INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Triskell

Reliable and efficient component based sofware engineering

Rennes

THEME 1C

Table of contents

1. Team

Scientific head
Jean-Marc Jézéquel [professor, Rennes 1 University]

Administrative assistant
Myriam David [TR Inria]
Nadège Filland [Until 9/9/2003]

Inria staff
Didier Vojtisek [Research engineer Inria]
Loïc Helouët [Research scientist Inria]

CNRS staff
Claude Jard [DR, Research scientist CNRS, Professor, ENS Cachan/Bretagne]

Faculty member Université de Rennes 1
Yves Le Traon [Assistant Professor Université de Rennes 1]
Noël Plouzeau [Assistant Professor Université de Rennes 1]

Visiting scientist
Bernhard Rumpe [Invited Professor, Tech. Univ. Munich, April to June]
Alexandre Petrenko [Senior research scientist, CRIM, Montreal]

Post-doc
Olivier Defour [from September 1st, 2003 to May 31th, 2004]

Technical staff
Frederic Fondement [Inria (project CAFE) until October 1st 2003]
Jean-Philippe Thibault [Inria (project RNTL ACCORD) since october 1st 2002 until October 1st 2003, (project FAMILIES) from October 1st 2003 to June 30th, 2005]
Julien Thomas [Inria (Project MAGDA2), from march 15th 2002 to January 15th, 2004]
Erwan Drezen [Inria (Project Carroll/Motor Carroll/Mutation from June 2003 to June 2004]

PhD Students
Benoît Baudry [MENRT grant]
Éric Cariou [MENRT grant]
Thomas Chatain [ENS Cachan, from September 1st 2003]
Emmanuel Donin de Rosière [CIFRE grant, from October 1st 2003]
Franck Fleurey [MENRT grant]
Jacques Klein [INRIA grant]
Karine Macedo [INRIA grant]
Christophe Métayer [CIFRE grant]
Clémentine Nébut [INRIA-région grant]
Simon Pickin [University of Madrid]
Damien Pollet [MENRT grant]
Tewfik Ziadi [INRIA grant]

2. Overall Objectives

Key words: *Components*, *objects*, *contracts*, *design patterns*, *aspects*, *frameworks*, *UML*, *software product lines*, *MDA*, *test*, *validation*, *requirements engineering*, *scenarios*.

2.1.1. Research fields

In its broad acceptation, Software Engineering consists in proposing practical solutions, founded on scientific knowledge, in order to produce and maintain software with constraints on costs, quality and deadlines. In this

field, it is admitted that the complexity of a software increases exponentially with its size. However on the one hand, the size itself of the software is on average multiplied by ten every ten years, and on the other hand, the economic pressure resulted reducing the durations of development, and in increasing the rates of modifications made to the software.

To face these problems, today's mainstream approaches build on the concept of component based software. The assembly of these components makes it possible to build families of products made of many common parts, while remaining opened to new evolutions. As component based systems grow more complex and mission-critical, there is an increased need to be able to represent and reason on such assemblies of components. This is usually done by building models representing various aspects of such a product line, such as for example the functional variations, the structural aspects (object paradigm), of the dynamic aspects (languages of scenarios), without neglecting of course nonfunctional aspects like quality of service (performance, reliability, etc.) described in the form of contracts, or the characteristics of deployment, which become even dominating in the field of reactive systems, which are often distributed and real-time. Model Driven Engineering (MDE) is then a sub-domain of software engineering focusing on reinforcing design, validation and test methodologies based on multi-dimensional models.

2.1.2. Project-team Presentation Overview

The research domain of the Triskell project is the reliable and efficient design of software product lines by assembling software components described with the UML. Triskell is particularly interested in reactive and distributed systems with quality of service constraints.

Triskell's main objective is to develop model-based methods and tools to help the software designer to obtain a certain degree of confidence in the reliability of component assemblies that may include third-party components. This involves, in particular, investigating modeling languages allowing specification of both functional and non-functional aspects and which are to be deployed on distributed systems. It also involves building a continuum of tools which make use of these specification elements, from off-line verifiers, to test environments and on-line monitors supervising the behavior of the components in a distributed application.

Another goal of the Triskell project is to explicitly connect research results to industrial problems through technology transfer actions. This implies, in particular, taking into account the industrial standards of the field, namely UML, Corba Component Model (CCM), Com+/.Net and Enterprise JavaBeans.

Triskell is at the frontier of two fields of software: the field of specification and formal proof of software, and that of design which, though informal, is organized around best practices (*e.g.; design patterns or the use of off-the-shelf components*). We believe that the use of the techniques that we develop will make it possible to improve the transition between these two worlds, and will contribute to the fluidity of the processes of design, implementation and testing of software.

3. Scientific Foundations

3.1. Overview

The Triskell project studies new techniques for the reliable construction of software product lines, especially for distributed and reactive software. The key problems are components modeling and the development of formal manipulation tools to refine the design, code generation and test activities. The validation techniques used are based on complex simulations of models building on the standards in the considered domain.

3.2. Object Oriented Technologies for Distributed Software Engineering

Key words: *Objects*, *design patterns*, *frameworks*, *software components*, *contracts*, *UML.*

3.2.1. Object-Oriented Software Engineering

The object-oriented approach is now widespread for the analysis, the design, and the implementation of software systems. Rooted in the idea of modeling (through its origin in Simula), object-oriented analysis,

design and implementation takes into account the incremental, iterative and evolutive nature of software development [52][43]: large software system are seldom developed from scratch, and maintenance activities represent a large share of the overall development effort.

In the object-oriented standard approach, objects are instances of classes. A class encapsulates a single abstraction in a modular way. A class is both *closed*, in the sense that it can be readily instanciated and used by clients objects, and *open*, that is subject to extensions through inheritance [57].

3.2.2. Design Pattern

Since by definition objects are simple to design and understand, complexity in an object-oriented system is well known to be in the *collaboration* between objects, and large systems cannot be understood at the level of classes and objects. Still these complex collaborations are made of recurring patterns, called design patterns. The idea of systematically identifying and documenting design patterns as autonomous entities was born in the late 80's. It was brought into the mainstream by such people as Beck, Ward, Coplien, Booch, Kerth, Johnson, etc. (known as the Hillside Group). However the main event in this emerging field was the publication, in 1995, of the book *Design Patterns: Elements of Reusable Object Oriented Software* by the so-called Gang of Four (GoF), that is Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides [46]. Today, design patterns are widely accepted as useful tools for guiding and documenting the design of object-oriented software systems. Design patterns play many roles in the development process. They provide a common vocabulary for design, they reduce system complexity by naming and defining abstractions, they constitute a base of experience for building reusable software, and they act as building blocks from which more complex designs can be built. Design patterns can be considered reusable micro-architectures that contribute to an overall system architecture. Ideally, they capture the intent behind a design by identifying the component objects, their collaborations, and the distribution of responsibilities. One of the challenges addressed in the Triskell project is to develop concepts and tools to allow their formal description and their automatic application.

3.2.3. Framework

Frameworks are also closely related to design patterns. An object-oriented software framework is made up 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 [39]. 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 [46]. A framework is thus different from a classical class library in that the flow of control is usually often bi-directional between the application and the framework. The framework is in charge of managing the bulk of the application, and the application programmer just provides various bits and pieces. This is similar to programming some event driven applications, when 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 may use several design patterns, some of them general purpose, some of them domain-specific. Design patterns and frameworks are thus closely related, but they do not operate at the same level of abstraction: 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.

3.2.4. Component

The object notion also provides the bases needed to develop the concept of *software component*, for which Szyperski's definition [62] is now generally accepted, at least in the industry:

> *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.*

Component based software relies on assemblies of components. Such assemblies rely in turn on fundamental mechanisms such as precise definitions of the mutual responsability of partner components, interaction means between components and their non-component environment and runtime support (e.g. DotNet, EJB, Corba Component Model).

Components help reducing costs by allowing reuse of application frameworks and components instead of redeveloping applications from scratch (product line approach). But more important, components offer the possibility to radically change the behaviors and services offered by an application by substitution or addition of new components, even a long time after deployment. This has a major impact of software lifecycle, which should now handle activities such as:

- design of component frameworks,
- design of reusable components as deployment units,
- validation of component compositions coming from various origins,
- component life-cycle management

It seems clear that empirical methods without real component composition models that have appeared during the emergence of a real component industry (at least in the Windows world) are now the cause of untractable validation and integration problems can not be transposed to more critical systems (see for example the accidental destruction of Ariane 501 [54]).

Providing solutions for formal component composition models and for verifiable quality (notion of *trusted components*) are especially relevant challenges. Also the methodological impact of component-based development (for example within the maturity model defined by the SEI (CMM model)) is also worth attention.

3.2.5. Contracts

Central to this trusted component notion is the idea of *contract*. A software contract captures mutual obligations and benefits among stake-holder components, for example between the client of a service and its suppliers (including subclasses). Contracts strengthen and deepen interface specifications. Along the lines of abstract data type theory, a common way of specifying software contracts is to use boolean assertions called pre- and post-conditions for each service offered, as well as class invariants for defining general consistency properties. Then the contract reads as follows: the client should only ask a supplier for a service in a state where the class invariant and the precondition of the service are respected. In return, the supplier promises that the work specified in the postcondition will be done, and the class invariant is still respected. In this way rights and obligations of both client and supplier are clearly delineated, along with their responsibilities. This idea was first implemented in the Eiffel language [58] under the name *Design by Contract*, and is now available with a range of expressive power into several other programming languages (such as Java) and even in the Unified Modeling Language (UML) with the Object Constraint Language (OCL) [63]. However, the classical predicate based contracts are not enough to describe the requirements of modern applications. Those applications are distributed, interactive and they rely on resources with random quality of service. We have shown that classical contracts can be extended to take care of synchronization and extrafunctional properties of services (such as throughput, delays, etc) [42].

3.2.6. UML

Some prototypes in laboratories provide answers to these problems, but the massive adoption of UML in many industrial domains open new perspectives to make the underlying ideas evolve, scale up, and hence

become profitable. Unlike its predecessors, (OMT, Booch, etc.), that only proposed a graphical syntax, UML is partially formalized by a meta-model (expressed itself as a UML model) and contains a very sophisticated constraint language called OCL (*Object Constraint Language*), that can be used indifferently at the model level or at the meta-model level. All this makes it possible to consider formal manipulations of models that capture many aspects of software, both from the technical side, (with the four UML main dimensions: data, functional, dynamic, and deployment) and on the process side, ranging from the expression of requirements and the analysis to design (framework models and design patterns) and test implementation.

3.3. Mathematical foundations for distributed and reactive software

Key words: *Labeled transition systems*, *partial orders*, *event structures*.

Labeled transitions systems are the mathematical structure that characterizes best the foundations of research on software models [40]. This structure was developed 50 years ago. However, models of real systems can be very large, and it is not always possible to build the complete model before performing a formal manipulation. In some cases, it is possible to apply lazy construction methods (also called on the fly). Concurrency is another fundamental aspect that must be considered by models. This is the central concept needed for the analysis of distributed systems [53].

3.3.1. Transition systems

A transition system (or LTS) is a directed graph which edges, called transitions, are labeled by letters from an alphabet of **events**. The vertices of this graph are called **states**. A LTS can be defines as a tuple $M = (Q^M, A, T^M \subset Q^M \times A \times Q^M, q^M_{init})$, in which Q^M is a set of states, q^M_{init} is an initial state, A is a set of events, T^M is a transition relation.

Note that from this definition, the set of states in a LTS s not necessarily finite. Usually, the term **finite state automata** is used to designate a LTS with a finite set of states and events. In fact, automatas are the simplest models than can be proposed. They are often used to model reactive (and usually distributed) systems. Within this framework, events represent the interactions (inputs and outputs) with the environment. The term input/output LTS (IO-LTS) is often used to designate this kind of automata.

Labeled transition systems are obtained from reactive systems specifications in high-level description languages such as UML. The construction of a LTS from a specification is done using an operational semantics for this language, which is usually formalized as a deduction rules system. For simple languages such as process algebras (like CCS), operational semantics can be defined using less than axioms and inference rules, while for notations such as UML, semantics would be defined in more than 100 pages.

For performance reasons, these operational semantics rules are never used directly, and are subject to several transformations. For example, the way states are encoded is an efficiency factor for LTS generation.

Computation of transformations of LTS can be resumed to search and fix-point calculus on graphs. These calculi can be performed either explicitely or implicitly, without an exhaustive calculus or storage of a LTS.

Classical algorithms in language theory build explicitely finite state automatas, that are usually integrally stored in memory. However, for most of the problems we are interested in, exhaustive construction or storage of an LTS is not mandatory. Partial construction of an LTS is enough, and strategies similar to lazy evaluation in functional programs can be used: the only part of LTS computed is the part needed for the algorithm.

Similarly, one can forget a part of a LTS previously computed, and hence recycle and save memory space. The combination of these implicit calculus strategies for LTS allow the study of real size systems even on reasonably powerful machines.

3.3.2. Non interleaved models

One of the well known drawbacks of LTSs [44] is that concurrency is represented by means of behaviors interleaving. This is why LTS, automatas and so on are called “interleaved models”. With interleaved models, a lot of memory is lost, and models represented can become very complex. Partial order models partially solve these problems.

A partial order is a tuple $(E, \leq, \Pi, \phi, \Sigma, I)$ in which:

- E represents a set of atomic events, that can be observable or not. Each event is the occurrence of an action or operation. It is usually considered that an event is executed by an unique process in an system.
- $\leq$ is a partial order relation that describes a precedence relation between events. This order relation can be obtained using the hypotheses that:

  i. processes are sequential : two events executed by the same process are causally ordered.

  ii. communications are asynchronous and point to point: the emission of a message precedes its reception.

- σ is an alphabet of actions.
- I is a set of process names
- $\Pi : \Sigma \rightarrow I$ is an action placement function.
- $\phi : E \rightarrow \Sigma$ is an event labeling function

A partial order can be used to represent a set of executions of a system in a more "compact" way than interleaved models. Another advantage of partial order models is to represent explicitely concurrency : two events that are not causally dependant can be executed concurrently. In a LTS, such a situation would have been represented by an interleaving.

A linearization of a partial order is a total order that respect the causal order. Any linearization of a partial order is a potential execution of the system represented. However, even if partial order can represent several executions, linearizations do not represent a real alternative. This problem is solved by a more complete partial order model called event structures.

A prime *event structure* [65] is a partial order equipped with an additional binary conflict relation. An event structure is usually defined by a tuple $(E, \leq, \sharp, \Pi, \phi, \Sigma, I)$ where:

- $E, \leq, \Pi, \phi, \Sigma, I$ have the same signification as previously,
- $\sharp \subseteq E \times E$ is a binary and symmetric relation that is inherited through causality ($\forall e \sharp e', e \leq e'' \Longrightarrow e'' \sharp e'$).

The conflict relation of an event structure defines pairs of events that can not appear in the same execution of the system represented, hence introducing alternative in partial orders. The potential executions of a system represented by an event structures are linearizations of conflict free orders contained in the structure. The main advantage of event structures is to represent at the same time concurrency and alternative in a partial order model. We think that these models are closer to human understanding of distributed systems executions than interleaved models.

4. Application Domains

4.1. Software for Telecommunication and large Distributed Systems

Key words: *telecommunication*, *distributed systems*, *software engineering*, *test*, *UML*.

The telecommunication world is evolving rapidly. New worldwide infrastructures connecting multiple components are settling. Mobile communications is also a growing sector and new services are constantly developed. Conversly many large systems, such as Air Traffic Control, have a distributed nature: they usually involve many processing nodes linked through communications links.

On the software point of view, building such a new application from scratch is no longer possible. There is a real need for flexible solutions allowing to deal at the same time with a wide range of needs (product lines

modeling and methodologies for managing them), while reducing the time to market (such as derivation and validation tools).

Triskell has gained experience in model engineering, and finds here a propitious domain. The increasing software complexity and the reliability and reusability requirements fully justify the methods developed by our project. The main themes studied are reliable software components composition, UML-based developments validation, and test generation/diagnosis from UML models.

The research activity in Triskell focuses at the same time on development efficiency and reliability. The main applications of this research mainly concern reliable construction of large scale communicating software, and object oriented systems testing.

Reliability is an essential requirement in a context where a huge number of softwares (and sometimes several versions of the same program) may coexist in a large system. On one hand, software should be able to evolve very fast, as new features or services are frequently added to existing ones, but on the other hand, the occurrence of a fault in a system can be very costly, and time consuming. A lot of attention should then be paid to interoperability, *i.e. the ability for software to work properly with other. We think that formal methods may help solving this kind of problems. Note that formal methods should be more and more integrated in an approach allowing system designer to build software globally, in order to take into account constraints and objectives coming from user requirements.*

Software testing is another aspect of reliable development. Testing activities mainly consist in ensuring that a system implementation conforms to its specifications. Whatever the efforts spent for development, this phase is of real importance to ensure that a system behaves properly in a complex environment. We also put a particular emphasis on on-line approaches, in which test and observation are dynamically computed during execution.

5. Software

5.1. umlaut ng : Extendible model transformation tool and framework.

Participants: Erwan Drézen, Frédéric Fondement, Jean-Marc Jézéquel, Bernhard Rumpe, Damien Pollet, Jean-Philippe Thibault, Didier Vojtisek [correspondant].

Key words: *MOF*, *UML*, *MDA*, *model transformation*, *component*, *patterns*, *validation*.

Model transformation is a key aspect of the MDA(Model driven Architecture). It allows the assistance and/or automation for model creation from early stages of the development to code generation. It has received a lot of attention in recent months through the RFP QVT (Request For Prososal: Query View Transformation).

UMLAUT has evolved into UMLAUT NG (Next Generation) in order to anticipate these new requirements. This new version now clearly separates the repository part from the transformation engine. It is no longer dependent on a specific metamodel and can manipulate any of them (including the various versions of UML). This allows a greater flexibility and reusability of the user's transformations.

MDA (*Model Driven Architecture*) is an approach to application modelling and generation that has received a lot of attention in recent months. This is a logical evolution of the UML (*Unified Modelling Language*) usage supporting the following ideas:

- Models expressed in a formally defined notation are a cornerstone to system understanding.
- Building systems can be organized around a set of models by imposing a series of transformations between models, organized into an architectural framework of layers and transformations.

For example this evolution allows to formalize and automate the use of PIM (*Platform Independent Model*) and PSM (*Platform Specific Model*). The resulting conception lifecycle creates abstract models (platform independent) which are successively refined into more concrete models (more an more platform dependent). It gives a way to work at the best abstraction level for a given problem.

One of the major point to be addressed is the model transformation aspect of the problem. With UMLAUT, Triskell already had a tool that allows us to address a big part of that. However, it was limitated to handle UML models. Thus, UMLAUT evolved into UMLAUT NG (*next generation*). This way Triskell stays a leader in the domain and reuses its expertise.

UMLAUT NG provides all the main features of UMLAUT plus the ability to manipulate any kind of models on any kind of repositories. A transformation can be run on any repository that havs compatible metamodels. The metamodels are defined using MOF. UMLAUT NG is now composed of a transformation language compiler and a framework of transformations written in this language It allows complex model transformations. A major idea that leads UMLAUT NG evolution is that a transformation is a kind of program so it must apply the MDA approach to itself.

As a central tool in the team, UMLAUT NG helps us investigating different research areas related to model transformation works. Since 1998, Triskell has mainly used it in the UML context to demonstrate several concepts. For example, to apply design patterns, to support the Design by Contract approach, to weave modelling aspects, to generate code, to simulate functional and extra functional features of a system, or use validation tools on the model. All these concepts will probably be investigated further. Recent works are: The definition of a tooling suite for expressing requirements in a MDA aware test environment, application of design by contract using aspect weaving, and Product Line class architecture derivation [32].

UMLAUT NG as its predecessor will be distributed as an open-source software. Running demonstrations are available on the following web pages: http://modelware.inria.fr/.

In 2003, UMLAUT NG was used within these projects in collaboration with industry:

RNTL ACCORD involving Softeam, FT R&D, EDF, CNAM, ENST, ENSTBr and LIFL about assembling components using contracts in a distributed environment;

CARROLL MOTOR with Thalès R&D and CEA, development of the new compiler (independent of repositories and metamodels);

CARROLL MUTATION with Thalès R&D, Thalès Airborne System and CEA, development of transformations useful for the test of a military application in a MDA context;

IST QCCS with Schlumberger, Sema, KD-soft, and TU Berlin about conception by contracts and aspect weaving in UML;

ITEA CAFE with (in France) Softeam, Thalès, about transformation of product lines;

ITEA FAMILIES with (in France) Softeam, Thalès, about transformation of product lines, as the continuation of CAFE;

5.2. sofat : A toolbox for Scenario analysis.

Participant: Loïc Hélouët.

Key words: *Scenarios*, *sequence diagrams*, *analysis*, *simulation*, *validation*.

Representing distributed systems behaviors by means of informal chronograms is an usual practice. This representation can be used for requirements capture phases, to represent traces of a system during debugging, or as test scenarios. Most of CASE tools propose a scenario notation : Message Sequence charts for SDL tools, Sequence diagrams for UML environments, etc.

Despite their simplicity, scenarios have a huge expressive power ass soon as basic chronograms can be composed. Hence, analysing such scenarios without an appropriate tool rapidly becomes impossible. SOFAT (Scenario Oracle and Formal Analysis Toolbox) is a toolbox allowing formal manipulation of scenarios.

Scenarios are often used to describe informally behaviors of distributed systems. They can be used for documentation, to capture requirements, to express test purposes, etc. The building blocks for scenarios are chronograms, that describe ordering beetween events occuring in a system's execution. Message Sequence Charts(MSC) and the future UML 2.0 sequence diagrams provide means to compose scenarios by means of choice, sequence, loop, and parallel composition operators. The resulting expressive power of such languages is very high, but as a consequence, many formal properties become undecidable. For example, model-checking Message Sequence Charts is not possible in general.

This kind of limitation could render scenarios useless. Fortunately, undecidable properties are usually linked to obfuscated use of scenarios, and several sub-classes of scenarios allowing intersting formal manipulations have been identified.

The sofat toolbox allows for a classification of MSC according to formal criterions. For example, SOFAT can detect if a MSC can be represented by an equivalent finite autoamton, if it can be simulated, if all choices contained are performed by a single process, and so on. When an automaton can be derived from a MSC, sofat realizes this transformation. Then, all formal manipulations allowed on automatas (model-checking, abstraction, composition, ...) become possible for MSCs. Some MSCs contain ambiguities that even prevents then from being simuated. SOFAT detects such ambiguities, and allows for an interactive simulation of scenarios when possible. This simulation is based on the event structure semantics provided in [50].

SOFAT has been deposed at the APP (Agence pour la Protection des Programmes) under number IDDN.FR.001.080027.000.S.P.2003.000.10600. It is availaible from the following webpage: http://www.irisa.fr/triskell/perso_pro/helouet/SOFAT

5.3. Mutator : Mutation testing tool family for OO programs

Participants: Yves Le Traon [correspondant], Benoit Baudry, Franck Fleurey.

Key words: *Test*, *test by mutation*, *Java*, *.Net*.

The level of confidence in a software component is often linked to the quality of its test cases. This quality can in turn be evaluated with mutation analysis: faulty components (mutants) are systematically generated to check the proportion of mutants detected ("killed") by the test cases. The work consisted in proposing specific OO mutation operators and the corresponding tools for Java and C# programs : the Mutator line of mutation tools is available for Java anc C# languages. This work has been carried out in collaboration with Daniel Deveaux from UBS.

5.4. Requested: a toolbox for requirement simulation and testing

Participants: Yves Le Traon [correspondant], Clémentine Nébut, Erwan Drézen, Franck Fleurey.

Key words: *Test*, *requirement simulation*, *requirement testing*, *textual requirements*, *use cases*.

The objective of the Requested toolbox is to offer a MDA transformation from textual requirements to simulable requirements within the UML (use cases + scenarios). It allows the simulation of requirements and the automated generation of test objectives. Two tools are under development:

1. the transformation of natural language requirements expressed in the LDE language (Langage de Description des Exigences) into a use case model, enhanced with contracts. This tool is not stable yet and thus not available. It is currently used and tested in the mutation project.
2. the UCTS system allows the simulation of the use case model, enhanced with contracts, and the automated generation of test objective. The first version is available.

More precisely, UCTSystem is a prototype tool designed to perform automatic test generation from UML requirements. It uses UML use cases enhenced with contracts (*i.e. precondition and postconditions) to build an execution model allowing all valid sequences of use cases. Using this execution model and several test criteria, it generates test objectives as sequence of use cases to exerce. It includes both criteria for functional testing and a criterion for robusness testing. Those test objectives are then mapped into test cases using test templates.*

6. New Results

6.1. Model-Driven Engineering with Contracts, Patterns, and Aspects

Participants: Jean-Marc Jézéquel, Loïc Helouët, Yves Le Traon, Noël Plouzeau, Éric Cariou, Jacques Klein, Karine Macedo, Christophe Métayer, Damien Pollet, Olivier Defour, Tewfik Ziadi.

It is now widely accepted that functional aspects (Platform Independent Models (PIM) in OMG's Model Driven Architecture terminology) should be dissociated from non-functional ones (such as persistency, fault-tolerance, transactions and so on) and their platform specific implementations. The Unified Modeling Language (UML) gives the designer a rich, but somehow disorganized, set of views on her model as well as many features to add non-functional annotations to a model. We have worked to how to organize models around the central notions of (1) quality of service contracts (for specifying non-functional properties a la QML on PIMs) and (2) aspects for describing how they can be implemented. Based on our experience in the IST QCCS project, we posed to model contracts in UML with a small set of stereotypes, and to represent aspects like design pattern occurrences, that is using parameterized collaborations equipped with transformation rules expressed with an object-oriented extension of OCL2 working at the meta-model level. This relies on our transformation based approach as implemented in the UMLAUT framework to build design level aspect weavers, based on a meta-level interpreter, that takes a PIM as entry point, processes the various aspect applications as specified by the designers, and outputs another (detailed design, or PSM level) UML model that can serve as a basis for code generation.

6.1.1. Contract Based Component Modeling

Participants: Jean-Marc Jézéquel, Noël Plouzeau, Éric Cariou, Karine Macedo, Olivier Defour.

In 2003 the Triskell has completed the definition of a UML based meta-architecture for extrafunctional property specification. Our work relies on the specification of quantitative dimensions, on the definition of constraint spaces (as an extension of QML [45]). Component-based software design is composed of several activities:

1. design of fundamental components that provide service abstractions, for services that are not component based (*e.g. the environment, the operating system);*
2. *design of communication components allowing to reify specific collaboration patterns among components [11];*
3. *design of "macro" components by assembling and encapsulating other existing components; these components expose the quantitative properties of the services they provide;*
4. *application design, including feedback of quantitative properties to the user; this includes also possibilities for the user to make choices on the desired quality of service (e.g. lower the throughput to get better image transmission).*

The Triskell team designed in 2001 an extension of the UML metamodel [64]. This extension allows the addition of extra-functional contracts to a component model that is also an extension of the UML 2 component model. Then this platform independent model is refined to get platform specific models that rely on the platform services and configuration options [36]. This platform dependent model is generated from the platform independent one using UML model transformation techniques and tools. During year 2003, this notation extension and its associated tools [35] have been tested on the user applications provided by the partners of the QCCS project (see section 7.3).

6.1.2. Design based on Model Transformation

Participants: Jean-Marc Jézéquel, Yves Le Traon, Christophe Métayer, Damien Pollet, Tewfik Ziadi.

In many large organizations, the model transformations allowing the engineers to more or less automatically go from platform-independent models (PIM) to platform-specific models (PSM) are increasingly seen as vital assets. As tools evolve, it is critical that these transformations are not prisoners of a given CASE tool.

Considering that a CASE tool can be seen as a platform for processing a model transformation, we have proposed to reflectively apply the MDA to itself [20][34]. We have thus proposes to describe models of transformations that are CASE tool independent (platform-independent transformations or PIT) and from them to derive platform-specific transformations (PST).

In the context of product lines, handling the various derivations of a product can be a daunting (and costly) task. To tackle this problem, we have proposed a method based on the use of a creational design patterns to uncouple the variations from the selection process. This makes it possible to automatically derive a given product from the set of all possible ones, and to specialize its design model accordingly [32]. We defined a set of patterns for modeling variability issues of a Product Line Architecture to define architectural constraints for Product Line expressed in UML as meta-level OCL constraints and to propose an approach to automate the derivation process [30][32][31].

6.1.3. Aspect Oriented Design

Participants: Jean-Marc Jézéquel, Loïc Helouët, Noël Plouzeau, Jacques Klein, Karine Macedo, Olivier Defour.

Nowadays the academic community and the software industry begin to recognize the value of aspect oriented programming [56]. Applying aspect orientation to the software design phase gives birth to *aspect oriented design*. During the year 2003 the Triskell team has enhanced the preliminary technique defined in 2002. This technique uses message sequence charts MSC as a notation to specify the behaviours of model fragments designed as aspects. It was mature enough to enable J.-M. Jézéquel to give tutorials on "Model-Driven Engineering with Contracts, Patterns, and Aspects" at AOSD 2003 (2nd International Conference on Aspect-Oriented Software Development) and ASE 2003 (Automated Software Engineering) [17].

6.2. Validation and Test

6.2.1. Model Based Testing: Product Line and Test Patterns

Participants: Jean-Marc Jézéquel, Yves Le Traon, Simon Pickin, Bernhard Rumpe, Clémentine Nébut, Erwan Drézen, Franck Fleurey.

To deal with functional testing of product lines (PL), we propose a method bridging the gap between PL requirements and functional test cases dedicated to each product [25]. The core of the method is to let the tester concentrate on building particular test objectives, named "Behavioral Test Patterns", which consist of a selection from the set of simple scenarios attached to the use cases [27]. Then TGV tool find the corresponding test case for each product, depending of the operational semantics of each product, computed from a UML model by Umlaut tool [12].

We propose a complete chain for test cases derivation from functional requirements: it consists in improving the use cases by declarative information under the form of contracts as an anchor for further testability purposes [26]. The test cases are then generated in two steps: use case orderings are deduced from use case contracts and then scenarios are substituted to each use case to produce a test case [28][29]. The theory and protype tools is being applied on 2 weapon systems in collaboration with TAS (Thales Aeroported Systems) within the Mutation project (see section 7.8).

6.2.2. Testability and Fault Diagnosis in OO Designs

Participants: Jean-Marc Jézéquel, Yves Le Traon, Franck Fleurey, Benoît Baudry.

While detecting testability weaknesses (called testability anti-patterns) of an OO design is a crucial task, one cannot expect from a non-specialist to make the right improvements, without guidance or automation. To overcome this limitation, we investigate solutions integrated to the OO process. We focus on the design patterns as coherent subsets in the architecture, and we explain how their use can provide a way for limiting the severity of testability weaknesses, and of confining their effects to the classes involved in the pattern [18]. Indeed, design patterns appear both as a usual refinement instrument, and a cause of complex interactions into a class diagram and more specifically of testability anti-patterns. To reach our objective of integrating

the testability improvement to the design process, we propose first a testability grid to make the relation between each pattern and the severity of the testability anti-patterns, and the we present our solution, based on a definition of patterns at metalevel, to automate the instantiation of patterns constrained by testability criteria [10].

The need for testing-for-diagnosis strategies has been identified for a long time, but the explicit connexion from testing to diagnosis is rare. We started with a study of an algorithm for fault localization that consists in cross-checking information collected from test cases execution traces. Analysing the type of information needed for an efficient localization, we identified several criteria that must be verified by the test cases in order to obtain an accurate diagnosis. We used these test adequacy criteria with automated test data generation to generate test cases for several JAVA programs. We then investigated and compared the relevance of those criteria by giving the generated test cases as an input to the fault localization algorithm. Based on the result of those studies, we proposed a test-for-diagnosis methodology [10].

6.2.3. Test Cases Synthesis

Participants: Claude Jard, Yves Le Traon, Benoît Baudry, Franck Fleurey.

While the generation of basic test cases set is easy for sequential systems, improving its quality may still require prohibitive effort. We focused on the issue of automating the test optimization. We first looked at genetic algorithms but to overcome disappointing experimentation results on the studied systems, we propose an original algorithm no longer at the "animal" level but at the "bacteriological" level. We called it bacteriological algorithm. We validated this algorithm on several .NET and Java systems using various test adequacy criteria: mutation score, code coverage and a criterion defined for an accurate diagnosis [10].

On the other hand, synthesis of distributed test cases is a complex problem from the beginning. We defined an automatic test generation method for distributed systems, based on a symbolic exploration of the system's state space [15]. The specification of the system is given as a set of elementary actions called *tiles*, which operate on symbolic variables and parameters through guarded assignments. From this specification a *causal graph* is then computed, which encodes the different histories of the systems, seen as partial orders. Finally, from a finite prefix of the causal graph we extract a set of distributed symbolic testers that can be executed against a distributed implementation of the system. This work combines for the first time the previous approaches on the generation of distributed testers using event structures developed by Claude Jard last year, and symbolic techniques, as developed in the Vertecs group by Vlad Rusu for example.

6.3. Scenarios for Distributed Systems Engineering

Participants: Loïc Hélouët, Claude Jard, Thomas Chatain, Emmanuel Donin de Rosière.

Key words: *Télécommunication*, *scenarios*, *partial orders*, *concurrency*, *UML*, *MSC*.

Scenarios are a partial order based formalism, that represents sample behaviors of a system. Despite their incompleteness, scenarios are useful to detect properties of a system at early stages of development.

6.3.1. Formal manipulation of scenarios

Scenarios depict interactions between objects. They are supposed to be incomplete and abstract representations of typical executions of a system. The building blocks for most of scenario languages are very simple chronograms. Formally, these chronograms can be considered as partial orders on events occurring in a system's execution.

However, simple partial orders are not sufficient to describe interesting behaviors. For this reason, most of scenario languages provide composition operators. This idea to design distributed systems behaviors with composition of partial orders is not new, and was proposed by [61]. In a more formal and generic way, scenario languages can be considered as partial order automatas. However, knowing a theoretical model behind scenarios does not solve all scenario problems. The languages obtained with algebraic compositions of partial order rapidly have the expressive power of rational subsets, a category of languages for which most of interesting properties are undecidable. For this reason it is impossible to decide in the general case if two MSCs describe the same sets of executions, or if the set of traces of a MSC is included in the traces of another,...

These undecidability results do not mean that scenarios are unusable, but rather that scenarios should be used with precaution. In fact, it is often possible to identify a class of scenarios for which a given formal manipulation is possible. A classification of scenarios is of primordial importance. So far, several classes of scenarios have been identified, among which:

- confluent scenarios [60]
- local choice [41]
- reconstructible scenarios [51]
- bounded scenarios [38]
- recognizable scenarios [59]

Among the main formal manipulations that are usually needed for a description language, projection is often a basic need. A part of the work on scenarios accomplished this year has concerned scenario projections. The main problem with a scenario projection is that the projection of a partial order automata can not be systematically described as a new partial order automata. Some pattern can not be decomposed as composition of communication patterns, and form a kind of infinite braid, that can not be cut into communication patterns without losing some causalities. We have proposed algorithms to detect when a HMSC projection can still be represented as an order automata, and to compute this projection [21].

This notion of projection is an essential manipulation for several applications. For example, projecting scenarios is essential to be able to detect common parts, or to restrict some behavior without losing causalities in behavioral product lines.

The simplicity of scenarios makes them a ideal language to capture requirements. However, when someone wants to define typical behaviors or conversely exceptional ones, the process usually end with a **set** of scenarios. These scenarios describe several "views" of the system, that are attached to specific functionalities or to specific contexts. A coherent behavior of the whole system is then a composition of all these views. However, this composition can not be performed with usual operators (sequence, alternative loops, and parallel composition), as views may contain a lot of redundancy. The ideal composition for scenarios can be seen as a kind of aspects weaving, and inherit techniques from program superposition. The difficult part when merging two scenarios is to match redundant parts in scenarios : matching is often undecidable, and there is usually more than one matching possibility. A solution is to this problem is to define explicitely which parts of composed scenarios are to be identified in the fusion. A recent proposal has been made for Live Sequence Charts [48] to compose several LSCs at runtime. However, this approach faces the matching problem.

With Benoît Caillaud, we have proposed another approach, consisting in a *fibered product* of scenarios. This fibered product merges two scenarios using an interface scenario that helps defining uniquely a desired matching. This product is an interesting operator, but the interface definition remains a difficult task. However, for some cases, the projection defined previously may help defining interfaces automatically.

6.3.2. Scenarios for security properties analysis

Recently, scenarios have shown to be a good model for analysis of security properties. As explained before, scenarios are useful to represent typical samples of systems behaviors. They are not good candidates for applications where exhaustive definition of behaviors is required. However, they can still provide useful analysis possibilities to distributed systems. For example, scenarios contain useful information about causality in distributed systems. This information can be used to identify illegal information flows in systems (this situation is currently called "covert channel"). Working on scenarios helps finding covert information passing strategies during early phases of a system's development, when modification costs are low.

We have proposed an algorithm to detect potential information flows in protocols from a scenarios description [24]. The principle of this algorithm is to detect if actions performed by an object identified as a "Sender" has observable consequences for another object called the "Receiver" (this property is often called **non interference** [47]), and if this information forms a code. So far, the strategies considered to send information were based on iteration of a set of scenarios. This strategy is very simple, and can be easily

detected by a monitoring application. However, more elaborated strategies using the protocol in a quasi-normal way can be used, and will hence be very difficult to detect at runtime. A possibility to consider all encoding strategies is to use game theory. In fact, a covert message passing can be seen as a game in which a pair (sender, receiver) play against a protocol and all other users to send information of unbounded size. This work has been performed during Aldric Degorre's training course.

The objective of this work on covert channel is to make sure that for a given protocol described through its scenarios of use, the bandwidth of any covert channels is lower than a given bound. So far, the framework we have proposed for bandwidth computation uses the (Max,+) algebra, and can only give rough upper and lower bounds. We are currently investigating with Eric Fabre and Aline Roumy from the Sigma2 team other solutions based on information theory to characterize more accurately maximal bandwidth and channel capacity for a given set of scenarios.

7. Contracts and Grants with Industry

7.1. ACCORD (rntl)

Participants: Jean-Marc Jézéquel, Noël Plouzeau, Éric Cariou, Jean-Philippe Thibault.

Key words: *UML*, *components*, *contracts*, *QoS*.

The RNTL ACCORD (Assembly and Contract Support for Component-Oriented Software Design) is a precompetitive project. Contract 202C022700 31330 011, 1/12/2001 – 1/12/2003.

Information systems design more and more often relies on building from software components. This approach is attractive, because it may reduce costs and delays, but includes risks, particularly when integrating components from diverse origins. A framework for analysis and design dedicated to components assembly has been set up to make complex, evolutive and distributed systems reliable. One of the ACCORD project objectives was to give a semantic to components and their assembly. Its approach was to give a specification, for each component, that describes precisely its assembly contracts. The project also manages components reusable interactions, as connectors that may be used in many assemblies. One of ACCORD results was the definition of an UML profile for "business components" allowing to express contracts. This gives to the UML the ability to manage components assembly, while remaining independent of concrete implementations on platforms like EJB, CCM or .NET. The ACCORD approach has the following advantages :

- It eases the understanding of a system whose functional description is complete: components and components assemblies;
- It makes the system more flexible: once assemblies are structured and set up, one may easily make (or remake) system elements (component, application, server, and so on);
- It increases reusability: on the one hand an assembly becomes itself a reusable element, on the other hand knowledge of valid assemblies leverages their understanding, and so their real reusability;
- It imposes nothing on the software infrastructure used for components implementation. Conformity between functional architecture and infrastructure is no longer implicit (hidden in the design) but is on the contrary an explicit projection, ready for tool assistance.

Many infrastructures for components development exist (COM+, .NET, CCM, EJB). However, the experience of our project partners showed that it was more interesting (and safer) to remain platform independent, but also that underlying models of these infrastructures do not allow to understand the system as a whole, seen as a reconfigurable assembly. This acknowledgement meets several OFTA (Observatoire Français des Techniques Avancées) recommendations.

ACCORD project was organized into three workpackages (WP):

- WP #1: abstract model definition to modelize components and assemblies. This abstract model stresses on semantic description of component functional and non functional properties, and on components cooperation description by means of contracts.
- WP #2: definition of projection rules from abstract models to technological models of CCM and EJB components, all these models being UML profiles.
- WP #3: validation of the two preceding sets by two implementations of components based applications. The first one concerns an information system (case study proposed by France Télécom); the other one deals with a coupling system for scientific software components (case study proposed by EDF).

The ACCORD project main results were:

- speeding up the state of the art on components assembly in the UML framework;
- improving the understanding and usage of "design by components" approach; implementations of the third set are examples of viability of the ACCORD approach;
- spreading results in accordance with UML 2.0 standardization. All profiles are "open sources" and UML 2.0 compliant.

7.2. MAGDA2 (rnrt)

Participants: Claude Jard, Julien Thomas.

Key words: *Network management*, *Distributed fault diagnosis*, *Petri nets unfolding*, *MPLS/SDH*.

Magda2 (modelisation et apprentissage pour la gestion distribuee des alarmes de bout en bout) is a national project within the RNRT program scope. November 2001 - November 2003. Partners : Alcatel, France-Telecom R&D, Ilog, Irisa, LIPN. In the following of the previous project Magda, we design and implement a new approach to distributed diagnostics, based on a UML modeling of faults and alarms propagation in heterogeneous networks. The Triskell and Sigma2 teams contribute together to this subject.

The Magda2 project is in continuation of the successful approach invented during the Magda project. This approach is completely new in the domain of alarm correlation and diagnosis. It is based on the automated synthesis of distributed diagnosers from a formal specification of the management layer, interpreted using a partial-order semantics of behaviors. Beyond the theory, we have acquired a significant experience on fault propagation in SDH/SONET networks and now in MPLS to deal with heterogeneous networks. One contribution of the Triskell project in 2003 was to define a UML framework to describe the management architecture (using a class diagram) and the elementary behaviors of the management objects (using simplified sequence diagrams). The model is compiled into abstract rules, written in JRules, to be used by the distributed diagnosis algorithm.

7.3. QCCS (IST)

Participants: Jean-Marc Jézéquel, Noël Plouzeau, Karine Macedo.

The IST QCCS project aims at designing a software development method and its associated tool. The method puts promotes the use of software components that include distribution and quality of service management. The method's process relies heavily on aspect-oriented design and programming techniques.

The work plan of the QCCS project was as follows:

- design of the component-based method;
- definition of the class of quality of service properties that are relevant to components and specification of a notation for these extrafunctional properties;
- definition of a framework and associated tools for component design, using an aspect-oriented approach;

- validation of the method and tools using three applications designed by the industrial partners of the QCCS project (“user-driven” experimentation).

The QCCS project started in november, 2000 with the following partners:

- Schlumberger/Sema (Spain) is in charge of the project management and is designing a workflow application to test the QCCS method and tools;
- the Irisa/Inria Rennes is in charge of the method design and participates in the tool design and test activities (with a man power of sixty man-months);
- the Technische Hochschule Berlin is in charge of the tool development and participates in the method design;
- the Czech company KD Software is in charge of designing another test application for the method;
- the University of Cyprus is in charge of designing a third application.

The project ended in septembre, 2003 with a positive evaluation from the project reviewers. The following publications were made during the QCCS work: [64] on the UML metamodel extension, [49] on the tool architecture.

7.4. ARTIST (IST)

The Triskell project is a member of the Artist project (Artist stands for Advanced Real-Time Systems Information Society Technologies, #IST-2001-34820). The Artist network aims at defining new research orientations in the field of embedded real time systems. The Triskell team is specifically involved in the *Component-based Design and Development* activity [33].

7.5. CAFÉ (ITEA Eureka)

Participants: Jean-Marc Jézéquel, Loïc Hélouët, Yves Le Traon, Benoît Baudry, Frédéric Fondement, Clémentine Nebut, Simon Pickin, Tewfik Ziadi.

Key words: *UML*, *products family*, *architecture recovery*, *patterns*, *methods*, *COTS*.

The main objective of CAFÉ was to develop a UML based infrastructure allowing for software product lines management. The CAFÉ project involves 27 partners, among which Philips, Nokia, Siemens, Fraunhofer IESE, Thales, Telvent, European Software Institute, Universidad Politecnica de Madrid, University of Essen, University of Helsinki. The Triskell team is involved in 3 tasks of CAFÉ.

System families are strategic assets, which can be used for European advancement. Prime export products are based on system families. As the importance of technology increases most of those products have become information intensive. Unfortunately software-engineering technology has been developed mainly for creating one product at a time and existing process models do not well address system family development.The companies participating in this consortium have investigated and developed technology for system families now for five years and experiences gained so far are very significant. The structuring of systems into product families allows sharing design effort within the product family and such counters the impact of ever growing complexity. This makes it possible to sustain, or even increase, the rate of product introduction.

Software development is influenced by concerns originating from several sources, grouped into 4 categories:

- **Business** concerns the way that profit is made by the resulting products,
- **Architecture** concerns the technology needed to build the system,
- **Organisation** concerns the organisation in which the software is developed,
- **Process** concerns the responsibilities and dependencies during software development.

The development of software for control, supervision and management of utilities falls within the category of complex system engineering. These systems have stringent requirements, such as time responsiveness in order to know the state of the utility distribution network in real time; customisability so that the system user interface can be used in different cultural contexts and according to different national standards of power distribution and management. The market for these systems can be characterised by two main issues: the liberalisation processes across Europe, which requires a short time-to-market for software providers; and the quality of service, as the main observable aspect for consumer satisfaction. System-family technology will improve the current situation and enable the development of the new generation of these systems, that can be characterised by their high quality, shorter life cycle, market orientation, customisation of user and control interfaces, and interoperability with third-party subsystems. In order to achieve it, the domain architecture must be discovered, and a component-based method specific to this application domain must be defined.

Significant research results have been obtained in CAFÉ in the area of System Family Engineering, mainly:

- an organisational process to evaluate opportunity of a product family development: catalogue of family development process implementation activities, calculus of return on investment, measurements on product-lines in place, viability estimation, transition to family development from single product devlopment.
- an architectural process to initiate a product line and make it a lasting process: separation of domain and family development, variability engeenering splitting product features in common and variable parts, asset management, component identification and adaptation, tool support.
- a quality process to improve competitivity: quality of architecture design and evaluation; strategy, methodology and process of testing, verification and validation of components.

7.6. FAMILIES (ITEA Eureka)

Participants: Jean-Marc Jézéquel, Loïc Hélouët, Yves Le Traon, Jacques Klein, Clémentine Nebut, Jean-Philippe Thibault, Tewfik Ziadi.

Key words: *UML*, *products family*, *architecture recovery*, *patterns*, *methods*, *COTS*.

FAMILIES is a next project in a sequence of following projects: ARES and PRAISE, then ESAPS, and CAFE.

ITEA projects ESAPS, and CAFÉ has lead to a recognized European community on the subject of System Family Engineering. The community presently has leadership over its American Counterpart, the SEI Product-Line Initiative. The FAMILIES project aims at growing the community, consolidating results into fact-based management for the practices of FAMILIES and its preceding projects, and to explore fields that were not covered in the previous projects, in order to complete the Framework.

The consolidating work in FAMILIES will lead to:

- A reuse economics framework, to deal with the questions on when, why and how a family approach has to be introduced. It is accompanied with a decision model, checklists and questionnaires.
  Work package 1: Reuse economics, Fact-based business and organisation maturity.
- A family maturity model, which will complement the CMM and CMMI maturity models.
  Work package 2: Family maturity, Fact-based process maturity, and consolidated tool requirements.
- Patterns, styles and rules related to satisfaction of business related quality requirements in the family, accompanied by quality models, supporting processes, check lists, questionnaires and approaches towards standardisation of quality of service requirements.
  Work package 3: Family quality, Fact-based architecture maturity.
- A methodology (process, tools, guidelines, and examples) supporting the separation of the domain aspects, the technical aspects (quality of services) and the technological aspects (platforms) in consistent models, in the MDA standardisation frame.
  Work package 4: Model driven family engineering.

- Extending reuse over larger parts of the organization, introducing an integrated approach to combine existing legacy assets into a family, or even to a system population.
  Work package 5: Families integration, Exploring reuse over family boundaries.

The project also has a specific work package, WP6 that takes care of exploitation and dissemination.

7.7. MOTOR (carroll)

Participants: Erwan Drézen, Jean-Marc Jézéquel, Damien Pollet, Jean-Philippe Thibault, Didier Vojtisek.

Key words: *model transformation*, *QVT*, *OMG*, *MDA*.

MOTOR (MOdel TransfORmations) project is part of CARROLL action with Thalès. March 2003 - March 2004. It also involves Inria/Atlas and CEA. Model transformation is a key aspect of the MDA(Model driven Architecture). It allows the automation and/or assistance for model creation from abstract phasis of the development to code generation. It gives formal, reproducible and conformance aspects in the engineering process. MOTOR aims to participate to the definition of these techniques in close relation to the RFP QVT (Query View Transformation) normalisation process at the OMG, to apply it to different problems and instrument it.

- MOTOR motivation
  MDA(Model Driven Architecture) is an important approach to enterprise-scale software development that is already having significant impact in the software industry. Many organizations are now looking at the ideas of MDA as a way to organize and manage their application solutions. It's a way to maximize ROI (Return On Investment) through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. A important need of this approach is the availablity of a model transformation structure. This need has been taken into account by the OMG (Object Management Group) through the standardization of a model transformation language QVT (Query View Transformation)
  This project is a collaboration between Thalès, CEA and the Atlas team.
- MOTOR impacts
  - Technical goals are :
    * The definition of the architecture needed for model transformation.
    * The application of the QVT technology in different areas like : model refinement or aspect weaving.
    * The creation of tools that validate the technical solutions.
  - Industrial goals are :
    * The automation of model transformation tasks.
    * The capitalization of the know-how in model transformation. This means the creation of reusable transformations using frameworks, libraries, etc.
    * The independence from the modelling tools. The end user may need to use his transformations from different repositories. The architecture must support a way to reuse these transformations with a minimal adaptation cost.
    * The reuse of legacy transformation. It already exists a lot of model transformations. It is unwise to rewrite them. The architecture must allow the connection and reuse of them for example as blackbox transformations.

– Standardization
The main work done in this project is closely related to the work currently on-going at OMG : the QVT RFP (Request For Proposal). This RFP received 8 submissions which where really different in their approaches as they do not take into account the same requirements . Then, the process which lead to a consensus is difficult. The risk to obtain a non satisfactory standard is really high, slowing down the advantages of the MDA approach.
So, MOTOR also have standardization actions at the OMG in order to assure ourselves that the normalized language takes into account every needed aspects of the model transformation.

The work is mainly visible through the evolution of the tool UMLAUT NG and the actions to the OMG.

7.8. MUTATION (carroll)

Participants: Yves Le Traon, Didier Vojtisek, Erwan Drézen, Frank Fleurey, Clémentine Nebut.

Key words: *test*, *UML*, *methodology*.

MUTATION ("Modélisation UML pour l'Automatisation de la production des tests" in french) is a project developed by CEA/LIST (LLSP), THALES Research and Technology, THALES Airborne Systems and INRIA. This project aims to increase productivity during the testing steps of the development process.

The purpose of MUTATION is to carry out a survey about the possibility to automate testing procedures. It holds the following parts:

- providing means to define testing scenarios at different levels of abstraction
- generation of testing cases
- assistance for understanding the generated testing cases through some criterions

MUTATION is a project developed by CEA/LIST (LLSP), THALES Research and Technology, THALES Airborne Systems and INRIA. This project aims to increase productivity during the testing steps of the development process.

The purpose of MUTATION is to carry out a survey about the possibility to automate testing procedures. It holds the following parts:

- providing means to define testing scenarios at different levels of abstraction,
- generation of testing cases,
- assistance for understanding the generated testing cases through some criterions

The technical issues of MUTATION are:

- defining rules in order to formalize requirements,
- defining rules in order to formalize the detailed software conception,
- automatic tests generation at different levels of abstraction,
- providing a low cost training for an industrial team

The three successive parts of MUTATION are:

- definition of a language dedicated to the writting of requirements and its associated methodology; a user guide and some examples are also to be written,
- defining a technology about the qualification of cover criterions and UML issues (UML extensions),

- applying the proposed concepts on a real case provided by THALES; it should allow to evaluate the improvement in term of productivity.

This project will allow THALES to evaluate the possibility to automate the generation of tests scenarios through UML models. At the end of the project, THALES shall have a methodology and technological items allowing to adapt the process used today within its teams.

8. Other Grants and Activities

8.1. National projects

8.1.1. CNRS action on Supervision and Control of Large Distributed Systems

Participant: Claude Jard.

Claude Jard, in cooperation with Jean-Bernard Stefani (INRIA Rhone-Alpes), have launched a prospective action on supervision of large dynamic distributed systems. The challenge is to fill the gap between the system scientific community (design of observable architectures, design of sensors) and the control community (algorithms). Three other groups participate: LIP6 in Paris, LAAS in Toulouse and LORIA in Nancy. The action began in September 2003 and will work during one year to produce a report. This project is granted by the CNRS/STIC.

8.1.2. CNRS action on MDA

Participants: Jean-Marc Jézéquel, Noël Plouzeau.

Triskell is participating to a prospective action on the subject of Model Driven Architecture. The challenge is to fill the gap between the various scientific community interested in models, from real-time to databases through software engineering. The action began in September 2003 and will work during two years to produce a report. This project is granted by the CNRS/STIC.

8.2. International working groups

8.2.1. Standardization at OMG

Triskell project participates to normalization action at OMG (http://www.omg.org/):

- Triskell project participates to the RFP MOF2.0 QVT Query/view/Transformation. This RFP standardizes a model transformation language which is a key point in efficiently applying MDA.
- Triskell project is also involved in other OMG groups which are related to the team interests. For example, it participates to the ORMSC group which formalizes the MDA approach, to the MDA user SIG which represents the end user point of view for MDA. It is also invloved in the more general Analysis and Design group which promotes standard modelling techniques including UML and MOF.

8.2.2. Collaboration with foreign research groups:

- Centre for Distributed Systems and Software Engineering, Monash University, Melbourne, Australia. Collaboration on Trusted Components and Contracts. Professor Heinz Schimdt has been invited in the Triskell team during 3 months in 2002. Christine Mingins has co-authored a book with J.-M. Jézéquel [55].
- Software engineering group (Pr. Keller's group), University of Montréal, Canada, on meta-modeling (H. Sahraoui).

- Asynchronous testing, The Triskell project has established strong links with the CRIM research center in Montreal, especially with the group headed by A. Petrenko. A. Petrenko has been spending 2 months with Triskell in 2003. During his stay, granted by the university of Rennes and the ENS Cachan, common research has been developed on the problem of asynchronous testing with Claude Jard, aiming at understanding the distortion of behaviors induced by a distant interaction with a system under test, using FIFO queues. A student of the ENS did his summer intership in this group at CRIM and collaboration is continuing.
- Carleton University, Ottawa, Canada: Triskell has developed a collaboration on test and objects with Lionel Briand's team at Carleton Univesity.
- Technical University of Munich, Germany on meta-modeling and agile methodologies. B. Rumpe, Editor in Chief of the SoSyM journal, was an invited professor with Triskell for 3 months in 2003.
- ETH Zurich (Pr. Meyer's team), Switzerland on Trusted Components. B. Meyer came to Rennes several times in the past few years.

9. Dissemination

9.1. Scientific community animation

9.1.1. Journals

9.1.1.1. Jean-Marc Jézéquel

is an Associate Editor of the following journals:

- IEEE Transactions on Software Engineering
- Journal on Software and System Modeling: SoSyM
- Journal of Object Technology: JOT
- L'Objet

9.1.2. Examination Committees

9.1.2.1. Jean-Marc Jézéquel

was in the examination committee of the following PhD thesis and "Habilitation à Diriger les Recherches":

- Gwen Salaun, june 2003, université de Nantes (chair) ;
- Eric Cariou, June 2003, université de Rennes (adviser);
- Benoit Baudry, June 2003, université de Rennes (adviser);
- Simon Pickin, July 2003, université de Rennes (adviser);
- Mickael Peltier, October 2003, université de Nantes (chair) ;
- Salah Sadou (HDR), November 2003, université de Bretagne Sud (referee) ;
- Jacob Zimmermann, December 2003, université de Rennes (chair) ;

9.1.2.2. Claude Jard

was in the examination committee of the following PhD thesis and "Habilitation a Diriger les Recherches":

- Simon Pickin, July 2003, university of Rennes (chair) ;
- Isabel Demongonin (HDR), December 2003, university of Nantes (referee) ;

9.1.2.3. Yves Le Traon
was in the examination committee of the following PhD

- Olfa Abdellatif-Kaddour, September 2003,INPT (LAAS-Toulouse) university of Rennes (referee) ;

9.1.3. Conferences

9.1.3.1. Jean-Marc Jézéquel
has been a member of the programme committee of the following conferences:

- UML'2003 : 6th International Conference on UML. San Francisco, october 2003.
- AFADL'2003 (Approche Formelle pour l'aide au développement logiciel), Rennes, January 2003: Chair of the programme committee.
- LMO'2003 (Langages et Modèles à Objets), Vannes, January 2003.
- WISME 2003 Workshop in Software Model Engineering, October 21st, 2003, San Francisco, USA.
- MDAFA 2003 Workshop on Model Driven Architecture: Foundations and Applications, June 26-27, 2003, University of Twente, Enschede, The Netherlands.
- CADS'03 ECOOP 2003 Workshop on Communication Abstractions for Distributed Systems, July 2003.
- TECHNIQUES FOR TRUSTED COMPONENTS 2003 Prato, Italy, 8-11 January 2003.

9.1.3.2. Claude Jard
has been a member of the programme committee of the following conferences:

- AFADL'2003 (Approche Formelle pour l'aide au developpement logiciel), Rennes, January 2003.
- FORTE'2003 : International Conference on Formal Description Techniques, Berlin, Germany, October 2003.
- MSR'2003 (Modelisation des systemes reactifs), Metz, October 2003, member of the steering committee.

9.1.3.3. Yves le Traon
was a member of the programme committee of the following conferences:

- ISSRE 2003 (Software Reliability Engineering), Fort Collins, USA, November 2003 [member].
- Software Metrics Symposium 2003, Ottawa, june 2003 [member].
-

9.1.4. Workshops

9.1.4.1. Yves le Traon
was General Chair of the following workshops:

- OCM 2003 (Objets, Composants, Modèles).
- AFADL 2003 (Approches Formelles dans l'Assistance au Développement de Logiciels).

J.-M. Jézéquel gave tutorials on "Model-Driven Engineering with Contracts, Patterns, and Aspects" at AOSD 2003 (2nd International Conference on Aspect-Oriented Software Development) and ASE 2003 (Automated Software Engineering). He also gave an invited talk at the Formal Methods for Components and Objects (FMCO'03) conference [37].

9.2. Teaching

Jean-Marc Jézéquel teaches OO Analysis and Design with UML (Iup3 and Diic2) at Ifsic, as well as at Supélec (Rennes) and ENSTB (Rennes). He also gives an advanced course on model driven engineering for Diic3 students.

Claude Jard teaches formal modeling and analysis of distributed systems, mainly at the Ecole Normale Superieure de Cachan.

Noël Plouzeau teaches OO Analysis and Design to DESS Isa (Ifsic).

Loïc Hélouët gives introductory courses on UML for the DU-GL at the IFSIC (Institut de Formation Supérieur en Informatique et Communication). He also teaches UML for the Mastere at the ENSTB (Ecole Nationale Supérieure de Télécommunications de Bretagne).

The Triskell team receives several DEA and summer trainees every year.

9.3. Miscellaneous

- J.-M. Jézéquel is Chair of the Steering Committee of the UML Conferences series. He is appointed to the board of the Committee of Projects of INRIA Rennes. He is Chair of the "Club Objet de l'Ouest", member of the Trusted Component Initiative steering committee, and member of the OFTA working group on Model Engineering.
- C. Jard is a member of section 7 board of the National Committee for scientific research. He also participates to scientific council of the STIC department at the CNRS. C. Jard is a member of the strategy committee of TNI-Valiosys, and of the scientific committee of the centre de recherche en informatique de Montréal, Canada. He is also member of the prioritary thematic network on complex embedded system of the CNRS.

10. Bibliography

Major publications by the team in recent years

[1] A. BEUGNARD, J.-M. JÉZÉQUEL, N. PLOUZEAU, D. WATKINS. *Making Components Contract Aware.* in « IEEE Computer », number 7, volume 13, July, 1999.

[2] E. FABRE, A. AGHASARYAN, A. BENVENISTE, C. JARD. *Fault Detection and Diagnosis in Distributed Systems : an Approach by Partially Stochastic Petri Nets.* in « Journal of Discrete Events Dynamic Systems », volume 8, 1998, pages 203-231.

[3] L. HÉLOUËT, C. JARD, B. CAILLAUD. *An Event Structure Semantics for Message Sequence Charts.* in « Mathematical Structures in Computer Science (MSCS) journal », volume 12, 2002, pages 377–403.

[4] C. JARD, J.-M. JÉZÉQUEL, A. L. GUENNEC, B. CAILLAUD. *Protocol Engineering using UML.* in « Annales des Telecoms », number 11–12, volume 54, November, 1999, pages 526–538.

[5] J.-M. JÉZÉQUEL. *Object Oriented Software Engineering with Eiffel.* Addison-Wesley, March, 1996, ISBN 1-201-63381-7.

[6] J.-M. JÉZÉQUEL. *Reifying Variants in Configuration Management.* in « ACM Transaction on Software Engineering and Methodology », number 3, volume 8, July, 1999, pages 284–295.

[7] J.-M. JÉZÉQUEL, J.-L. PACHERIE. *Object-Oriented Application Frameworks.* John Wiley & Sons, New York, 1999, chapter EPEE: A Framework for Supercomputing.

[8] J.-M. JÉZÉQUEL, M. TRAIN, C. MINGINS. *Design Patterns and Contracts.* Addison-Wesley, October, 1999, ISBN 1-201-30959-9.

[9] Y. L. TRAON, T. JÉRON, J.-M. JÉZÉQUEL, P. MOREL. *Efficient OO Integration and Regression Testing.* in « IEEE Trans. on Reliability », number 1, volume 49, March, 2000, pages 12–25.

Doctoral dissertations and "Habilitation" theses

[10] B. BAUDRY. *Assemblage testable et validation de composants.* Ph. D. Thesis, Université de Rennes 1, 2003.

[11] E. CARIOU. *Contribution à un processus de réification d'abstractions de communication.* Ph. D. Thesis, Université de Rennes 1, 2003.

[12] S. PICKIN. *Test des composants logiciels pour les télécommunications.* Ph. D. Thesis, Université de Rennes 1, 2003.

Articles in referred journals and book chapters

[13] A. BENVENISTE, E. FABRE, C. JARD, S. HAAR. *Diagnosis of asynchronous discrete event systems, a net unfolding approach.* in « IEEE Transactions on Automatic Control », number 5, volume 48, May, 2003, pages 714–727.

[14] L. HÉLOUËT. *Distributed system requirement modeling with Message Sequence Charts: the case of the RMTP2 Protocol.* in « International Journal of Information and Software Technology », number 11, volume 45, 2003.

[15] C. JARD. *Synthesis of distributed testers from true-concurrency models of reactive systems.* in « International Journal of Information and Software Technology », volume 45, 2003, pages 805–814.

[16] J.-M. JÉZÉQUEL, H. HUSSMAN. *Editorial for the Special Issue on the UML2002 Conference.* in « Journal on Software and System Modeling », number 3, volume 2, October, 2003, pages 151–152.

[17] J.-M. JÉZÉQUEL. *Model-Driven Engineering with Contracts, Patterns, and Aspects.* in « Tutorial Program of AOSD 2003: 2nd International Conference on Aspect-Oriented Software Development », ACM-IEEE, March, 2003.

Publications in Conferences and Workshops

[18] B. BAUDRY, Y. L. TRAON, G. SUNYÉ, J.-M. JÉZÉQUEL. *Measuring and Improving Design Patterns Testability.* in « Proceedings of Metrics Symposium 2003 », Sydney, Australia, September, 2003.

[19] A. BENVENISTE, S. HAAR, E. FABRE, C. JARD. *Distributed monitoring of concurrent and asynchronous systems (plenary address).* in « Proc. of CONCUR'2003, Marseille », series LNCS, Springer Verlag, 2003.

[20] J. BÉZIVIN, N. FARCET, J.-M. JÉZÉQUEL, B. LANGLOIS, D. POLLET. *Reflective Model Driven Engineering.* in « Proceedings of UML 2003, San Francisco », series LNCS, volume 2863, Springer, G. B. P. STEVENS, editor, pages 175–189, October, 2003.

[21] B. GENEST, L. HÉLOUËT, A. MUSCHOLL. *High-level Message Sequence Charts and Projections.* in « Proc of CONCUR 2003 », 2003.

[22] L. HÉLOUËT. *Projection et comparaison de Message Sequence Charts.* in « Proc. of Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'2003) », 2003.

[23] L. HÉLOUËT. *Projections et Comparaisons de Scénarios.* in « Proc of AFADL'2003, Approches Formelles dans l'Assistance au Développement de Logiciels », Rennes, France, Jan., 2003.

[24] L. HÉLOUËT, M. ZEITOUN, C. JARD. *Covert channels detection in protocols using scenarios.* in « Proc. of SPV'03 Security Protocols Verification », 2003.

[25] C. NEBUT, F. FLEUREY, Y. LE TRAON, J.-M. JÉZÉQUEL. *A Requirement-based Approach to Test Product Families.* in « Proc. of the 5th workshop on Product Families Engineering (PFE-05) », series LNCS, Springer Verlag, 2003.

[26] C. NEBUT, F. FLEUREY, Y. LE TRAON, J.-M. JÉZÉQUEL. *Requirements by Contracts allow Automated System Testing.* in « Proc. of the 14th. IEEE International Symposium on Software Reliability Engineering (ISSRE'03) », 2003.

[27] C. NEBUT, S. PICKIN, Y. LE TRAON, J.-M. JÉZÉQUEL. *Automated Requirements-based Generation of Test Cases for Product Families.* in « Proc. of the 18th IEEE International Conference on Automated Software Engineering (ASE'03) », 2003.

[28] B. RUMPE. *An Agile Test-based Modeling Approach using UML.* in « International Conference on Software Engineering Research and Practice (SERP'03) », Las Vegas, Nevada, USA, June, 2003.

[29] B. RUMPE. *Model-based Testing of Object-Oriented Systems.* in « Formal Methods for Components and Objects (FMCO'02) », series LNCS, Springer Heidelberg, J. S. DE BOER, editor, 2003.

[30] T. ZIADI, L. HÉLOUËT, J.-M. JÉZÉQUEL. *Modélisation de lignes de produits en UML.* in « Proc. of LMO 2003, Langages et Modèles à Objets », Vannes, France, Fev., 2003.

[31] T. ZIADI, L. HÉLOUËT, J.-M. JÉZÉQUEL. *Toward a UML Profile for Software Product Lines.* in « Proceedings of the Fifth Internationl Workshop on Product Familly Engineering (PFE-5) », series LNCS, Springer Verlag, 2003.

[32] T. ZIADI, J.-M. JÉZÉQUEL, F. FONDEMENT. *Product Line Derivation with UML.* in « Proceedings Software Variability Management Workshop, University of Groningen Departement of Mathematics and Computing Science », February, 2003.

Internal Reports

[33] E. BRINKSMA, G. COULSON, I. CRNKOVIC, A. EVANS, S. GÉRARD, S. GRAF, H. HERMANNS, J.-M. JÉZÉQUEL, B. JONSSON, A. RAVN, P. SCHNOEBELEN, F. TERRIER, A. VOTINTSEVA. *Component-based Design and Integration Platforms: a Roadmap.* Technical report, number IST-2001-34820, The ARTIST consortium, April, 2003.

[34] J. BÉZIVIN, P. VALDURIEZ, J.-M. JÉZÉQUEL, R. MARVIE, J.-M. GEIB. *The MDA vision at INRIA.* Document ad, number 2003-06-01, OMG, June, 2003.

[35] N. PLOUZEAU, J.-M. JÉZÉQUEL, T. WEIS. *Final report on the QCCS method and tools.* Technical report, number D333, The QCCS consortium, September, 2003.

[36] N. PLOUZEAU. *Final specification document of the weaving process.* Technical report, number D361, The QCCS consortium, july, 2003.

Miscellaneous

[37] J.-M. JÉZÉQUEL. *Model-Driven Engineering: Basic Principles and Challenges.* Invited Presentation at Formal Methods for Components and Objects (FMCO'03), Leiden, Netherlands, November, 2003.

Bibliography in notes

[38] R. ALUR, M. YANNAKAKIS. *Model Checking of Message Sequence Charts.* in « Proc. 10th Intl. Conf. on Concurrency Theory », Springer Verlag, pages 114-129, 1999.

[39] B. APPLETON. *Patterns and Software: Essential Concepts and Terminology.* in « Object Magazine Online », May, 1997.

[40] A. ARNOLD. *Systèmes de transitions finis et sémantiques de processus communicants.* Masson, 1992, 196 p..

[41] H. BEN-ABDALLAH, S. LEUE. *Syntactic Detection of Process Divergence and non-Local Choice in Message Sequence Charts.* in « Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'97 », series Lecture Notes in Computer Science, volume 1217, Springer-Verlag, E. BRINKSMA, editor, pages 259 - 274, Enschede,The Netherlands, April, 1997.

[42] A. BEUGNARD, J.-M. JÉZÉQUEL, N. PLOUZEAU, D. WATKINS. *Making Components Contract Aware.* in « IEEE Computer », number 7, volume 13, July, 1999.

[43] G. BOOCH. *Object-Oriented Analysis and Design with Applications.* edition 2nd, Benjamin Cummings, 1994.

[44] L. CASTELLANO, G. DE MICHELIS, POMELLO,L.. *Concurrency versus Interleaving: An Instructive Example.* in « BEATCS: Bulletin of the European Association for Theoretical Computer Science », volume 31, 1987.

[45] S. FROLUND, J. KOISTINEN. *QML: A Language for Quality of Service Specification.* 1998.

[46] E. GAMMA, R. HELM, R. JOHNSON, J. VLISSIDES. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley, 1995.

[47] J. GOGUEN, J. MESEGUER. *Security policies and security Models.* in « Proc of IEEE Symposium on Security and Privacy », I. C. S. PRESS, editor, pages 11-20, April, 1982.

[48] D. HAREL, R. MARELLY. *Come, Let's Play : Scenario-Based Programming Using LSCs and the Play-Engine.* Springer, 2003.

[49] W. HO, J.-M. JÉZÉQUEL, F. PENNANEAC'H, N. PLOUZEAU. *A Toolkit for Weaving Aspect Oriented UML Designs.* in « Proceedings of 1st ACM International Conference on Aspect Oriented Software Development, AOSD 2002 », Enschede, The Netherlands, April, 2002.

[50] L. HÉLOUËT. *Analyse des exigences des systèmes répartirs exprimées par des langages de scénarios.* Ph. D. Thesis, Université de Rennes 1, Octobre, 2000.

[51] L. HÉLOUËT, C. JARD. *Conditions for synthesis of communicating automata from HMSCs.* in « 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS) », GMD FOKUS, S. GNESI, I. SCHIEFERDECKER, A. RENNOCH, editors, Berlin, avril, 2000.

[52] M. JACKSON. *System Development.* Prentice-Hall International, Series in Computer Science, 1985.

[53] C. JARD. *Vérification dynamique des protocoles.* Habilitation à diriger les recherches de l'université de Rennes 1, décembre, 1994.

[54] J.-M. JÉZÉQUEL, B. MEYER. *Design by Contract: The Lessons of Ariane.* in « Computer », number 1, volume 30, January, 1997, pages 129–130.

[55] J.-M. JÉZÉQUEL, M. TRAIN, C. MINGINS. *Design Patterns and Contracts.* Addison-Wesley, October, 1999, ISBN 1-201-30959-9.

[56] G. KICZALES, J. LAMPING, A. MENDHEKAR, C. MAEDA, C. VIDEIRA LOPES, JEAN-MARC. LOINGTIER, J. IRWIN. *Aspect-Oriented Programming.* in « ECOOP '97 — Object-Oriented Programming 11th European Confer ence », 1997.

[57] B. MEYER. *Reusability: The Case for Object-Oriented Design.* in « IEEE SOFTWARE », number 3, March, 1987, pages 50–64.

[58] B. MEYER. *Applying "Design by Contract".* in « IEEE Computer (Special Issue on Inheritance & Classification) », number 10, volume 25, October, 1992, pages 40–52.

[59] R. MORIN. *Recognizable sets of Message Sequence Charts.* in « Proc. of STACS 2002 », series LNCS, number 2285, Springer-Verlag, pages 523, 2002.

[60] A. MUSCHOLL. *Matching Specifications for Message Sequence Charts.* in « Lecture Notes in Computer Science », volume 1578, 1999, pages 273–287.

[61] V. PRATT. *Modeling Concurrency with Partial Orders.* in « International journal of Parallel Programming », number 1, volume 15, Mai, 1986, pages 33-71.

[62] C. SZYPERSKI. *Component Software: Beyond Object-Oriented Programming.* ACM Press and Addison-Wesley, New York, N.Y., 1998.

[63] J. WARMER, A. KLEPPE. *The Object Constraint Language.* Addison-Wesley, 1998.

[64] T. WEIS, C. BECKER, K. GEIHS, N. PLOUZEAU. *An UML Meta Model for Contract Aware Components.* in « Proceedings of UML 2001 », series LNCS, volume 2185, Springer Verlag, pages 442–456, 2001.

[65] G. WINSKEL. *Event Structures.* in « Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course », volume 255, Springer-Verlag, G. R. W. BRAUER, editor, pages 325-392, Bad Honnef, september, 1986.