

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

5.4. Reflex

Participants: Eric Tanter, Jacques Noyé [correspondant], Pierre Cointe, Patricio Salinas, Simon Denier.

Key words: *Java*, *Javassist*, *metaobject protocol*, *behavioral reflection*, *partial reflection*.

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.

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.

[3] is a fist study to apply our approach to C++

- 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.emn.fr/x-info/reflex, and more information on the ideas behind Reflex can be found in [31]. Reflex has been used for teaching reflection at the Master level within our EMOOSE and DEA 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

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

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

At the theoretical level, we study two component model extensions: explicit interaction protocols including the notion of substitutability, and asynchronous and synchronous communications with property checking. On a more practical side, we work on techniques to better adapt component-based applications to their environment. We also experiment with a first integration of aspects and components.

6.1.1. Explicit Protocols

As part of his Ph.D. thesis, Andrés Faréas has developed the CwEP (Component with Explicit Protocols) model based on the principle of explicit interaction protocols. These interaction protocols are defined on the basis of finite-state automata. They provide explicit operations for the management of component identities for communication as well as security purposes, such as role-based access control [12].

This model includes a formal notion of substitutability of components derived from [54]. This notion of substitutability has been extended to validate the correctness notions between the specifications of such components and their respective implementations [24].

The EAOP model and especially its associated analysis techniques have been used to prototype an aspect language for the dynamic manipulation of the CwEP components and to formally investigated the preservation of the substitutability property.

6.1.2. Property Checking

We are also interested in more general ways to check properties about components and their associated architectures. In this approach the dynamic behavior of a component is represented as a structured symbolic transition system. We compute the global dynamic behavior of an architecture using the principle of the synchronous product. We have defined a general mixed technique [19] to formally describe the data part and the dynamic behavior of such a concurrent and communicating architecture. Previous experiences show that proof assistants like Larch Prover or PVS may allow us to prove general properties, even temporal logic ones [25].

Often concurrent systems consider an abstract point of view with synchronous communications. However it seems more realistic and precise to consider asynchronous communicating systems, since it provides a more primitive communication protocol and maximizes the concurrency. The technique of [19] is extended with buffer for queued messages, message emissions and message receipts. This extension provides a uniform model allowing asynchronous and synchronous communications. Asynchronous communications lead to more complex dynamic behavior, it increases the need for verification tools. We also define an algorithm [26] devoted to an analysis of the dynamic behavior of the system. This algorithm decides if the system has bounded mailboxes and computes the reachable mailbox contents of the system.

6.1.3. Adaptation

In [20] we have shown that adaptability is a key feature of components, and proposed a definition of components taking this feature into account. We have discussed the fact that the standard implementations of components are limited to a *shallow* adaptation of components, limited to wrapper code around the initial component implementation, which is left intact. We have shown that the implementation can be adapted, too, without breaking component encapsulation by relying on specialization techniques (more precisely, we are interested in combining partial evaluation and slicing). A key to not breaking encapsulation is then to use specialization scenarios, as introduced, in the context of C, by Le Meur et al. [52]. This can be complemented with the use of component generators in order to further decouple component production and use. We are currently working on a component model, language, and infrastructure based on these ideas (see S. Pavel's Ph.D. and the master theses [40][35]).

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 [21], we propose to build self-adaptive component-based applications to fit their evolving environment. We present the general approach we recommend and the corresponding Java development framework we developed to support it. The approach follows 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. Integrating Aspects with Components

We have validated the EAOP model [6] for e-commerce applications in the context of the European project EASYCOMP (see Sections 6.2 and 8.3). Concretely, we have integrated EAOP with the *Vienna Component Framework* (VCF) developed at TU Wien, Austria. This integration enables crosscutting concerns to be added dynamically to industrial component models, such as EJB and .NET components. Finally, a German compagny, H.E.I. GmbH, has developed an e-commerce application using VCF and EAOP. This case study has allowed the unanticipated introduction of an aspect for business rules (rebate policies) into an existing application.

6.2. Aspects

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

Key words: *separation of concerns*, *events*, *trace*, *monitor*, *join points*, *crosscutting*, *aspects weaving*, *static analysis*, *AspectJ*, *EAOP*, *Reflex*.

Glossary

(AspectJ) join points: are well defined points in the execution of a program.

(AspectJ) pointcuts: are a means of referring to collections of join points and certain values at those join points.

Crosscut: a specification of where aspect actions should be woven in a base program.

Advices a.k.a Actions: A specification of what an aspect computes. In AspectJ, they are method-like constructs used to define additional behavior at join points.

Aspect: a concern crosscutting a set of standard 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 a new program.

Aspect conflicts: two aspects may conflict, in particular when they share crosscuts.

We continue the development and the implementation EAOP model. We investigate its application in the field of OS. In order to compare the reflective and AOP approaches, we apply the Reflex infrastructure to an alternative implementation of EAOP.

Event-based AOP is an approach to aspect-oriented programming, based on the concept of a central monitor that receives synchronous events from join points in the application and possibly triggers some actions on sequences of related events [6]. The great expressive power of EAOP makes it possible to reason about events patterns, thus supporting temporal reasoning.

This model offers different variants according to the languages used to define crosscuts and advices/actions [17]. We currently consider two main variants. In the first variant, *static EOAP*, crosscuts definitions are restricted to regular expressions. This limits expression power but allows formal studies (*e.g.*, static analysis of interactions). In the second variant, *dynamic EAOP*, crosscuts are defined in a Turing complete language. This allows very expressive crosscuts to be defined, however aspect conflicts must be detected and resolved dynamically.

6.2.1. Static EAOP

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 a regular expression, but we allow equality constraints between several events to be expressed with variables. We have also developed more expressive support for conflict resolution that takes into account equality constraint variables and proposed a notion of enriched interface that makes explicit the family of base programs which can be woven with an aspect. This extension provides support for reuse of aspects [22].

This static version of EAOP has also been used in the Ph.D. thesis of Andrés Farías [12] in order to extend components with explicit protocols (see Section 6.1).

This research is currently pursued in two further directions: the definition of a general DSL for expressing crosscut definitions (Luc Teboul's Ph.D.) based on a generalization of the AspectJ's *cflow* operator, and the definition (in collaboration with S. Conchon hosted by the project from January to August 2003) of a calculus for EAOP according to the tradition of process calculi.

6.2.2. Dynamic EAOP

We have implemented a prototype for the EAOP model for Java. It was exemplified in [23] with a toy e-commerce application as a first demo. In order to deal with more realistic applications, the original prototype has been optimized by introducing a continuation passing style transformation for aspects. This prototype has

allowed us to show how different mechanisms such as aspect composition, aspect instantiation and aspect scope can be defined uniformly as general composition operators of aspects.

6.2.3. Reflection and EAOP

In [31], we have discussed the possibility of using a reflective infrastructure such as Reflex as a basis for providing runtime AOP, with the possibility of very easily introducing new join points and implementing crosscutting metaobjects. Then EAOP can be seen as a particular instantiation of partial behavioral reflection, where all hooks forward control to a unique omnipotent metaobject called the monitor (see Section 6.3). This work has allowed us to apply EAOP to base programs available in byte-code only and initiate a study about AOP and reflection.

6.2.4. Aspects in OS

Furthermore, we have applied the EAOP model in order to automatically evolve Linux kernel code to support the Bossa system (see Section 5.1). Concretely, we have defined an aspect for instrumentation of the Linux kernel with event generation statements for the Bossa runtime system. The aspect definition has been formalized in terms of rules in a temporal logic and the corresponding transformation system has been implemented [32][33].

In this approach, the transformation process is performed on the kernel source code at compile time. In the context of Web caches (see Section 5.3), we have developed another strategy for dynamically changing cache prefetch strategies without interrupting request servicing [29][28][27]. The evolution is then performed at binary level at run time. We are investigating on how to unify both approaches in a common framework.

6.3. Post-Objects

Key words: *opening*, *specialization*, *Java*, *java.lang.reflection*, *Reflex*, *design patterns*, *Eclipse*.

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

Glossary

to reify: making object.

to reflect: diving an object into the data flow.

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

temporal selection: consists in selecting when reifications are 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. Design Pattern Reifications

The two 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 one hand automatic code generation from a given model [11] and on the other hand detection of complete and distorted forms of design-pattern in Java sources [13].

These theses have developed the common PDL (Pattern Description Language) meta-model for the description of patterns. A first tool, PatternsBox, uses this meta-model to describe meta-patterns, to instantiate these meta-patterns into abstract patterns, to parameterize these abstract patterns, and to generate associated source

code. A second tool, Ptidej, uses the same meta-model to represent re-engineered libraries and collaborates with an explanation-based constraint solver to detect complete and distorted forms of patterns

6.3.2. 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 [31] 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).

The [30] explores how Reflex could be used to deal with interactive inspection. Work in progress includes:

- an implementation of the EAOP model in Reflex to demonstrate that dynamic/runtime AOP can be seen as a subset of partial behavioral reflection. More generally, using Reflex as an infrastructure to model in Java (and then to compare) different AOP approaches (this is S. Denier's Ph.D. work),
- explicit support for the roles of assembler and metalevel architect in addition to that of metaprogrammer (see E. Tanter's upcoming Ph.D. thesis).

6.3.3. Reflection and Aspects 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 [14], 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 plan to investigate TRAITS as defined in [42] to provide a clean Smalltak/Squeak MOP, which would then be used in turn to integrate traits, aspects and components in a uniform way.

6.4. AOP for OS Kernels

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

Participants: Gilles Muller, Thomas Ledoux, Jean-Marc Menaud.

6.4.1. Scheduler Policies

The Bossa framework has been publicly available for one year. It is fully compatible with the Linux 2.4.18 kernel and can be used as a direct replacement. Most of our work has been done to improve performance and 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 [39][38]. 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 [33][32] and Section 5.1). Additionally, we have investigated the extension of Bossa to multiprocessors [37] and the integration of Bossa in the .NET framework [36].

6.4.2. Prefetching and extensible Caches

Our work on MicroDyner has focused on the design and implementation of a robust prototype that can be made publicly available [29][28]. We are currently experimenting with applying MicroDyner to SQUID - the most used free-software Web cache - so as to dynamically change its internal cache strategies, such as prefetching. Preliminary evaluations show no performance degradation [27].

7. Contracts and Grants with Industry

7.1. IBM Eclipse Fellowships

Participants: Pierre Cointe, Yann-Gaël Guéhéneuc.

Guéhéneuc's thesis [13] explores models and algorithms to identify semi-automatically micro-architectures in source code, which are similar to design-patterns and to ensure their traceability between implementation and reverse engineering phases. Metamodeling is used to describe design-patterns and Java programs. It leads to characterize certain interclass relations (association, aggregation and composition) offered by design languages such as UML, to precise their properties (access type, lifetime, exclusivity and multiplicity) and to identify them with static and dynamic analyses. It also leads to translate design-patterns into constraint systems and to identify micro-architectures, which are similar to design-patterns (complete and distorted forms), by solving constraint satisfaction problems. Explanation-based constraint programming allows guiding the solve interactively and explaining identified micro-architectures.

The Ptidej tool suite implements the proposed models and algorithms and is publicly available at http://www.emn.fr/x-info/ptidej. Ptidej has been supported by an IBM Eclipse Fellowships grant of 35 000 US $ for the period July 2002 to December 2003 (see http://www.eclipse.org/technology/research.html).

7.2. Microsoft Research

Participants: Gilles Muller, Jean-Marc Menaud, Mario Südholt, Rickard A. Åberg, Arnaud Denoual.

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 [32][33] and the integration of Bossa within the .NET framework [36].

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

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 FRE between CNRS, University of Nantes and École des Mines de Nantes.

OBACO partipates with the Triskell and Atlas teams in the *Club Objet de l'Ouest* which fosters cooperation in object technologies between public research laboratory and industry.

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.

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, Patricio Salinas.

The ARCAD project is an RNTL project running from December 2000 to December 2003 with a 107 Keuros funding for our team.

The project federates 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.

The goal of ARCAD is to propose a component-based software architecture enabling the building of distributed adaptable applications. Experiments are realized in the ObjectWeb context with JOnAS and Fractal (see http://www.objectweb.org).

In 2003, the main results were REFLEX [31] (see also Section 5.4), a DSL for *Fractal* to describe adaptation policy [21] (see also Section 6.1), and a new *Fractal ADL*.

The default *Fractal ADL*, developed by France Telecom R&D, uses an XML syntax to describe the initial architecture of a Fractal application. This makes the language very flexible, easily extensible, and amenable to automated processing using XML technologies (like XSLT). But even with the appropriate tools like XML-aware editors, simple architecture descriptions can quickly become difficult to read, understand and maintain. P.-C. David has developed an alternative language (and the corresponding parser) with a much more concise and readable syntax but without extra cost induced by component sharing, attribute values specification, external templates, ...This ADL is publicly available at: http://fractal.objectweb.org/tutorials/simple-adl/index.html.

8.2.3. Action incitative CORSS

Participants: Gilles Muller, Jean-Marc Menaud.

The aim of this 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, disk and network.

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, Mario Südholt.

The aim of this 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. EASYCOMP IST Project

Participants: Mario Südholt, Gustavo Bobeff, Pierre Cointe, Rémi Douence, Jacques Noyé, Andrés Farías.

The EASYCOMP project is an IST project running from June 2000 to November 2003 funded by the European Union (350 Keuros for our team). The goal of this project is to provide a uniform composition model and corresponding infrastructure facilitating composition of software artifacts over all of the component lifecycle (see http://www.easycomp.org). The project federates work by 10 partners, 7 academic and 3 industrial ones, from 6 countries: apart from EMN, University of Karlsruhe, University of Kaiserslautern, Linköping University, Twente University, Technical University of Vienna, Budapest University of Technology and Economics, H.E.I. GmbH, ILOG S.A., and Q-Labs S.A.

In 2003, OBASCO group has produced two kinds of result. First, the EAOP model has been completed and a corresponding publicly-available tool developed (see Section 5.2). Second, program specialization techniques in the context of component-based software engineering have been investigated.

8.3.2. ELASTEX ALFA Project

Participant: Jacques Noyé.

The European project ELASTEX (European And Latin American Students EXchange) of the ALFA (*América Latina, Formación Académica*) programme gathers ten partners from Latin America and Europe: *Universidad de Chile* (Chile), *Universidade Federal do Rio de Janeiro* (Brazil), *Pontificia Universidade Católica do Paraná* (Brazil), *Universidad Nacional de la Plata* (Argentina), *Universidad de Los Andes* (Columbia), *Vrije Universiteit Brussel* (Belgium), *Universidade Nova de Lisboa* (Portugal), *Universiteit Twente* (The Netherlands). The objectives of the project were to contribute to the organization of the EMOOSE Master program and to strengthen the scientific relationship between the partners through teacher and student exchanges. This project, started in September 2001, came to an end in August 2003.

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. He also chairs an *Action Spécifique* on methodologies and tools for the design of operating 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 Vannes (see http://lmo.iu-vannes.fr/).

9.1.1.4. OCM 2003:

Following the Objet XX and OCM XX series, OCM 2003 was organized this year in Rennes, on June 5th, in cooperation with the Triskell and Atlas projects. The theme was *Software migration and software interoperability*. See http://www.ocm-ouest.org for more details.

9.1.1.5. Les jeudis de l'objet:

This bimonthly industrial seminar organized by our group is now seven years old. Together with the annual conference OCM, it has become a great place for local industry to learn, exchange ideas, 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, 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 is a member of the *RSTI L'Objet* journal editorial board. He has reviewed papers for TOPLAS and TSI.

He co-organized the workshop on Software Composition in the context of ETAPS 2003 in Warsaw with the main partners of the EASYCOMP project (U. Assmann, E. Pulvermueller, I. Borne and N. Bouraqadi). He is a program committee member of the OOPS special technical track at the 19th ACM Symposium on Applied Computing (see http://oops.disi.unige.it).

He will serve as a program committee member of the eTX (eclipse Technology Exchange) and SC (Software Composition) workshops at ETAPS (Barcelona, March/April 2004, see http://www.lsi.upc.es/etaps04). He will be a track leader, on Generative Programming and Aspects, at the Unconventional Programming Paradigms workshop (Saint-Malo, September 2004, see http://upp.lami.univ-evry.fr/).

9.1.2.2. T. Ledoux:

He reviewed papers for the TSI special issue on *Systèmes à composants adaptables et extensibles*. He was a program committee member of the LMO 2003 conference (*Langages et Modèles à Objets*, Vannes, February 2003).

9.1.2.3. G. Muller:

He was a program committee member of DSN 2003 (IEEE Conference on Dependable Systems and Networks, San Francisco, June 2003), CFSE-3 (3rd French Conference on Operating Systems, La Colle sur Loup, October 2003), DAIS 2003 (4th IFIP International Conference on Distributed Applications & Interoperable Systems, Paris, November 2003).

He was a jury member for the award of the best French Ph.D. in operating systems issued in October 2003 during CFSE-3, and a jury member for the *Atlantitiel* Innovation Award, which was awarded to four innovating companies in the *Pays de la Loire* council.

He is a program committee member of DSN 2004 (IEEE Conference on Dependable Systems and Networks, Firenze, June 2004), and FTDCS 2004 (International Workshop on Future Trends of Distributed Computing Systems, Suzhou, China, May 2004).

9.1.2.4. J. Noyé:

He was a program committee member of LMO 2003 (*Langages et Modèles à Objets*, Vannes, February 2003). He is a program committee member of LMO 2004 (Lille, March 2004)

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).

9.1.2.6. M. Südholt:

He was a program committee member of AOSD 2003 (Aspect Oriented Software Design, Boston, March 2003), and a program committee member of the workshops ACP4IS at AOSD 2003, and SC at ETAPS 2003. He is a program committee member of AOSD 2004 (Lancaster, March 2004), and of the workshops FOAL at AOSD 2004 and SC at ETAPS 2004.

9.1.3. Thesis committees

9.1.3.1. P. Cointe:

He has been the scientific advisor of Hervé Albin-Amiot (thesis defense on 6/2/2003) [11] and Yann-Gaël Guéhéneuc (thesis defense on 23/6/2003) [13]. He reviewed the HDR of M. Huchard (U. Montpellier/LIRMM, 4/04/03), and participated in the HDR committees of O. Boissier (EMSE/U. StEtienne, 12/2/03) and N. Beldiceanu (University of Paris 6, 28/4/03). He served in the Ph.D. committees of T. Quinot (University of Paris 6, 24/3/03) and Andrès Farías (EMN/U. of Nantes, 18/12/2003).

9.1.3.2. T. Ledoux:

Was chair of the Ph.D. committee of Victor Budau (INT Evry, 19/11/2003).

9.1.3.3. G. Muller:

He has been the advisor of Luciano P. Barreto (thesis defense on 30/8/2003, U. of Rennes I/INRIA Compose group). He reviewed the Ph.D. of S. Patarin (U. of Paris 6/LIP6, 4/6/2003), the Ph.D. of A. Sénart (INPG, 6/11/2003), and the Ph.D. of Christian Khoury (U. of Paris 6/LIP6, 12/2003).

9.1.3.4. J-C. Royer:

He participated in the HDR of M. Huchard (U. Montpellier/LIRMM, 4/04/03).

9.1.3.5. M. Südholt:

He has been the scientific advisor of A. Farías (thesis defense on 18/12/2003) [12].

9.1.4. Evaluation committees and expertise

9.1.4.1. P. Cointe

He was a member of the *Grand Jury* organized by France Telecom R&D in charge of evaluating the theme "Architectures and Software Infrastructures" in October. As INRIA, he was a member of the CR2 hiring committee organized by the Rhônes Alpes UR (Grenoble, June 2003) and reviewed one proposal for the ACI *Security*. As a member of the MSTP, he was in charge of evaluating proposals for the ACI *Young Researchers*. He evaluated some LIRMM proposals submitted to the *Languedoc Roussillon* council.

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 network of partners, SENSO, includes the partners of the ELASTEX project (see 8.3) together with the *Technische Universität Darmstadt* (Germany) and the Monash University (Australia). The program is managed by the team 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 fifth promotion graduated in September 2003 while the sixth promotion was about to start their first semester. See also: http://www.emn.fr/EMOOSE.

9.2.2. DEA infomatique de Nantes

The faculty members of the team participate to this master program and give lectures about new trends in the field of object-oriented software engineering.

EMOOSE and the DEA are two opportunities to welcome student interns for their master thesis:

- DEA (February - July 2003):
  Arnaud Denoual, Sebastian Pavel and Hervé Duchesne [36][40][37].
- EMOOSE (February - July 2003):
  Diego De Sogos, Kaiye Xu and Yan Chen [35][41][34].

9.3. Collective Duties

9.3.1. P. Cointe:

He is chairman of the Computer Science Department at EMN and as such the co-chair of the *pôle informatique* associated to the CPER 2000-2006. He is also a member of the MSTP (*Mission Scientifique Technique Pédagogique*) since March 2003. He was a member of the CNU (the French *University National Council*) until September 2003.

10. Bibliography

Major publications by the team in recent years

[1] M. AKSIT, A. BLACK, L. CARDELLI, P. COINTE, AL. *Strategic Research Directions in Object Oriented Programming.* in « ACM Computing Surveys », number 4, volume 28, December, 1996, pages 691-700.

[2] M. BOURAQADI-SAÂDANI, T. LEDOUX, F. RIVARD. *Safe Metaclass Programming.* in « Proceedings of OOPSLA 1998 », number 10, volume 33, ACM-Sigplan, ACM Press, C. CHAMBERS, editor, pages 84–96, Vancouver, British Columbia, USA, October, 1998.

[3] P. COINTE. *Les langages à objets.* in « Technique et Science Informatique », number 1-2-3, volume 19, 2000, pages 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) », number 12, volume 22, ACM Press, J. L. ARCHIBALD, editor, pages 156–167, Orlando, Florida, USA, October, 1987.

[5] R. DOUENCE, P. FRADET, M. SÜDHOLT. *A framework for the detection and resolution of aspect interactions.* in « Proceedings of the 1st ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02) », pages 173-188, Pittsburgh, USA, October, 2002.

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

[7] T. LEDOUX. *OpenCorba: a Reflective Open Broker.* in « ACM Meta-Level Architectures and Reflection, Second International Conference, Reflection'99 », series Lecture Notes in Computer Science, volume 1616, Springer-Verlag, P. COINTE, editor, pages 197–214, Saint-Malo, France, July, 1999.

[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 », USENIX Association, pages 17–30, San Diego, California, October, 2000.

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

Books and Monographs

[10] P. COINTE, editor, *Les langages à objets.* series Traité multi-volumes sur les sciences de l'ingénieur, série 4C., Hermès, to appear.

Doctoral dissertations and "Habilitation" theses

[11] H. ALBIN-AMIOT. *Idiomes et patterns Java: Application à la synthèse de code et à la détection.* Ph. D. Thesis, École des Mines de Nantes and Université de Nantes, February, 2003.

[12] A. FARIAS. *Un modèle de composants avec des protocoles explicites.* Ph. D. Thesis, École des Mines de Nantes and Université de Nantes, December, 2003.

[13] Y.-G. GUÉHÉNEUC. *Un cadre pour la traçabilité des motifs de conception.* Ph. D. Thesis, École des Mines de Nantes and Université de Nantes, June, 2003.

Articles in referred journals and book chapters

[14] N. M. BOURAQADI-SAÂDANI, T. LEDOUX. *Supporting AOP Using Reflection.* M. AKSIT, S. CLARKE, T. ELRAD, R. E. FILMAN, editors, in « Aspect-Oriented Software Development », Addison-Wesley, 2003, to appear.

[15] 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, colloque en l'honneur de Jean-François Perrot », number 4, volume 10, http://www.lip6.fr/colloque-JFP/, to appear.

[16] R. DOUENCE, P. FRADET. *The next 700 Krivine machines.* in « Higher-Order and Symbolic Computation », to appear.

[17] R. DOUENCE, P. FRADET, M. SÜDHOLT. *Trace-Based Aspects.* M. AKSIT, S. CLARKE, T. ELRAD, R. E. FILMAN, editors, in « Aspect-Oriented Software Development », Addison-Wesley, 2003, to appear.

[18] G. MULLER, J. LAWALL, S. THIBAULT, R. E. V. JENSEN. *A Domain-Specific Language Approach to Programmable Networks.* in « IEEE Transactions on Systems, Man and Cybernetics », number 3, volume 33, August, 2003, pages 370–381.

[19] J.-C. ROYER. *The GAT Approach to Specify Mixed Systems.* in « Informatica », number 1, volume 27, 2003, pages 89–103, http://ai.ijs.si/informatica/, ISSN 0350-5596.

Publications in Conferences and Workshops

[20] G. BOBEFF, J. NOYÉ. *Modeling Components using Program Specialization Techniques.* in « Eighth International Workshop on Component-Oriented Programming », J. BOSCH, C. SZYPERSKI, W. WECK, editors, Darmstadt, Germany, July, 2003, In conjunction with ECOOP 2003.

[21] P.-C. DAVID, T. LEDOUX. *Towards a Framework for Self-Adaptive Component-Based Applications.* in « Proceedings of DAIS'03 », series Lecture Notes in Computer Science, Federated Conferences, Springer-

Verlag, pages 1–14, Paris, November, 2003.

[22] R. DOUENCE, P. FRADET, M. SÜDHOLT. *Composition, Reuse and Interaction Analysis of Dynamic Aspects.* in « Proceedings of the 3rd International Conference on Aspect-Oriented Software Development (AOSD 2004) », ACM Press, Lancaster, UK, March, to appear.

[23] R. DOUENCE, M. SÜDHOLT. *Un modèle et un outil pour la programmation par aspects événementiels.* in « LMO 2003 », Hermès, pages 105–117, Vannes, February, 2003, Version anglaise disponible en rapport interne.

[24] A. FARIAS, Y.-G. GUÉHÉNEUC. *On the Coherence of Component Protocols.* in « Workshop on Software Composition associated to ETAPS 2003 », number 5, volume 82, Elsevier Science (Electronic Notes in Theoretical Computer Science), U. ASSMANN, E. PULVERMUELLER, I. BORNE, N. BOURAQADI, P. COINTE, editors, April, 2003.

[25] J.-C. ROYER. *Temporal Logic Verifications for UML: the Vending Machine Example.* in « RSTI - L'objet, 4th Rigorous Object-Oriented Methods Workshop », number 4, volume 9, Lavoisier, K. LANO, A. EVANS, T. CLARK, editors, pages 73–92, 2003.

[26] J.-C. ROYER, M. XU. *Analysing Mailboxes of Asynchronous Communicating Components.* in « On The Move to Meaningful Internet Systems 2003: Coopis, DOA, and ODBASE », series LNCS, volume 2888, Springer Verlag, D. C. S. R. MEERSMAN, AL., editors, pages 1421-1438, 2003.

[27] M. SÉGURA-DEVILLECHAISE, J.-M. MENAUD, J. LAWALL, G. MULLER. *Extensibilitée Dynamique dans les Caches Web : une Approche par Aspects.* in « 3ème Conférence Française sur les Systèmes d'Exploitation (CFSE'03) », pages 477–487, La Colle sur Loup, October, 2003.

[28] 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 », ACM Press, pages 110–119, Boston, Massachusetts, USA, March, 2003.

[29] M. SÉGURA-DEVILLECHAISE, J.-M. MENAUD. *MicroDyner : Un noyau efficace pour le tissage dynamique d'aspects sur processus natif en cours d'exécution.* in « LMO 2003 », Hermès, pages 109–133, Vannes, February, 2003.

[30] E. TANTER, P. EBRAERT. *A Flexible Approach to Interactive Runtime Inspection.* in « ECOOP Workshop on Advancing the State-of-the-Art in Runtime Inspection (ASARTI 2003) », Darmstadt, Germany, July, 2003.

[31] 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) », number 11, volume 38, ACM Press, R. CROCKER, G. L. STEELE, JR., editors, pages 27–46, Anaheim, California, USA, October, 2003.

[32] R. A. ÅBERG, J. LAWALL, M. SÜDHOLT, G. MULLER, A.-F. LE MEUR. *On the automatic evolution of an OS kernel using temporal logic and AOP.* in « Proceedings of the 18th IEEE International Conference on Automated Software Engineering 2003 (ASE 2003) », pages 196–204, Montreal, Canada, October, 2003.

[33] R. A. ÅBERG, J. LAWALL, M. SÜDHOLT, G. MULLER. *Evolving an OS Kernel using Temporal Logic and Aspect-Oriented Programming.* in « The Second AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS) », March, 2003.

Internal Reports

[34] Y. CHEN. *Aspect-Oriented Programming (AOP): Dynamic Weaving for C++.* Technical report, EMOOSE: Vrije Universiteit Brussel and École des Mines de Nantes, August, 2003.

[35] D. DE SOGOS. *Component Generators: Towards Adaptable and Efficient Software Components.* Technical report, EMOOSE: Vrije Universiteit Brussel and École des Mines de Nantes, August, 2003.

[36] A. DENOUAL. *Intégration de Bossa dans .NET.* Technical report, DEA: Université et École des Mines de Nantes, September, 2003.

[37] H. DUCHESNE. *Conception d'ordonnanceurs pour systèmes multiprocesseurs.* Technical report, DEA: Université et École des Mines de Nantes, September, 2003.

[38] J. L. LAWALL, G. MULLER, A.-F. L. MEUR. *Domain-Specific Verification for Efficient Operating System Extensions.* Technical report, number 03/03/INFO, École des Mines de Nantes, 2003.

[39] G. MULLER, J. L. LAWALL, L. P. BARRETO, J.-F. SUSINI. *A Framework for Simplifying the Development of Kernel Schedulers: Design and Performance Evaluation.* Technical report, number 03/2/INFO, École des Mines de Nantes, 2003.

[40] S. PAVEL. *Lignes de Produits Logiciels en ArchJava (Software Product Lines in ArchJava).* Technical report, DEA: Université et École des Mines de Nantes, September, 2003.

[41] K. XU. *Analysis and Implementation Asynchronous Component Model.* Technical report, EMOOSE: Vrije Universiteit Brussel and École des Mines de Nantes, August, 2003.

Bibliography in notes

[42] 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) », ACM Press, R. CROCKER, G. L. STEELE, JR., editors, pages 47–64, Anaheim, California, USA, October, 2003.

[43] 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 », ACM SIGPLAN Notices, 31(5), pages 237–248, Philadelphia, PA, May, 1996.

[44] P. COINTE. *CLOS and Smalltalk - A comparison.* A. PAEPCKE, editor, in « Object-Oriented Programming: The CLOS Perspective », MIT PRESS, 1993, chapter 9, pages 216-250.

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

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

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

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

[49] 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 », series Lecture Notes in Computer Science, volume 1241, Springer-Verlag, M. AKSIT, S. MATSUOKA, editors, pages 220–242, Jyväskylä, Finnland, June, 1997.

[50] M. MCILROY. *Mass produced software components.* in « Proceedings of the NATO Conference on Software Engineering », NATO Science Committee, P. NAUR, B. RANDELL, editors, pages 138-155, Garmish, Germany, October, 1968.

[51] N. MEDVIDOVIC, R. TAYLOR. *A Classification and Comparison Framework for Software Architecture Description Languages.* in « IEEE Transactions on Software Engineering », number 1, volume 26, January, 2000, pages 70-93.

[52] A. L. MEUR, C. CONSEL, B. ESCRIG. *An Environment for Building Customizable Software Components.* in « IFIP/ACM Working Conference - Component Deployment », Springer-Verlag, pages 1-14, Berlin, Germany, June, 2002.

[53] 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) », pages 19–24, Kolding, Denmark, September, 2000.

[54] O. NIERSTRASZ. *Regular Types for Active Objects.* O. NIERSTRASZ, D. TSICHRITZIS, editors, in « Object-Oriented Software Composition », Prentice Hall, 1995, chapter 4, pages 99-121.

[55] R. PRIETO-DIAZ, J. NEIGHBORS. *Module Interconnection Languages.* in « The Journal of Systems and Software », number 4, volume 6, November, 1986, pages 307-334.

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

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

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

[59] S. THIBAULT, C. CONSEL, G. MULLER. *Safe and Efficient Active Network Programming.* in « 17th IEEE Symposium on Reliable Distributed Systems », pages 135–143, West Lafayette, IN, October, 1998.

[60] D. THOMAS. *Reflective Software Engineering - From MOPS to AOSD.* in « Journal Of Object Technology », number 4, volume 1, October, 2002, pages 17-26, http://www.jot.fm/issues/issue_2002_09/column1.

[61] 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) », number 12, volume 22, ACM Press, J. L. ARCHIBALD, editor, pages 168–182, Orlando, Florida, USA, October, 1987.