Object-Oriented Frameworks for Distributed Systems : A Survey

Wai Ming Ho, Jean-Marc Jézéquel

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

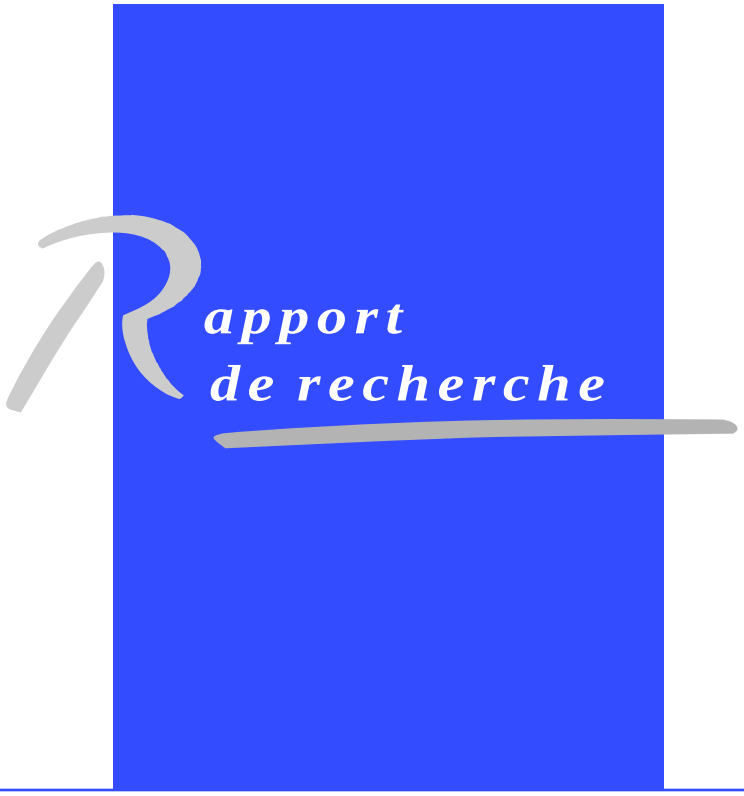
Object-Oriented Frameworks for Distributed Systems : A Survey

Wai Ming Ho , Jean-Marc Jézéquel

N ° 3590

Decembre 1998

THÈME 1



ISSN 0249-6399

Object-Oriented Frameworks for Distributed Systems : A Survey

Wai Ming Ho* , Jean-Marc Jézéquel*

Thème 1 — Réseaux et systèmes
Projet PAMPA

Rapport de recherche n3590 — Decembre 1998 — 22 pages

Abstract: Object-oriented frameworks are gaining importance to help reduce development efforts in large complex systems. They help developers leverage the knowledge of experienced domain experts, thus reducing the complexity of the development of large systems. Distributed applications are inherently complex and are therefore difficult to develop. Frameworks are used to hide away these complex issues, freeing the developer to concentrate on the application requirements instead. This paper will present an overview of object oriented frameworks and describe a few example frameworks targetted for distributed applications development. It will conclude by stating future trends in frameworks and a suggestion of promising areas of interest for research.

Key-words: Frameworks, Object-Oriented, Distributed Systems, Parallel Computing

(Résumé : tsvp)

This project is supported by CNET/France Telecom. Contract No. *981B070/1*

* {waimingh, jezequel}@irisa.fr

Unit´e de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
T´el´ephone : 02 99 84 71 00 - International : +33 2 99 84 71 00
T´el´ecopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Des Frameworks Orienté-Objets pour les Systèmes Répartis : Une Étude

Résumé : Les frameworks orienté-objets prennent de l'importance pour aider à réduire les efforts de développement de systèmes complexes. Ils aident les développeurs à réutiliser la connaissance des experts de domaine, de ce fait réduisant la complexité du développement de gros systèmes. Les applications réparties sont en soi complexes et sont donc difficules à developper. Des frameworks sont employés pour cacher ces issues complexes, libérant le developpeur qui pourra se concentrer sur les besoins de l'application à la place. Cet article présente une vue d'ensemble des frameworks orienté-objets et décrit quelques exemples de framework pour le développement d'applications distribués. Nous conclurons en énonçant les futures tendances dans les frameworks et en proposant des pistes de recherche prometteuses.

Mots-clé : Frameworks, Orienté Objet, Systèmes Répartis, Calcul Parallèle

Contents

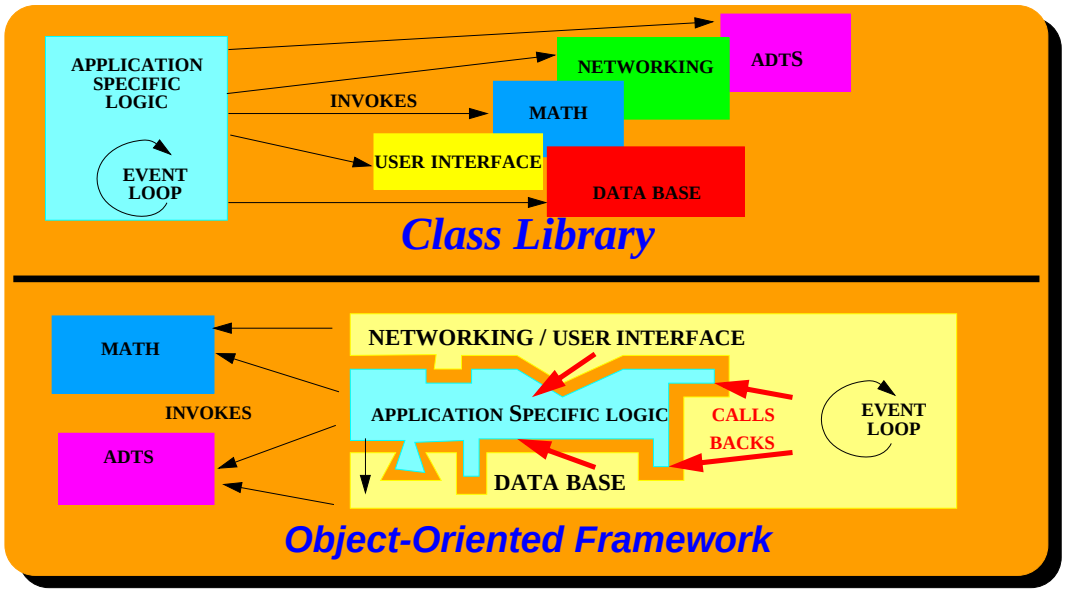


Figure 1: Difference between frameworks and class libraries

1 Introduction

An object-oriented software framework is made of a set of related classes which can be specialized or instantiated to implement an application. It is a reusable software architecture that provides the generic structure and behavior for a family of software applications, along with a context which specifies their collaboration and use within a given domain [5].

A framework differs from a complete application in that it lacks the necessary application-specific functionality. It can be considered as a prefabricated structure, or template, of a working application, where a number of pieces in specific places, called *plug-points* or *hot spots*, are either not implemented or given overridable implementations. To obtain a complete application from a framework, one has to provide the missing pieces, usually by implementing a number of call-back functions (that is, functions that are invoked by the framework) to fill the plug-points. In an object-oriented context, this feature is achieved by the dynamic binding: an operation can be defined in a library class but implemented in a subclass in the application specific code. A developer can thus customize the framework to a particular application by subclassing and composing instances of framework classes [20].

A framework is thus different from a classical class library in that the flow of control is most often bi-directional between the application and the framework (see Figure 1). The framework is in charge of managing the bulk of the application, and the application programmer just provides various bits and pieces. A bit like when programming some event driven applications, the application programmer usually has no control over the main control logic of the code.

Design patterns can be used to document the collaborations between classes in a framework. Conversely, a framework uses several design patterns, some of them general purpose, some of them domain-specific. Design patterns and frameworks are thus closely related, but they are not at the same level: a framework *is made of* software, whereas design patterns represent knowledge, information and experience *about* software. In this respect, frameworks are of a physical nature, while patterns are of a logical nature: frameworks are the physical realization of one or more software pattern solutions; patterns are the instructions for how to implement those solutions.

Application frameworks are important in the development of complex software systems because it helps to encapsulate the application domain complexities. The application developer can focus on the task of satisfying an application's requirement without worrying about too much *how* the requirements can be specified. The application framework provides a set of reusable components that the developer can customize to realize the requirements of the application.

In the field of distributed computing, frameworks play an important role in hiding the complexities of networking, distribution, execution environment and platform dependencies. For example, the Adaptive Communication Environment [46] encapsulates details of low level network programming and operating system services so that developers can concentrate on fulfilling their application requirements. The Eiffel Parallel Execution Environment [27] encapsulate details of parallelism so that developers of numerically intensive applications do not have to concern themselves with details of optimization of parallel architectures.

Therefore, besides reuse, frameworks also simplify systems development and leverage application domain knowledge. The next sections of this paper will attempt to provide a more detailed overview of frameworks. The first part discusses classification of frameworks and their characteristics. Next, it examines some techniques for constructing frameworks. Then, the third part will explore the characteristics of several existing distributed computing frameworks. Finally, it concludes with a summary of future trends in object oriented frameworks and the identification of promising research directions.

2 Classification and Characteristics of Frameworks

2.1 Black/White/Gray box

The reuse or extensibility of a framework provides a simple basis for broadly classifying frameworks. The two extreme categories that results from this classification are black box and white box frameworks [18]. A black box framework is one that supports re-usability or extensibility by composing various framework components (see Figure 2(left)). Here, the application classes (normal square boxes) are composed of various the framework components (square boxes with a line inside, enclosed inside a dashed rectangle). Every framework component addressing a specific domain has a well-defined interface. A set of components sharing the same interface but providing different results on its operations gives variability to cover differing applications requirements. This is an application of the *Strategy* [20] or *Functor* design pattern. Each specific implementation of a component serves as a "strategy" that the client application can use to realize some part of its requirements. At the other extreme of this classification is the white box framework. The mechanism

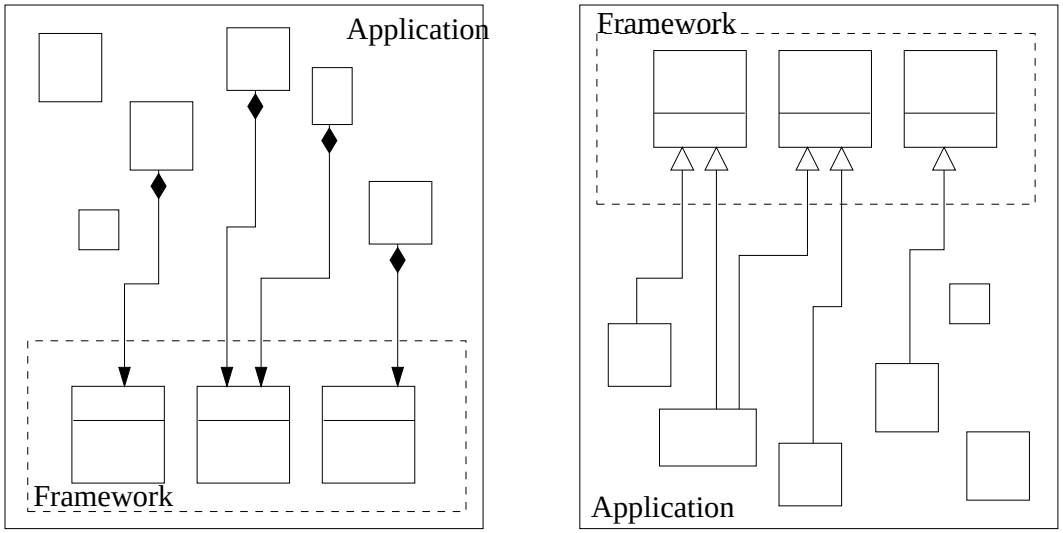


Figure 2: Black Box vs. White Box Reuse

employed here is class inheritance (see Figure 2(right)). White box frameworks contain classes that are incomplete. These classes may contain one or more abstract interface or empty methods. Client applications must provide a meaningful *body* to these framework classes through an inherited derived class. These incomplete methods are termed as *hook methods* or *hot spots* because they are points within the framework where a developer "attaches" application specific code to produce a complete application. [18] notes that hook methods is a safe and systematic way to decouple the stable interfaces of the application framework from the variations in each application instantiation. In design pattern terminology, this is an application of the *Template* [20] pattern. The empty or incomplete framework classes forms a kind of template for potential applications. Application developer using the framework must fill in the implementation for these skeletal structures to complete the framework application.

Most existing frameworks are however, "Gray", as they are not strictly black, nor strictly white. They are a combination of inheritable and composable parts. Generally, white box reuse requires a good understanding of the framework, and tends to produce tightly coupled systems. Black box reuse, on the other hand, requires less knowledge of internal framework structure, and tends to result in loosely coupled systems. However, it is harder to build a black box framework as its designer will have to define interfaces that anticipate a wider reuse potential [18].

2.2 Enterprise/Middle-ware Integration/System Infrastructure

An alternative view of frameworks is to look at its area of application or problem domain. There are generally three main categories for this way of classification [18]:

- System Infrastructure
- Middle-ware Integration
- Enterprise Application

A system infrastructure framework is built to provide a portable and efficient abstraction of the underlying operating environment (operating system, user interfaces, communications etc). It is generally used as internal tools for software development. Examples of system infrastructure framework are the Eiffel Parallel Execution Environment (EPEE) [27], the Adaptive Communication Environment (ACE) [46] and graphical user interface frameworks like ET++ [1] and InterViews [31]. EPEE, for example, frees the developer from the concerns of parallel optimization while ACE provides a simple interface for the development of application using low level operating system and communication services. Similarly, ET++ and InterViews provides an easy to use framework for the creation of graphical user interface (GUI) applications.

Middleware integration frameworks provide a seamless way to integrate, reuse and extend software in a distributed environment. The main concern in most distributed environments is the heterogeneity of the operating platform and system architecture. Therefore, developers of applications for such environments faces many problems on interoperability and compatibility issues. Thus, integration frameworks like the Common Object Request Broker Architecture (CORBA) aims to reduce these complexities by encapsulating the heterogeneous nature of the distributed environment. The Computer Integrated Manufacturing (CIM) framework [17] also shows the role of integration framework in providing interoperability for software developed by different companies.

Finally, enterprise frameworks attempts to address specific business applications domain, for example, telecommunications, manufacturing or banking. The Gebos system [7] is built from a layered banking framework. The construction of such a framework is difficult and expensive [18] because of the high demands of domain specific knowledge for those developing the framework. Of course, the benefit of such an investment is reduced effort in application development for the business domain concerned. For example, Gebos makes it possible to adapt a new banking application for a new bank in a relatively short time [7].

2.3 Application Control Flow

Contrary to conventional application flow, application developed from frameworks generally experience an “Inversion of Control” [18]. This means that the framework defines the control flow of the application. The role of the application is to redefine the hook methods to perform application specific tasks. There are, however, a number of frameworks that are termed “callable” frameworks. Such frameworks allow the application to retain the control flow and the role of the framework is to

Framework	Type	Domain	Control Flow
EPEE	Gray	Parallel Computing System Infrastructure	Callable
CORBA	White	Distributed Application Middleware Integration	Callable on client. Inversion of control on server
ACE	White	Communications System Infrastructure	Callable
BAST	Gray	Reliable Distributed Object Middeware Integration	Callable
CIM	White	Manufacturing System Middleware Integration	Callable
POOMA	Gray	Parallel Computing System Infrastructure	Callable
ET++	Gray	Graphical User Interface System Infrastructure	Inversion of Control
InterViews	Black	Graphical User Interface System Infrastructure	Callable

Table 1: Classfication of Several Existing Frameworks

provide services to the application. An example is the CIM Framework [17] developed by SEMATECH, the semiconductor manufacturing technology consortium. Table 1 provides a summary of the classification of several well-known frameworks.

2.4 Documentation

As mentioned earlier, most existing frameworks are "gray". In order to extend or re-use it, the developer has to understand the architecture of the framework. He or she must know how the classes interact and the constraints that exist among the classes to ensure correct implementation of an application from the framework. Therefore, there must be adequate documentation to help developers make proper use of the framework concerned. However, documenting a framework is a major task because it has a large class hierarchy that contains many internal behaviors, interactions and hidden dependencies. These issues are particularly important for white box reuse. [11] presented an example of how inadequate documentation can give rise to a condition called *inconsistent method* [30]. In Figure 3, when the implementation of a method `process_all` is changed in the based class, the derived class' assumption that `process_all` calls `process` is invalid. Thus, `log_action` will not be called as originally intended. Therefore, `process` and `process_all` in the derived class have become inconsistent.

1. Cookbook and recipes
2. Example applications

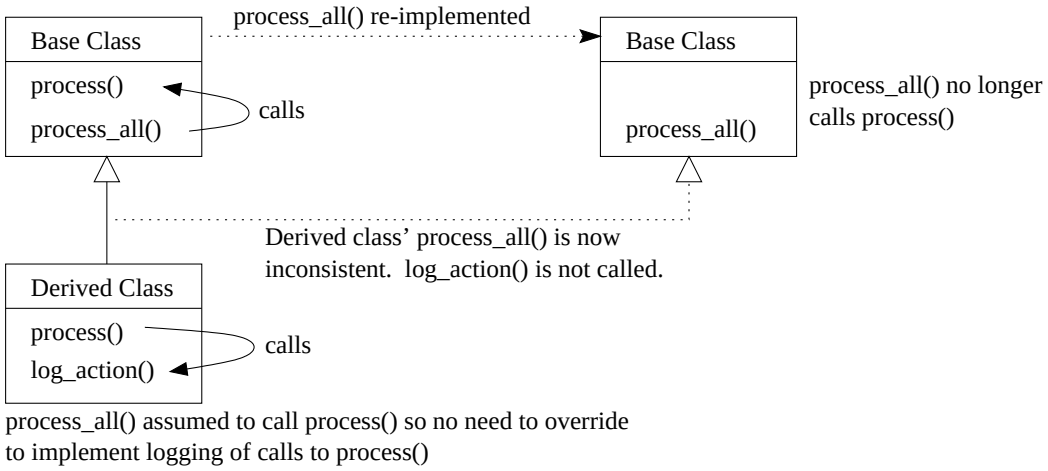


Figure 3: Inconsistency of class methods

3. Design patterns [29]
4. Framework overview
5. Interface contracts (specifications of obligations and collaborations). Also reuse contract [11]
6. Class descriptions (with class hierarchy)

Each of the above method has its strength and weaknesses. For example, using cookbook, recipes or example applications to document framework provides a quick learning path. However, it is impractical to document every possible alternative of extensibility. Design patterns and framework overview gives a good conceptual view of the framework, but it does not address hidden constraints within framework. Such constraints can be better described by the interface or reuse contract. When it comes to explaining implementation details, one may choose class description and its associated class hierarchy. In general, (3), and (4) are good ways to introduce the overall structure and concept of the framework while (1) and (2) provide a shortcut for learning how to build applications using the framework. Finally, (5) and (6) are important for understanding the internals of a framework, especially for white box reuse. Therefore, an effective documentation may require a combination of two or more of the above methods.

2.5 Integration/Inter-Operability and Standards

Most middle-ware integration frameworks aim at putting together software systems in a distributed environment. Some examples of such frameworks are object request broker, transactional database and messaging middleware [18]. From a certain point of view, middleware frameworks are trying to solve problems related to interoperability and incompatibility in heterogeneous environments. The

Object Management Group's (OMG) CORBA and CIM both addresses issues of integrating different application systems, and allowing these systems to interoperate among one another. Another important aspect of their effort is the standardization of the framework architecture. In view of the fact that framework is inherently costly to construct, the ability to integrate different frameworks together in a single application system will make it more appealing to invest in framework development. Therefore, standards must exist to ensure that different vendors will produce frameworks that can be integrated together in various ways. For instance, many frameworks have their own event loop, thus making it difficult to combine two frameworks into one application. Work remains to be done in this respect and standardized integration framework may play an important role here in the future.

2.6 Framework Maturity

Frameworks evolve with use and revisions. The level of maturity of a framework depends on the number of times it has been refined. Design metrics were presented in [6] to measure the architectural stability and maturity of a framework. We can therefore attempt to classify frameworks according to their maturity level, and also perform an estimate on the number of revisions it will take to acquire the stable mature state based on empirical data. This type of information will be useful for the management and assessment of framework development and usage [18, 6]

3 Techniques for Framework Constructions

3.1 Approaches to Developing Frameworks

If software development of a large complex system is difficult, then framework conception and development for systems within a business domain will be a more difficult task. Various methods have been proposed to reduce the difficulty of such a task. A bottom-up approach was suggested in [45]. The author starts by looking at the class structures of a specific application for the application domain concerned. This class structure is used to derive points of variability or *hot spots*. Finally, a generalization transformation is performed to convert the specialized class structures into a *hot spot* subsystem. In contrast, A top-down approach was presented in [3] and [7]. They first identify the domain models through strong knowledge of the domain. These models are then layered [7] to reduce complexity, and also to provide flexibility for application developers. However, frameworks presented by [42, 11] suggest that the construction of frameworks consist of a combination of top down and bottom up approach. A wide domain specific knowledge is necessary to evolve a set of specialized applications in the domain into a framework. To enable easy adaptation of frameworks, a set of design guidelines for the construction of *open systems* frameworks can be found in [14]. These guidelines build upon an existing well designed framework that has clearly defined *hot spots* [45] and *framework contracts* [11]. They summarizes to:

1. Identify axes of variability
2. Satisfy the interoperability requirement

3. Satisfy the distribution requirement
4. Satisfy the extensibility requirement

These approaches are still too general for a methodological approach to framework design. Most frameworks are still constructed in an *ad hoc* manner by experienced developers. There is yet an approach for the development of frameworks from grass roots specifications. Design patterns can give an important leverage to the development of frameworks and [45, 7, 48] have presented various concrete examples of their use. In the field of agent technologies, the notion of using agents as the building components of frameworks have been studied by [9, 10]. This might suggest an alternative approach to framework construction.

3.2 Validation and Embedded Testing

Validation of framework design is no doubt an important issue and the ability to embed testing into a framework serves as the first step into ensuring developing testable code. In [26], it is mentioned that embedding test methods into the framework allows them to be inherited into the application objects. Thus testing is made reusable and an integral part of the framework. Beyond the validation of a framework, applications extended from it should also be validated. Application extensions of a framework should conform to the intentions of the framework designer. In [38], the author suggested using the notion of *increment conformance* to verify applications developed from framework. The idea is to ensure that every increment made in the framework for the application satisfies a set of *composability constraints*. There are many ways to specify constraints. Descriptions for some ways to specify contraints, as cited in [38], can be found in [37, 23, 36, 2, 35, 50, 34]

4 Distributed & Parallel Computing Frameworks

This section will give a brief coverage of several frameworks dedicated to distributed and parallel computing. The first subsection discusses various middleware integration frameworks and the next will give an overview of several parallel computing system infrastructure framework. The interest for these system infrastructure frameworks is the ability to develop parallel applications with relative ease.

4.1 Middle-ware integration frameworks

4.1.1 Common Object Request Broker Architecture (CORBA)

CORBA is an architectural standard maintained by the Object Management Group (OMG). CORBA aims to provide a framework for deploying distributed object oriented applications [21, 49]. It encapsulates details of communication protocols, networking transports and implementation platforms from the application developer. The OMG Interface Description Language (IDL) provides an encapsulation layer over object implementation details. IDL describes object interface independent

of implementation details like programming language, machine architecture and operating environment. CORBA is considered as a white box framework, though most most implementations supports programming using a compositional style. However, the underlying architecture relies on inheritance as its core reuse mechanism.

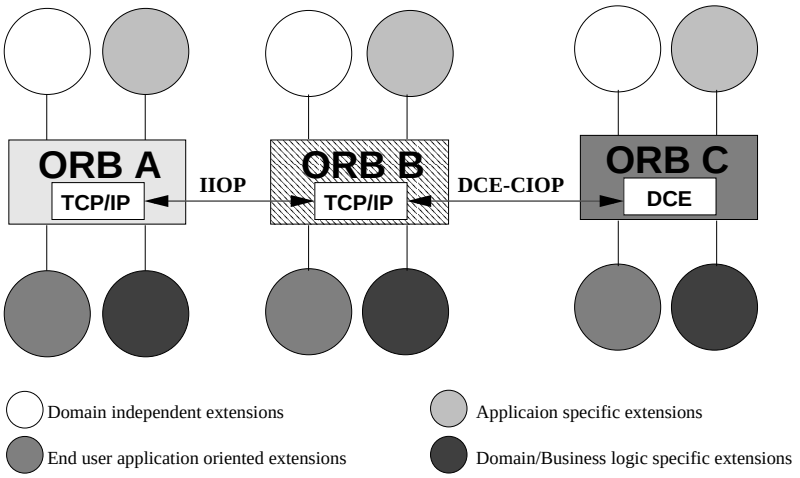


Figure 4: The Common Object Request Broker Architecture

The main component of CORBA is the object request broker (ORB). It serves as a kind of "backbone" for client and server object interactions (see Figure 4). Extensions, whether domain specific or non-specific, can be added on top of the ORB. Therefore, the ORB offers a potential for reusing many existing software. At present, the Object Management Architecture (OMA) reference model defines four major interface categories - object services, common facilities, domain interfaces and application interfaces [49]. Each of these four interfaces varies in their domain specificness. Common services, for example, is domain independent. Two widely used services are the naming service and the trading service. The naming allows clients to search for named objects while the trading service allows clients to find objects based on their properties. Domain interface addresses a more specific scope. Although it is still domain independent, domain interface focus on providing common facilities for end-user applications. For instance, OpenDoc's Distributed Document Compound Facility (DDCF) [4] is targeted at end-user document manipulation applications. The remaining two interfaces from those listed above address domain and application specific services respectively.

OMG addressed interoperability issues through the definition of various inter-ORB Protocols (IOP). So far, there are two main categories of IOP - General IOP (GIOP) and Environment Specific IOP (ESIOP). GIOP defines a communication protocol for any connection-oriented transport whereas ESIOP addresses systems that already have a distributed computing infrastructure installed [49]. For example, the Internet IOP (IIOP) is an implementation of GIOP for TCP/IP transport, and DCE Common IOP (DCE-CIOP) implements an ESIOP for systems that have DCE installed. Furthermore, there is also a standard object reference format, called the Interoperable Object Refe-

rence (IOR) that allows any CORBA compliant system to locate and communicate with the object concerned.

4.1.2 Distributed Computing Environment (DCE)

DCE was first introduced in the early 1990s by the Open Software Foundation [28]. It is not an object oriented framework. One can regard it as an architectural description of the components that addresses issues of distributed computing. The core structure of DCE consists of the following core components:

- Security
- Time & Directory Services
- Distributed File System
- Remote Procedure Call (RPC)
- Management
- Threads

The operating systems and network transport services provide the base foundation for this architecture, while the components above forms a layer of abstraction for development of applications. Although it is not an object oriented specification, DCE provides a complete outline of the concerns that must be addressed for distributed computing. It can be used as a basis for developing future distributed computing framework. There exists an object oriented implementation of DCE using C++, called OODCE [16]

4.1.3 Distributed Component Object Model (DCOM)

DCOM is a proprietary specification developed by Microsoft and is an extension of their Object Embedding & Linking(OLE)/Component Object Model(COM) specifications [13, 12]. OLE/COM specifies a binary standard that allows applications to share data. It is centered around compound documents. Compound documents are entities that are composed of more than one type of data, usually a combination of text, graphics and analysis data. DCOM extends this interface for distributed computing via an RPC mechanism. A large portion of its distributed foundation is built from DCE's architecture, particularly its RPC, directory and security services. In a broad sense, we can view DCOM as an object oriented variant of DCE. Contrary to the white box reuse approach of CORBA, COM emphasizes on a black box reuse. Applications are build using a compositional styles on available COM components, and the components are use by invoking on their respective interfaces.

4.2 Advance Frameworks for Distributed Computing

A number of distributed computing frameworks has been developed by research institutes and the academia. In the field of networking frameworks, ACE [46] (ADAPTIVE Communication Environment) and BAST [19] (named after an Egyptian Goddess) are two notable frameworks.

4.2.1 ACE

ACE is a layered framework that encapsulates the networking services provided by different operating systems [46]. It is a framework rich in design patterns for distributed computing. The main feature of this framework is its ease of use, portability and flexibility and runtime reconfigurability. Its ease of use and portability of application code can be attributed to the layered architecture that hides the complexities of low level details of networking services. The layering also provides an Adaptive Service Executive (ASX) that allows applications to flexibly reconfigure communication services at runtime. Its dynamic reconfiguration capability allows software to use an alternative or updated component without the need to shutdown. Fundamentally, the ASX framework contains concepts from modular communication frameworks like STREAMS [44], *x*-kernel [25] and the Conduit framework [51]. All these frameworks features flexible reconfigurability by inter-connecting *building-blocks*. It is also possible to interface ACE applications with CORBA via a CORBA handler class.

4.2.2 BAST

BAST is another framework for distributed computing [19]. The emphasis of BAST is on reliability. Like ACE, it is layered. There are six layers in BAST

- Application
- Atomic Group/Commit
- Distributed Agreement
- Failure Detection
- Communication
- Transport

The core of BAST lies in the middle four layers. These layers consists of a collection of protocol classes. These classes are grouped together to implement various communication protocols. BAST uses the *Strategy* pattern [20] to support flexible protocol composition and the Dynamic Terminating Multi-cast (DTM) pattern [19] to extend BAST's protocol classes. Application developers using BAST will compose protocol classes implemented in BAST to obtained the desired features.

Protocol developers who wish to extend BAST for new protocols, or simply to extend existing protocols within BAST will use the DTM pattern. It is a framework with a simple architecture, yet it allows the construction of fault tolerant software. Like ACE, BAST also derives concepts from STREAMS [44], *x*-kernel [40], and Conduit+[51]. Conduit+, in particular, is very similar to BAST,

but it decomposes the problem to a finer grain than BAST [19], and Conduit+ is largely a black box framework, while BAST is a gray box framework.

4.3 Higher Levels of Abstraction

Both ACE and BAST present a rather low level abstraction for the construction of distributed software systems. In the development of distributed collaborative applications, we need a higher level of encapsulation of the underlying details of distribution. Distributed collaborative applications are applications that allows multiple users who are geographically separated to simultaneously work on a common shared data. This data may be a text document, business proposal, engineering designs or simply raw data collected from remote instruments. Although developers of distributed collaborative applications must have fundamental knowledge of issues in constructing distributed software, their concern is not to directly deal with them. Frameworks like Rusken's contract based framework [32], Corona [22], COAST [47] and multi-user Suite [15] present to developers a higher level mapping of distributed software concern.

Rusken, for example, is a virtual space browser built using a contract based framework. The concept applies the notion of design by contract [36] in the framework. It uses *Contract* classes as an encapsulation of quality of service among networked components. Distributed applications within the collaborative group send messages to one another in the context of one or more contracts. Within each of the contract classes are mechanisms for monitoring timeliness of response, fault detection, data consistency, distributed consensus, etc. Any errors or non-conformance raises a contract violation, which will be dealt with accordingly. For instance, an application may choose an alternative service provider if the primary service provider violated it's contract of timeliness. Therefore, developers using this framework need only deal with high level constructs called *contracts*.

Corona applies the publisher-subscribe paradigm [33] and the peer-group paradigm to address issues of user size scalability, reliability of communications and coupling behavior among collaborative users. Here, we can view data sources as potential publishers and users who wish to access these data sources as subscribers. At the subscriber level, individual subscribers are not aware of one another, thus the coupling is very weak. In addition, Corona may further partition subscribers into groups whereby messages are relayed by local distributors. The principal publisher will distribute messages to its distributors, which will in turn distribute it to the subscribers in its group. This strategy allows very good scalability of user size, while compromising on reliability and coupling or awareness. For applications that require a tighter coupling and reliability in their communications, Corona provides the peer-group service that allows users to collaborate more effectively. Peer-grouping also provides more extensive failure detection since group members have a stronger awareness of one another.

One other aspect of collaborative distributed applications is its ability to dynamically vary the collaborative coupling behavior. Both Corona and the contract based framework employs group-wise coupling mechanism. In the latter, grouping is done via group contracts, which determine intra-group interaction behavior [32]. COAST (COoperative Application Systems Toolkit), however, employs *deputy object modeling* [39] to dynamically adjust coupling behaviour. This method employs a form of chain of responsibility. Requests are first sent to deputy objects. If they don't handle it, the request is forwarded to their *boss* objects. The role of the deputy object is to handle request that has a local

effect, whereas the boss objects process requests that affect a coupled group. Thus, by generating a network of deputy and boss objects, users can dynamically configure the coupling behavior of their collaborative interactions with respect to different shared data.

Finally, computer collaborative work oftens involves a user interface element. Compatibility of multiple user interface among different users can present a problem. COAST imposes a constraint on applications to inherit its document, view and controller classes, thus encapsulating complex display tasks from the application developer. Multi-user Suite, on the other hand, employs a user interface management systems [41] consisting of dialogue managers. The role of these dialogue managers is to handle interactive user sessions with the shared application object. Therefore, details of user interface are hidden from applications built from the framework.

4.4 Parallel Computing Frameworks

Besides middle-ware integration frameworks and frameworks for developing communications software, another major area is the development of parallel frameworks for scientific applications and frameworks for validating distributed object oriented software. Object oriented frameworks have also proved to be very successful in encapsulating parallelism so that scientists and engineers developing programs that require high processor bandwidths need not worry about the details of parallelism and optimization in their code. For example, frameworks like the Eiffel Parallel Execution Environment (EPEE) [27] and POOMA [43] have enabled developers to write parallel computing code with the ease of writing code for a single processor machine. All the internal optimization and distribution are handled internally by the framework. This approach removes the need for developing a compiler that targets a particular parallel machine, thus making the code portable across different multiprocessor platforms. In [24], it has been statistically shown that frameworks can achieve better scalability and performance gains over conventional parallel compilers.

4.5 San Francisco : A Distributed Enterprise Framework

The *San Francisco* Framework [8] is developed by IBM in response to the need to provide a flexible and easy to use framework for developing distributed business applications. It is developed in Java and has a layered architecture consisting of:

1. Core Business Process
2. Common Business Objects
3. Foundation
4. Platform Dependencies

The topmost layer (1) provides a set of commonly used business logic that developers can customize to build an application. The second layer presents a set of basic objects that allows developers to implement custom business logic in their applications using these objects. Finally, for domains not presently addressed by (1) and (2), developers can extend the framework from the foundation

layer. This layer encapsulates system level dependencies to provide an infrastructure for developing distributed applications. Developers who wish to extend lower level capabilities such as fault tolerance can exploit the kernel services within an object model interface at the foundation layer.

Development on *San Francisco* is ongoing, and future enhancements may include interoperability with OMG's CORBA via OMG's IIOP and the release of new capabilities for layers (1) and (2).

5 Trends in Object Oriented Frameworks

Fayad and Schmidt [18] suggested a number of areas where major work on frameworks should be done. They are:

- Reducing Framework Development Effort
- Domain Specific Enterprise Framework
- Tendency towards black box framework
- Documentation of frameworks
- Processes for managing framework development
- Economics of using/developing frameworks
- Standards

A large majority of the work within the frameworks research community has been to deal with making the construction of frameworks easier. The previous section demonstrated just some of the published work attempts to provide guidelines to implement frameworks successfully.

Due to the limited experience of building domain specific frameworks, there are very few documented instances of such entities [18]. However, the continuing work on the first point noted above, and also the growing experience of frameworks and design patterns will allow more of these domain specific frameworks to be realized. For example, [7] documented an approach to building a large banking systems framework. The ideas presented there can be applied to many other business systems besides banking. There are also authors [3, 42] who have demonstrated ways of leveraging the knowledge of domain experts and framework designers. [3] suggested building a framework from domain knowledge. This provides a mechanism whereby a domain expert can impart his/her knowledge in the initial design process to reduce the refinement cycles needed later on during framework design. On the other hand, [42] suggest designing a framework that allows domain experts adapt part of the framework that deals with their respective domain knowledge.

With the increase in acquiring domain knowledge, it will become easier to develop black box interfaces. This will effectively encourage black box reuse because it has an inherently dynamic object composition relationship [18], a contrast to the static inheritance relationships of white box reuse.

The documentation techniques discussed in an earlier section indicates that current documentation methods on frameworks are inadequate. This will restrict the reuse of large scale frameworks [18]. Therefore, an effective means of documenting frameworks is needed to ensure a gentler learning curve, and encourage greater reuse.

As more an more development efforts are centered around frameworks, there will be a need to manage such development efforts. Classification such as framework maturity and design metrics [6] will become a useful means to support the management of such processes. There will also be a greater focus on evaluating the cost of building or using frameworks [18], thus linking development efforts to the business bottom line.

We will also need to develop standards for integration and inter-operability among different frameworks. Standards can help us leverage the combined efforts of independently developed frameworks in the software community. Application developers will have a wider choice of frameworks to choose from.

Therefore, the software development process must take frameworks into consideration. Developers may not need to build an entire application from scratch. Instead, they can take an existing framework and customize its hot spots to produce a fully functional application. UML (Unified Modeling Language) is an attractive modeling language for application development from frameworks because it contains notation for illustrating both fine and coarse grain relationships. For example, developers can use class diagrams to model static relationships between application and framework classes. Dynamic relationships can be modeled using state charts or sequence charts. If a top level view of the system is desired, UML's package and collaboration diagrams will highlight important interdependencies between application and framework components. Furthermore, the framework itself can be modeled using UML, providing continuity within the entire software development process. The use of a common notation for software models greatly improves communication among developers, regardless of their implementation platform and language. Thus, we are able to better reuse tested and proven solutions to build complex systems

6 Promising Research Directions

We intend to put our research effort in the formulation of framework modeling and construction for distributed applications. During the course of our research, we will study issues concerning system integrity and validity of distributed application frameworks. We believe that this is a major concern in the development of frameworks, and also applications that derive from it. As mentioned in [11], inconsistency in the reuse of framework will compromise the quality of applications developed from it. Therefore, the ability to validate a framework, and applications derived from it is an important issue. This issue is particularly important for distributed systems due to their asynchronous nature.

Areas of research that may provide insights are concepts of meta-frameworks, UML modeling and architectural patterns of existing distributed systems. Issues on reliability and concurrency will be studied, with the aim of providing useful abstractions for developing frameworks for distributed applications. Current studies made on frameworks like ACE, BAST and frameworks for collaborative distributed applications have provided a starting point for our research. They have provided valuable concepts that are important in distributed software systems. BAST and Rusken, for example, illus-

trates a common approach at solving distributed reliability problems. Although both frameworks address different abstraction levels, the logical design of using object oriented classes to encapsulate individual distribution problems provides an extensible, yet simple architectural framework for instantiating applications.

Despite the increasing number of successful frameworks that have been implemented, there is still a large amount of work to be done. Some of the most significant areas include techniques for reducing framework development complexity, framework documentation and frameworks interoperability and standards. More and more software systems will be built from frameworks, especially in the area of distributed object oriented computing. Existing software development process models and the CASE tools that support them will have to be revised to accomodate the use of frameworks. We conclude to further our research in the areas of application frameworks for distributed systems, with emphasis on framework system integrity and the ability to validate framework application instances.

References

[1] R. Marty A. Weinand, E. Gamma. Et++ - an object oriented application framework in c++. In *OOPSLA'88*, pages 46–57. San Diego, 1988.

[2] M. Aksit and L. Bergmans. Obstacles in object-oriented software development. In *Proceedings of OOPSLA'92*, volume 27 of *ACM SIGPLAN Notices*, pages 341–358. ACM, Oct 1992.

[3] M. Aksit, F. Marcelloni, and B. Tekinerdogan. Developing object-oriented frameworks using domain models. email: aksit, bedir@cs.twente.nl and france@iet.unipi.it, 1997.

[4] Apple Computer, Inc. and Component Integration Laboratories, Inc. and International Business Machines Corporation and Novell, Incorporated. *Compound Presentation and Compound Interchange Facilities, Part I*, omg document 95-12-30 edition, Dec 95.

[5] Brad Appleton. Patterns and software: Essential concepts and terminology. *Object Magazine Online*, May 1997.

[6] J. Bansiya. Assessment of application framework maturity using design metrics. Computer Science Department, University of Alabama, Huntsville. email: jbansiya@cs.uah.edu, tel: 205-890-7317, 1997.

[7] D. Baumer, G. Gryczan, R. Knoll, C. Lilienthal, D. Riehle, and H. Zulighoven. Framework development for large systems. *Communications of the ACM*, 40(10):52–59, oct 1997.

[8] K. Bohrer, V. Johnson, A. Nilsson, and B. Rubin. Business process components for distributed object applications. *Communications of the ACM*, 41(6):43–48, june 1998.

[9] D. Brugali and K. Sycara. Agent technology: A new frontier for the development of application frameworks? In M.E. Fayad, D.C. Schmidt, and R.E. Johnson, editors, *Object Oriented Application Frameworks*. Wiley, 1998.

[10] D. Brugali and K. Sycara. Towards agent oriented application frameworks. *ACM Computing Surveys*, 1998. to appear.

[11] W. Codenie, K. Hondt, P. Steyaert, and A. Vercammen. From custom applications to domain specific frameworks. *Communications of the ACM*, 1997.

[12] Microsoft Corporations. The component object model: Technical overview.

[13] Microsoft Corporations. Dcom technical overview.

[14] S. Demeyer, T.D. Meijler, O. Nierstraasz, and Steyaert P. Design guidelines for tailorable frameworks. *Communications of the ACM*, 40(10):60–64, oct 1997.

[15] P. Dewan and R. Choudhary. A high-level and flexible framework for implementing multiuser user interface. *ACM Transactions on Information Systems*, 10(4):345–380, 1992.

[16] J. Diley. Oodce: A c++ framework for the osf distributed computing environment. In *Proceedings of the Winter USENIX Conference*. USENIX Association, Jan 1995.

[17] D. Doscher and R. Hodges. Sematech's experiences with the cim framework. *Communications of the ACM*, 40(10):82–84, oct 1997.

[18] Mohamed Fayad and Douglas C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, October 1997.

[19] B. Gabinato and R. Guerraoui. BAST: A framework for reliable distributed computing. Technical report, Swiss Federal Institute of Technology, Operating Systems Laboratory, Computer Science Department, june 1997.

[20] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[21] Object Management Group. Corba 2.0 specifications.

[22] R. W. Hall, A. Mathur, Jahanian F., Prakash A., and C. Rassmussen. Corona: A communication service for scalable, reliable group collaboration system. In *Computer Supported Cooperative Work*, pages 140–149. ACM, ACM Press, 1996.

[23] R. Helm, I.M. Holland, and D. Gangopadhyay. Contracts: Specifying behavioural compositions in object-oriented systems. In *Proceedings of ECOOP/OOPSLA'90*, pages 169–180, 1990.

[24] W. Humphrey, G. Mark, T. Cleland, J. Cummings, J. Qiang, S. Habib, and R. Ryne. Particle beam dynamics simulations using the pooma framework. In *Proceedings from ISCOPE98*. ISCOPE, 1998.

[25] N.C. Hutchinson and L.L. Peterson. The *x*-kernel: An architecture for implementing network protocols. *IEEE Transaction on Software Engineering*, 17:64–76, jan 1991.

[26] Jean-Marc Jézéquel. An object-oriented framework for data parallelism. *ACM Computing Surveys*, To be published, December 1998.

[27] Jean-Marc Jézéquel and Jean-Lin Pacherie. *Object-Oriented Application Frameworks*, chapter EPEE: A Framework for Supercomputing. John Wiley & Sons, New York, 1998.

[28] B.C. Johnson. Distributed computing environment framework technical paper, June 1991.

[29] R. Johnson. Documenting frameworks using patterns. In *Proceedings of OOPSLA 92 Vancouver, B.C. Canada*, pages 63–76, oct 1992.

[30] G. Kiczales and J. Lamping. Issues in the design and specifications of class libraries. In *Proceedings of OOPSLA '92*, pages 435–451. ACM/SIGPLAN, Oct 1992.

[31] Mark A. Linton, Paul R. Calder, and John M. Vlissides. InterViews: A C++ graphical interface toolkit. Technical Report CSL-TR-88-358, Stanford University, Computer Systems Lab, jul 88.

[32] Stephane Lorcy, Noel Plouzeau, and Jean-Marc Jézéquel. Reifying quality of service contracts for distributed software. In *26th Conference on Technology of Object-Oriented Systems (TOOLS USA'98)*, August 1998.

[33] A. Mathur, R. Hall, F. Jahanian, A. Prakash, and C. Rasmussen. The publish/subscribe paradigm for scalable group collaboration systems. Technical Report CSE-TR-270-95, University of Michigan, Ann Arbor, Michigan, November 1995.

[34] S. Matsuoka, K. Wakita, and A. Yonezawa. Synchronisation constraints with inheritance: What is not possible - so what is? Technical report, Tokyo University, 1992.

[35] J.D. McGregor and D.M. Dyer. A not on inheritance and state machines. *Software Engineering Notes*, 18(4):61–69, Oct 1993.

[36] B. Meyer. Applying "design by contract". *IEEE Computer (Special Issue on Inheritance & Classification)*, 25(10):40–52, October 1992.

[37] P. Molin. Designing reliable systems from reliable components using the context-dependent constraint concept. In *Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE'96)*, pages 142–151, Nov 1996.

[38] P. Molin. Verifying framework-based applications by establishing conformance. Technical Report 18/96, University College of Karlskrona/Ronneby, Dept. of Computer Science and Business Administration, Soft Center, S-372 25 Ronneby, Sweden, 1996.

[39] Z. Peng and Y. Kambayashi. Deputy mechanisms for object oriented databases. In *Proceedings of the IEEE 11th Conference on Data Engineering*, March 1995.

[40] L. Peterson, N. Hutchinson, S. O'Malley, and M. Abott. Rpc in the *x*-kernel: Evaluating new design techniques. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 91–101. ACM, nov 1989.

[41] G. Pfaff. *User Interface Management Systems*. Springer-Verlag, 1985.

[42] E.J. Posnak, R.G. Lavender, and H.M. Vin. An adaptive framework for developing multimedia software components. *Communications of the ACM*, 40(10):43–47, oct 1997.

[43] J. et. al. Reynders. Pooma: A framework for scientific simulations on parallel architectures. In G.V. Wilson and P. Lu, editors, *Parallel Programming Using C++*, pages 553–594. MIT Press, 1996.

[44] D. Ritchie. A stream input-output system. *AT&T Bell Labs Technical Journal*, 63:311–324, oct 1984.

[45] H.A. Schmid. Systematic framework design by generalization. *Communications of the ACM*, 40(10):48–51, oct 1997.

[46] D.C. Schmidt. The adaptive communication environment - an object oriented network programming toolkit for developing communication software. In *Sun User Group Conferences*, June, Dec 1993.

[47] C. Schuckmann, L. Kirchner, J Schümmer, and J.M. Haake. Designing object-oriented synchronous groupware with coast. In *Computer Supported Cooperative Work '96*, pages 30–38. ACM, ACM Press, 1996.

[48] A. R. Silva. Framework, design patterns and pattern language for object concurrency. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, july 1997.

[49] S. Vinoski. Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 1997.

[50] D.A. Wilson and S.D. Wilson. Writing frameworks - capturing your expertise about a problem domain. In *Tutorial Notes, OOPSLA'93*, 1993.

[51] J.M. Zweig. The conduit: A communication abstraction in c++. In *Proceedings of the 2nd USENIX C++ Conference*, pages 191–203. USENIX Association, apr 1990.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifi que,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399