INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Triskell

Model Driven Engineering for Component Based Software

Rennes

THEME COM

Table of contents

1. Team

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

Administrative assistant
Myriam David [TR Inria]

Inria staff
Benoit Baudry [Research scientist Inria]
Didier Vojtisek [Research engineer Inria]

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
Pierre-Alain Muller [Assistant Professor Université de Mulhouse]

Post-doc
Olivier Defour [from September 1st, 2003 to February 28th, 2005]

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

PhD Students
Franck Chauvel [INRIA grant since October 2004]
Franck Fleurey [MENRT grant]
Marouane Himdi [CIFRE grant since March 2004]
Jacques Klein [INRIA grant]
Karine Macédo [INRIA grant until May 2004]
Christophe Métayer [CIFRE grant]
Clémentine Nébut [INRIA-région grant until November 2004]
Damien Pollet [MENRT grant]
Sébastien Saudrais [INRIA grant since October 2004]
Jim Steel [INRIA grant]
Tewfik Ziadi [INRIA grant until December 2004]

2. Overall Objectives

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

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 non-functional 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, 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 our techniques 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

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

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 [51][48]: 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 [55].

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 [50]. 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 [45]. 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 [50]. 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 concept also provides the bases needed to develop *software components*, for which Szyperski's definition [57] 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. .Net, 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.

Empirical methods without real component composition models have appeared during the emergence of a real component industry (at least in the Windows world). These methods are now clearly the cause of untractable validation and of integration problems that can not be transposed to more critical systems (see for example the accidental destruction of Ariane 501 [53]).

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 requirements and benefits among stake-holder components, for example between the client of a service and its suppliers (including subcomponents). 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 [56] 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) [58]. 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) [47].

3.2.6. Modeling with the UML

As in other sciences, we are increasingly resorting to modelling to master the complexity of modern software development. According to Jeff Rothenberg,

> *Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.*

The massive adoption of Unified Modeling Language (UML) in many industrial domains open new perspectives to make the underlying ideas on modeling 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 and 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.

3.2.7. Model Driven Engineering

Usually in science, a model has a different nature that the thing it models ("do not take the map for the reality" as Sun Tse put it many centuries ago). Only in software and in linguistics a model has the same nature as the thing it models. In software at least, this opens the possibility to automatically derive software from its model. This property is well known from any compiler writer (and others), but it was recently be made quite popular with an OMG initiative called the Model Driven Architecture (MDA).

The OMG has built a meta-data management framework to support the MDA. It is mainly based on a unique M3 “meta-meta-model” called the Meta-Object Facility (MOF) and a library of M2 meta-models, such as the UML (or SPEM for software process engineering), in which the user can base his M1 model.

The MDA core idea is that it should be possible to capitalize on platform-independent models (PIM), and more or less automatically derive platform-specific models (PSM) –and ultimately code– from PIM through model transformations. But in some business areas involving fault-tolerant, distributed real-time computations, there is a growing concern that the added value of a company not only lies in its know-how of the business domain (the PIM) but also in the design know-how needed to make these systems work in the field (the transformation to go from PIM to PSM). Reasons making it complex to go from a simple and stable business model to a complex implementation include:

- Various modeling languages used beyond UML,
- As many points of views as stakeholders,
- Deliver software for (many) variants of a platform,
- Heterogeneity is the rule,
- Reuse technical solutions across large product lines (e.g. fault tolerance, security, etc.),
- Customize generic transformations,
- Compose reusable transformations,
- Evolve and maintain transformations for 15+ years.

This wider context is now known as Model Driven Engineering.

3.3. Mathematical foundations for distributed and reactive software

Keywords: *Labeled transition systems, event structures, partial orders.*

Labeled transitions systems are the mathematical structures that characterize best the foundations of research on software models [46]. 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 [52].

3.3.1. Transition systems

A labeled 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 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 [49] 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 $(E, \leq, \Pi, \varphi, \Sigma, I)$ 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.
- $\varphi : E \rightarrow \Sigma$ 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* [59] is a partial order equipped with an additional binary conflict relation. An event structure is usually defined by a tuple $(E, \leq, \sharp, \Pi, \varphi, \Sigma, I)$ where:

- $E, \leq, \Pi, \varphi, \Sigma, I$ 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

Keywords: *UML, distributed systems, software engineering, telecommunication, test.*

In large scaled distributed systems such as developed for telecommunications, building 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 from UML models.

The research activity in Triskell focuses at the same time on development efficiency and reliability. Our main applications 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.

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

Participants: Franck Chauvel, Erwan Drézen, Franck Fleurey, Jean-Marc Jézéquel, Damien Pollet, Jim Steel, Jean-Philippe Thibault, Didier Vojtisek [correspondant].

MDA 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 the engineers to formalize and automate the use of PIM (*Platform Independent Model*) and PSM (*Platform Specific Model*). The resulting design lifecycle creates platform independent abstract models 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 main point to be addressed is the model transformation part of the problem. Triskell reuses its expertise acquired with its tool UMLAUT and improved it to deal with MDA specificities. Thus, UMLAUT evolved into UMLAUT NG (*next generation*) in order to use it in a wider range of applications. In addition to the manipulation of UML models, UMLAUT NG adds the ability to manipulate any kind of models on any kind of repositories. A transformation can be run on any repository that has compatible metamodels. The metamodels are defined using the 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 drove UMLAUT NG evolution is that a transformation is a kind of program so it must be possible to apply the MDA approach to itself.

As a central tool in the team, UMLAUT NG helps us investigating various 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.

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

Since UMLAUT NG was integrated into Eclipse environment, UMLAUT NG is now used by a growing community in the domain of model transformation. Amoung other we have users within: CEA, ENSIETA, ENST Bretagne, Swiss Federal Institute of Technology (Switzerland), University of Muenster (Germany), etc.

In 2004, UMLAUT NG was used within these projects in collaboration with industry:

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;

Itea Families with (in France) Softeam, Thalès, about transformation of product lines, as the continuation of the Itea project CAFE.

5.2. Mutator : Mutation testing tool family for OO programs

Keywords: *.Net, Java, Test, test by mutation.*

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

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 software proposes specific OO mutation operators and the corresponding tools for Java and C# programs since the Mutator line of mutation tools is available for Java and C# languages. This work has been carried out in collaboration with Daniel Deveaux from UBS.

5.3. Requested : a toolbox for requirement simulation and testing

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

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

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 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. Contract-based and Aspect Oriented Design

6.1.1. Modelling Quality of Service Aspects: Application to Software Components

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

Software components reuse reduces the global development costs. A software component can be defined as a composition unit equipped with interfaces and contextual dependencies. Nowadays, the choice of a peculiar software component depends on the services provided, but also on the quality of its realisation, which is defined by extrafunctional properties. These properties are also called quality of service (QoS) properties. As they will guide the choice of developpers, they must be considered from the beginning of the conception phase. Developers must define a desired quality of service, design quality management policies, i.e. policies that control the quality effectively provided by a component, and allow contract negociations. The concept of quality of service contract will be the fundamental concept for the solutions proposed in this thesis. Contracts specify rights and obligations for a client and a service provider. The notion of contract will provide feedback to an application hence allowing it to react to its environment through renegociation mechanisms. In 2003 the Triskell team has worked on QoS contract specification and implementation within the framework of the european project QCCS (www.qccs.org). The project's main contribution is a QoS contracts specification methodology for components, and its integration within a component-based developement environment relying on aspectoriented conception and realisation. First, the UML notation has been extended to provide component-based software designers with a conception and specification tool dealing with quality of services. At creation time of the contractualized components model designers can generate automatically these components equipped with a monitoring systems that checks if a quality of service is effectively reached, and with a contract renegociation mechanism. The whole work on this topic in described in Karine Macedo de Amorim's PhD thesis [10].

6.1.2. Extra-functional contract support in components

Participants: Olivier Defour, Jean-Marc Jézéquel, Noël Plouzeau.

According to Szyperski, a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. But it is well known that these contractually specified interfaces should go well beyond mere syntactic properties: they should also involve functional ones, as well as synchronization and Quality of Service (QoS) features. In large, mission-critical component based systems, it is also particularly important to be able to explicitly relate the QoS contracts attached to provided interfaces with the QoS contracts obtained from required interfaces. In [27] we propose a language called QoSCL (defined as an add-on to the UML2.0 component model) to let the designer explicitly describe and manipulate these higher level contracts and their dependencies. We show how the very same QoSCL contracts can then be exploited for validation of individual components and also validation of a component assembly, including getting end-to-end QoS information inferred from individual component contracts, by automatic translation to a constraint logic programming language. We illustrate our approach with the example of a GPS (Global Positioning System) software component, from its functional and contractual specifications to its implementation in a .Net framework.

6.1.3. Applying CLP to predict extra-functional properties of component-based models

Participants: Olivier Defour, Jean-Marc Jézéquel, Noël Plouzeau.

A component is the basic re-usable unit of composition to build composite systems by connecting to others through their provided and required ports. Checking the functional compliance between provided and required ports is necessary to build functional systems. At the same time, one of the most important issues today in Component-Based Software Engineering (CBSE) is the prediction of the composite structure Quality of Service (QoS) at design time, using the extrafunctional properties of its components. The Triskell has published results in that field in [26]. Our wourk focuses on specific CBSE issues, and the use of Constraint

Logic Programming (CLP) in this context. For each component providing and requiring services, we propose to specify the QoS properties as required and provided operations, called dimensions, on the component ports. In this model, a QoS property can depend on other QoS attributes, and be constrained by OCL pre and post-conditions. From this model, the QoS aspect of a component is translated into a QoS system of non-linear constraints over the reals: the dimensions and their pre/post-conditions as variables controlled by non-linear constraints. These constraints are either inequalities that bound the admissible QoS values, or non-linear functions that bind QoS properties between them. Using the CLP, we are able to determine if a QoS system can be satisfied, and to predict what quality level is required by the assembly from its environment, as a set of admissible intervals. The CLP is a general framework that can be implemented with a realistic effort, to reason about the component-based models QoS properties at design time, that is one of the most important issues in CBSE.

6.2. Model-Based Testing

6.2.1. Automatic Test Cases Optimization: a Bacteriologic Algorithm

Participants: Benoit Baudry, Yves Le Traon, Jean-Marc Jézéquel.

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: faults are injected into the software component (making mutants out of it) to check the proportion of mutants detected by the test cases (in the mutation terminology, the mutant is said "killed" by the test case). But while the generation of basic test cases set is easy, improving its quality may require prohibitive effort. This work focuses on the issue of automating the test optimization. The application of genetic algorithms looks like an interesting way to solve it. The optimization problem is modeled as follows from an evolutionary point of view: a test case can be considered as a predator while a mutant program is analogous to a prey. The aim of the selection process is to generate test cases able to kill as many mutants as possible, starting from an initial set of predators, which is the test cases set provided by the programmer. To overcome disappointing experimentation results, on .Net components and Eiffel classes, a slight variation on this idea is studied, no longer at the "animal" level (lions killing zebras) but at the bacteriological level. The bacteriological level indeed better reflects the test case optimization issue: it mainly differs from the genetic one by the introduction of a memorization function and the suppression of the cross over operator. [16] explains how the genetic algorithms have been adapted to fit with the issue of test optimization. [17] shows the properties of the resulting algorithm which differs so much from genetic algorithms that it has been given another name: bacteriological algorithm.

6.2.2. Measuring Design Testability of a UML Class Diagram

Participants: Benoit Baudry, Yves Le Traon.

Design-for-testability is a very important issue in software engineering. It becomes crucial in the case of OO designs where control flows are generally not hierarchical, but are diffuse and distributed over the whole architecture. We concentrate on detecting, pinpointing and suppressing potential testability weaknesses of a UML class diagram ([18]). The attribute significant from design testability is called “class interaction” and is generalized in the notion of testability anti-pattern: it appears when potentially concurrent client/supplier relationships between classes exist in the system. These interactions point out parts of the design that need to be improved, driving structural modifications or constraints specifications, to reduce the final testing effort. The testability measurement we propose counts the number and the complexity of interactions that must be covered during testing. The approach is illustrated on application examples, taken from varous application domains (a distributed chat software, a compiler, all the catalogue of design patterns from the GoF).

[24] synthesizes our research efforts in the field of object-oriented design testability measurement, that has also been done in collaboration with Gerson Sunyé from LINA lab. These efforts have two different goals. First, we identify recurrent design structures (or testability anti-patterns) that worsen software testability. Second, we use the UML extension mechanisms to better specify design information that can make implementation more testable. Although detecting testability anti-patterns during software 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, each definition of an anti-pattern is associated with an alternative design solution.

6.2.3. From Testing to Diagnosis: An Automated Approach

Participants: Franck Fleurey, Benoit Baudry, Yves Le Traon.

The need for testing-for-diagnosis strategies has been identified for a long time, but the explicit link from testing to diagnosis is rare. Here (see [28]), we start with the study of an algorithm for fault localization that consists of cross-checking information collected from test cases execution traces. Analyzing the type of information needed for an efficient localization, we identify the attribute (called Dynamic Basic Block) that restricts the accuracy of a diagnosis algorithm. Based on this attribute, a test criterion is proposed and validated through rigorous case studies: it shows that test cases can be completed to reach a high level of diagnosis accuracy. So, the dilemma between a reduced testing effort (with as few test cases as possible) and the diagnosis accuracy (that needs as much test cases as possible to get more information) is partly solved by selecting only test cases relevant for diagnosis.

6.2.4. System Testing of Product Families: from Requirements to Test Cases

Participants: Clémentine Nebut, Erwan Drézen, Yves Le Traon, Jean-Marc Jézéquel.

In large software system, it is mandatory to address the question of continuity in the refinement process from high level textual requirements to analysis, design and test. We focus on functional requirements, expressing the expected system services. The main constraint we seek to solve here is to remain both domain-independent (so portable for other application contexts), and to let the possibility for a company to create its own requirement assets. In this domain, our contribution is two-fold: a method [35], supported by a tool [21], to disambiguate incrementally textual requirements and create requirement assets (called “interpretation patterns”), and a new to use the underlying formal model as a first basis to ensure refinement continuity, by requirements simulation, and automated derivation of early analysis (use cases) and design models, expressed within the UML [11]. The approach has been applied on two components of last generation combat aircrafts developed by THALES Airborne System.

6.2.5. Contract-Based Testing: from Objects to Components and from contracts to diagnosis probes

Participants: Marouane Himdi, Yves Le Traon, Olivier Defour.

Contracts on classes have been first developed as an OO software design approach. They were also quickly used for supporting class testing, providing a form of design for testability. We identify the tracks to extend the contract-based built-in test technique to hierarchical components.

To do that, we build on our previous work on *stclass*, a framework supporting Design by Contract and built-in test for `Java`, and on *confract*, a contracting system for the *fractal* component platform. Tests are embedded in the components and are generated with respect to a category of contract (library, interface, composition). In collaboration with D. Deveaux, P. Collet and R. Rousseau [25], we studied how this approach, firstly dedicated to objects, can be valuable for components testability. As a result of the test process, the embedded contracts are more robust and offer an original way to improve the observability of the component-based system. Contracts make it aware of its execution, and thus able to detect erroneous behaviors at runtime.

In addition to detection of errors related to design, coding or deployment of an application, the diagnosis is a well-known technique for understanding the behaviour of a software system and an absolute requirement for its improvement. Unfortunately, applications become more difficult to diagnosis as functionalities provided become complex.

In collaboration with the KEREVAL company, we explore the use of dynamic probes (sensors) that will be injected into running system to collect various information. The innovative part of this approach is the use of generic probes to develop diagnosis framework [30].

6.3. Model-Driven Engineering

6.3.1. Meta-Model Independant Model Transformations

Participants: Damien Pollet, Jean-Marc Jézéquel, Didier Vojtisek, Pierre-Alain Muller, Jim Steel.

Model engineering attempts to solve how we can evolve complex software systems. Indeed, those systems must follow the evolution of new requirements and technologies, and this evolution is faster and faster compared to the business domain evolution. We thus propose to reuse the domain expertise independantly of any underlying technology, through model transformation techniques [38][37].

The contribution presented in [12] is an architecture for manipulating models which is independant of any specific metamodel. During development of model transformations, this architecture supports proven techniques of object-oriented software engineering. A reference implementation in functional programming specifies the semantics of the interface for accessing models.

Our approach is based on a MOF-level interface (MOF: Meta-Object Facility) for model manipulation. The associated programming language supports direct manipulation of model elements, because the metamodel structure dynamically extends the set of types available to the model transformation program. From a methodological point of view, we show that model transformations capture the implementation expertise for a business domain to a given technology ; it is therefore useful to model and develop complex transformations using sound software engineering and model engineering techniques. We illustrate this in practice using transformations for refactoring UML models (UML: Unified Modeling Language).

6.3.2. Statecharts transformation: a bridge to make event-based B more usable

Participants: Christophe Métayer, Franck Chauvel, Yves Le Traon, Jean-Marc Jézéquel.

Formal method and specially the B method offers advantages over traditional approaches in terms of maintainability, security and reliability. The main difficulty of formal method is the proof. Using Statecharts simplify the edition of a model B and allow control for the refinement transformation. We studied how the transformation from Statecharts into a B model. This transformation is close to classical techniques used in translation between B language and other languages. These technique was explored especially in the European project BOM and described in the article [15]. We now intend to empirically evaluate the techniques studied during the first year.

6.3.3. Statechart Synthesis with an Algebraic Approach for Product Lines

Participants: Tewfik Ziadi, Jean-Marc Jézéquel.

Software Product Line design techniques relying on UML model transformations are becoming mainstream. Both static and behavioral aspects of the Software Product Line and its derivation into a specific product are examined in [23][14]. Transformations algorithms are given to support these design techniques, and an algebraic approach is proposed to capture behavioral requirements specification and their translation into statecharts diagrams.

The idea of synthesizing statecharts out of a collection of scenarios has received a lot of attention in recent years. However due to the poor expressive power of first generation scenario languages, including UML1.x sequence diagrams, the proposed solutions often use ad hoc tricks and suffer from many shortcomings. The recent adoption in UML2.0 of a richer scenario language, including interesting composition operators, now makes it possible to revisit the problem of statechart synthesis with a radically new approach. Inspired by the way UML2.0 sequence diagrams can be algebraically composed, we first defined an algebraic framework for composing statecharts [42]. Then we showed how to leverage the algebraic structure of UML2.0 sequence diagrams to get a direct algorithm for synthesizing a composition of statecharts out of them. The synthesized statecharts exhibit interesting properties that make them particularly useful as a basis for the detailed design process. Beyond offering a systematic and semantically well founded method, another interest of our approach lies in its flexibility: the modification or replacement of a given scenario has a limited impact on the synthesis process, thus fostering a better traceability between the requirements and the detailed design [14].

6.3.4. Transformation of behavioral models based on compositions of sequence diagrams

Participants: Jacques Klein, Noël Plouzeau.

This work focuses on a behavior composition technique, based on sequence diagrams mergings, that allows a unique result model even in the case of multiple weavings (i.e. weaving more than two fragments) [34]. We distinguish the composition of models at a low level of abstraction from the composition at a high level of abstraction. The technique is illustrated by an observer pattern specification. Composing behavioral elements (which can be seen as aspects) at model level may play an important role within both MDA and aspect based software processes.

6.3.5. MDE and Validation: Testing Model Transformations

Participants: Franck Fleurey, Jim Steel, Benoit Baudry, Yves Le Traon.

The OMG's Model-Driven Architecture is quickly attracting attention as a method of constructing systems that offers advantages over traditional approaches in terms of reliability, consistency, and maintainability. The key concepts in the MDA are models that are related by model transformations. However, to provide an adequate alternative to existing approaches, MDA must offer comparable support for software engineering processes such as requirements analysis, design and testing. We attempt to explore the application of the last of these processes, testing ([29], to the most novel part of the MDA, that of model transformation. In ([29], we present a general view of the roles of testing in the different stages of model-driven development, and a more detailed exploration of approaches to testing model transformations. Based on this, we highlight the particular issues for the different testing tasks, including adequacy criteria, test oracles and automatic test data generation. We also propose possible approaches for the testing tasks, and show how existing functional and structural testing techniques can be adapted for use in this new development context.

7. Contracts and Grants with Industry

7.1. AOSD-Europe (Network of Excellence)

Keywords: *Aspect Oriented Design*.

Participants: Jean-Marc Jézéquel, Noël Plouzeau, Yves Le Traon, Jacques Klein, Sébastien Saudrais, Didier Vojtisek.

Aspect-Oriented Software Development (AOSD) supports systematic identification, modularisation, representation and composition of crosscutting concerns such as security, mobility, distribution and resource management. Its potential benefits include improved ability to reason about the problem domain and corresponding solution; reduction in application code size, development costs and maintenance time; improved code reuse; architectural and design level reuse by separating non-functional concerns from key business domain logic; improved ability to engineer product lines; application adaptation in response to context information and better modelling methods across the lifecycle. AOSD-Europe will harmonise and integrate the research, training and dissemination activities of its members in order to address fragmentation of AOSD activities in Europe and strengthen innovation in areas such as aspect-oriented analysis and design, formal methods, languages, empirical studies and applications of AOSD techniques in ambient computing. Through this harmonisation, integration and development of essential competencies, the AOSD-Europe network of excellence aims to establish a premier virtual European research centre on AOSD. The virtual research centre will synthesise the collective viewpoints, expertise, research agendas and commercial foci of its member organisations into a vision and pragmatic realisation of the application of AOSD technologies to improve fundamental quality attributes of software systems, especially those critical to the information society. It will also act as an interface and a centralised source of information for other national and international research groups, industrial organisations and governmental bodies to access the members' work and enter collaborative initiatives. The existence of such a premier research base will strengthen existing European excellence in the area, hence establishing Europe as a world leader.

Project duration: 2004-2011

Project budget: 9.6 Meuros

Project Coordinator: University of Lancaster

Participants: University of Lancaster, Technical University of Darmstadt, INRIA, VUB, Trinity College Dublin, University of Malaga, Katholieke Universiteit Leuven, Technion, Siemens, IBM Hursley Development Laboratory

7.2. Artist2 (Network of Excellence)

Keywords: *Real-Time Component Models*.

Participants: Jean-Marc Jézéquel, Noël Plouzeau, Benoit Baudry.

The strategic objective of the ARTIST2 Network of Excellence is to strengthen European research in Embedded Systems Design, and promote the emergence of this new multi-disciplinary area. We gather together the best European teams from the composing disciplines, and will work to forge a scientific community. Integration will be achieved around a Joint Programme of Activities, aiming to create critical mass from the selected European teams.

The ARTIST2 Network of Excellence on Embedded Systems Design will implement an international and interdisciplinary fusion of effort to create a unique European virtual centre of excellence on Embedded Systems Design. This interdisciplinary effort in research is mandatory to establish Embedded Systems Design as a discipline, combining competencies from electrical engineering, computer science, applied mathematics, and control theory. The ambition is to compete on the same level as equivalent centres in the USA (Berkeley, Stanford, MIT, Carnegie Mellon), for both the production and transfer of knowledge and competencies, and for the impact on industrial innovation.

ARTIST2 has a double core, consisting of leading-edge research in embedded systems design issues (described later in this document) in the Joint Programme of Research Activities (JPRA), and complementary activities around shared platforms and staff mobility in the Joint Programme of Integration Activities (JPIA).

The JPRA activities are pure research, and the JPIA are complementary efforts for integration. Both work towards deep integration between the participating research teams.

The JPRA and JPIA are structured into clusters - one for each of the selected topics in embedded systems design (in red). Teams may be involved in one or several clusters.

Around this double core is the Joint Programme of Activities for Spreading Excellence (JPASE). These are complementary activiites for disseminating excellence across all available channels, targetting industry, students, and other European and international research teams.

Building the embedded systems design scientific community is an ambitious programme. To succeed, ARTIST2 will build on the achievements and experience from the ARTIST1 FP5 Accompanying Measure (http://www.artist-embedded.org/) on Advanced Real-Time Systems. ARTIST1 provided the opportunity to test the concept of a two-level integration (within and between clusters) four clusters in ARTIST2 originated as “actions” in ARTIST1. Building the ARTIST2 consortium and associated structure is the culmination of discussions and ambitions elaborated within ARTIST1.

ARTIST2 will address the full range of challenges related to Embedded Systems Design, covering all aspects, ranging from theory through to applications. In this way, ARTIST2 is perfectly in line with the IST priority on embedded systems, and in particular with the focus area called “system design”.

7.3. FAMILIES (ITEA Eureka)

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

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

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

ITEA projects ESAPS and CAFÉ have 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 standardization 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 standardization 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.

7.4. MOTOR (carroll)

Keywords: *MDA, OMG, QVT, model transformation*.

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

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.

7.5. MUTATION (carroll)

Keywords: *UML*, *methodology*, *requirements*, *test*.

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

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:

- formalization of system requirements
- 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; the underlying idea is to automatically generate tests cases that can be associated to the system requirements. It holds the following parts:

- formalization of system requirements
- 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 has already produced some results such as

- requirements formalization as a model with an associated textual syntax
- detailed conception formalization
- test objectives generation through some criteria
- prototypes that support the underlying technologies
- prototypes pre-evaluation on a real system provided by THALES

This project, continued in a MUTATION 2 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.

7.6. Amadeus

Keywords: *MDA, MDE, UML, methodology*.

Participants: Jean-Marc Jézéquel, Franck Chauvel, Didier Vojtisek.

Amadeus is a project supported by the PRIR of "Région Bretagne". It involves ENSTBr, ENSIETA, Université de Bretagne Sud and Inria/Triskell. This project aims at building links between research teams in Brittany working on Model Driven Engineering. Its main scientific objectives are:

- Study relationships between the notion of design by contract and model refinements,
- Study formal projections of UML to help model verifications,
- Apply a robust design and validation methodology to the UML

7.7. KEREVAL

Keywords: *components, diagnosis, extra functional , probes*.

Participants: Marouane Himdi, Yves Le Traon.

Development of Generic Probes for Functional and Extra-Functional Diagnosis

In addition to detection of errors related to design, coding or deployment of an application, the diagnosis is a well-known technique for understanding the behaviour of a software system and an absolute requirement for its improvement. Unfortunately, applications become more difficult to diagnosis as functionalities provided become complex. In collaboration with the KEREVAL company, we explore the use of dynamic probes (sensors) that will be injected into running system to collect various information. The innovative aspect of this approach is the use of generic probes to develop diagnosis framework [30].

8. Other Grants and Activities

8.1. National projects

8.1.1. CNRS action on Real Time Components

Participant: Noël Plouzeau.

The Real Time Components *action spécifique* aims at bringing together teams active in the domain of specification and implementation of software components for real time platforms and applications. The group has started its work in September, 2003 and will produce its report by the end of 2004. Inria Rennes, Loria, Ecole des Mines de Nantes, CEA and several other national research teams cooperate in this project, which is granted by the CNRS/STIC.

8.1.2. CNRS action on MDA

Participants: Jean-Marc Jézéquel, Noël Plouzeau, Benoit Baudry.

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 has produced a final report in October 2004 [43]. This project was granted by the CNRS/STIC.

8.1.3. CNRS action on Testability

Participant: Yves Le Traon.

The AS Testability working group involves the ONERA-CERT, the LSR-IMAG and LCIS-INPG laboratory. The goal is to make the state of the:

- practice from an industrial point of view, through a questionnaire sent to industrial partners;
- art from a scientific point of view, by organizing an international workshop on testability assesment (IWoTA see www.issre.org), sponsored by the IEEE Computer Society and the IEEE reliability Society.

The AS testability results were finalized by december 2004.

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.
- Triskell initiated a wiki dedicated to share information about the OMG within the INRIA (http://omg.wiki.irisa.fr/).

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 [54].
- Software engineering group (Pr. Keller's group), University of Montréal, Canada, on meta-modeling (H. Sahraoui).
- Carleton University, Ottawa, Canada: Triskell has developed a collaboration on test and objects with Lionel Briand's team at Carleton University.
- 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, and visited us again in november 2004.
- 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. IEEE Computer Society

Through the involvement of Jean-Marc Jézéquel in the IEEE TSE board and Yves Le Traon in the organization of the IEEE ISSRE, the Triskell project bears 66% of the total involvement of INRIA in the IEEE Computer Society (source: IEEE Computer Society 2004 Directory).

9.1.2. Journals

9.1.2.1. 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

9.1.3. Examination Committees

9.1.3.1. Jean-Marc Jézéquel

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

- Karine Macedo, May 2004, université de Rennes (adviser);
- Torben Weis, May 2004, Technical University Berlin (referee);
- Yves Le Traon (HDR), July 2004, université de Rennes (referee);
- Clémentine Nebut, November 2004, université de Rennes (adviser);
- Mikal Ziane (HDR), December 2004, université de Paris 6 (president);
- Michel Hurfin (HDR), November 2004, université de Rennes (referee);
- Nicolas Belloir, December 2004, université de Pau (referee) ;
- Tewfik Ziadi, December 2004, université de Rennes (adviser) ;

9.1.4. Examination Committees

9.1.4.1. Yves Le Traon

was in the examination committee of the following PhD thesis ”:

- Clémentine Nebut, November 2004, université de Rennes (adviser);

9.1.5. Conferences

9.1.5.1. Yves le Traon

was General Chair of the following conference:

- ISSRE 2004 (Software Reliability Engineering), St Malo, November 2004.

9.1.5.2. Jean-Marc Jézéquel

has been a member of the programme committee of the following conferences:

- ICSE’2004 : (International Conference on Software Engineering), Edinburgh, May 23-28, 2004.
- AFADL 2004 (Approches Formelles dans l’Assistance au Développement de Logiciels), LIFC, Besançon, June 16-18, 2004.
- MDAFA 2004 Workshop on Model Driven Architecture: Foundations and Applications, June 21-22, 2004, Linköping, Sweden.
- ASE 2004, (Automated Software Engineering), Linz, Austria, September 20-25, 2004

9.1.6. Workshops

J.-M. Jézéquel gave tutorials on “Model-Driven Engineering with Contracts, Patterns, and Aspects” at the “Ecole jeunes chercheurs en programmation” of the CNRS, as well as at the summer school on “MDE for Embedded System Development”, Brest September 2004. He also gave an invited talk at the 11th Rencontre INRIA-Industrie - "L’ingénierie du logiciel", INRIA - Rocquencourt, France, January 2004 [44]. He was also responsible of setting up and moderating a panel on “MDA in practice” at ICSE 2004.

J.-M. Jézéquel participated to two Dagsthul seminars in 2004:

- Dagstuhl Seminar 04101 on Language Engineering for Model-Driven Software Development, March 2004, Dagstuhl, Germany.
- Dagstuhl Seminar 04511 on Architecting Systems with Trustworthy Components, December 2004, Dagstuhl, Germany.

P.-A. Muller presented an invited conference on *The TopModL Initiative* at the Fujaba days 2004, Darmstadt, 16 septembre 2004. P.-A. Muller presented an invited conference on *Moving from general-purpose to domain-specific modelling languages*, FDL’04, Lille, 15 septembre 2004.

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 and MasterPro students.
Noël Plouzeau teaches OO Analysis and Design to DESS Isa (Ifsic).
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 [20].

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", vol. 13, n° 7, July 1999.

[2] C. JARD, J.-M. JÉZÉQUEL, A. L. GUENNEC, B. CAILLAUD. *Protocol Engineering using UML*, in "Annales des Telecoms", vol. 54, n° 11–12, November 1999, p. 526–538.

[3] J.-M. JÉZÉQUEL, D. DEVEAUX, Y. LETRAON. *Reliable Objects: a Lightweight Approach Applied to Java*, in "IEEE Software", vol. 18, n° 4, July/August 2001, p. 76–83.

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

[5] J.-M. JÉZÉQUEL. *Reifying Variants in Configuration Management*, in "ACM Transaction on Software Engineering and Methodology", vol. 8, nº 3, July 1999, p. 284–295.

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

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

[8] G. SUNYÉ, A. LEGUENNEC, J.-M. JÉZÉQUEL. *Using UML Action Semantics for Model Execution and Transformation*, in "Information Systems, Elsevier", vol. 27, nº 6, July 2002, p. 445–457.

[9] Y. L. TRAON, T. JÉRON, J.-M. JÉZÉQUEL, P. MOREL. *Efficient OO Integration and Regression Testing*, in "IEEE Trans. on Reliability", vol. 49, nº 1, March 2000, p. 12–25.

Doctoral dissertations and Habilitation theses

[10] K. MACEDO. *Modélisation d'aspects qualité de service en UML : application aux composants logiciels*, Ph. D. Thesis, Université de Rennes 1, May 2004.

[11] C. NEBUT. *Génération automatique de tests à partir des exigences et application aux lignes de produits logicielles*, Ph. D. Thesis, Université de Rennes 1, 2004.

[12] D. POLLET. *Architecture pour le restructuration de modèles*, Ph. D. Thesis, Université de Rennes 1, 2004.

[13] Y. L. TRAON. *Contribution au test de logiciels orientés-objet*, Ph. D. Thesis, Habilitation à diriger les recherches de l'université de Rennes I, July 2004.

[14] T. ZIADI. *Manipulation de lignes de produits en UML*, Ph. D. Thesis, Université de Rennes 1, 2004.

Articles in referred journals and book chapters

[15] F. BADEAU, D. BERT, S. BOULMÉ, C. MÉTAYER, M.-L. POTET, N. STOULS, L. VOISIN. *Traduction de B vers des langages de programmation*, in "TSI", 2004.

[16] B. BAUDRY, F. FLEUREY, J.-M. JÉZÉQUEL, Y. LETRAON. *Automatic Test Cases Optimization: a Bacteriologic Algorithm*, in "IEEE Software", 2004.

[17] B. BAUDRY, F. FLEUREY, J.-M. JÉZÉQUEL, Y. L. TRAON. *From Genetic to Bacteriological Algorithms for Mutation-Based Testing*, in "Software, Testing, Verification & Reliability journal (STVR)", 2004.

[18] B. BAUDRY, Y. L. TRAON. *Measuring Design Testability of a UML Class Diagram*, in "Information & Software Technology (IST)", December 2004.

[19] F. CHAUVEL, J.-M. JÉZÉQUEL, D. VOJTISEK. *Validation dynamique de modèles UML avec points de variation sémantique*, in "Génie Logiciel", nº 69, June 2004, p. 24–30.

[20] J.-M. JÉZÉQUEL, M. BELAUNDE, J. BÉZIVIN, S. GÉRARD, P.-A. MULLER. *L'ingéniérie piloté par les modèles*, collection Arrago, chap. Les concepts de l'ingéniérie des modèles, nº 30, OFTA, Paris, May 2004.

[21] C. NEBUT, Y. LETRAON, J.-M. JÉZÉQUEL. *System Testing of Product Families: from Requirements to Test Cases*, SPRINGERVERLAG (editor)., chap. WP4, Families Research Book, LNCS, 2004.

[22] D. VOJTISEK, J.-M. JÉZÉQUEL. *MTL and Umlaut NG - Engine and Framework for Model Transformation*, in "ERCIM News 58", vol. 58, July 2004.

[23] T. ZIADI, J.-M. JÉZÉQUEL. *Product Line Engineering with the UML: Products Derivation*, SPRINGERVERLAG (editor)., chap. WP4, Families Research Book, LNCS, 2004.

Publications in Conferences and Workshops

[24] B. BAUDRY, Y. L. TRAON, G. SUNYE. *Improving the Testability of UML Diagram*, in "Proc. of the 1st IEEE Int. Workshop on Testability Assesment (IWoTA 2004)", 2004.

[25] P. COLLET, D. DEVEAUX, R. ROUSSEAU, Y. L. TRAON. *Contract-Based Testing: from Objects to Components*, in "Proc. of the 1st IEEE Int. Workshop on Testability Assesment (IWoTA 2004)", 2004.

[26] O. DEFOUR, J.-M. JÉZÉQUEL, N. PLOUZEAU. *Applying CLP to Predict Extra-Functional Properties of Component-Based Models*, in "Proceedings of Logic Programming: 20th International Conference, ICLP 2004", J. S. DE BOER (editor)., LNCS, nº 3132, Springer Heidelberg, September 2004.

[27] O. DEFOUR, J.-M. JÉZÉQUEL, N. PLOUZEAU. *Extra-functional contract support in components*, in "Proc. of International Symposium on Component-based Software Engineering (CBSE7)", May 2004.

[28] F. FLEUREY, B. BAUDRY, Y. L. TRAON. *From testing to diagnosis: An automated approach*, in "Proc. 19th IEEE International Conference on Automated Software Engineering (ASE'04)", 2004.

[29] F. FLEUREY, J. STEEL, B. BAUDRY. *MDE and Validation: Testing Model Transformation*, in "Proc. of the SIVOES-Modeva workshop, SIVOES (Specification Implementation and Validation Of Embedded Systems)-MoDeVa (Model Design and Validation)", 2004.

[30] M. HIMDI. *Development of Generic Probes for Functional and Extra-Functional Diagnosis*, in "Supplementary proc. of the 15th IEEE International Symposium on Software Reliability Engineering (ISSRE 2004 Student Paper)", 2004.

[31] J.-M. JÉZÉQUEL, O. DEFOUR, N. PLOUZEAU. *An MDA Approach to Tame Component Based Software Development*, in "Post Proceedings of Formal Methods for Components and Objects (FMCO'03)", J. S. DE BOER (editor)., LNCS, nº 3188, Springer Heidelberg, 2004.

[32] J.-M. JÉZÉQUEL, W. EMMERICH. *Panel MDA in Practice*, in "26th International Conference on Software Engineering (ICSE 04), Edinburgh, UK", ACM, nº to be published, 2004.

[33] J. KLEIN, B. CAILLAUD, L. HÉLOUËT. *Merging scenarios*, in "9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS), Linz, Austria", sep 2004, p. 209–226.

[34] J. KLEIN, N. PLOUZEAU. *Transformation of behavioral models based on compositions of sequence diagrams*, in "Proceedings of Model-Driven Architecture: Foundations and Applications 2004 (MDAFA), Linkoping, Sweden", jun 2004, 255.

[35] D. LUGATO, F. MARAUX, Y. LE TRAON, C. N. V. NORMAND, H. DUBOIS, J.-Y. PIERRON, J.-P. GALLOIS. *Automated functional test case synthesis from thalès industrial requirements*, in "Proc. of the 10th IEEE Real-Time and embedded technology and Applications Symposium", 2004.

[36] P.-A. MULLER, D. BRESCH, P. STUDER. *Model-Driven Architecture for Automatic-Control*, in "Proc. of UML 2004", 2004.

[37] P.-A. MULLER, C. DUMOULIN, F. FONDEMENT, M. HASSENFORDER. *The TopModL Initiative*, in "3rd Workshop in Software Model Engineering (WiSME 2004) at UML2004", 2004.

[38] P.-A. MULLER, P. STUDER, J.-M. JÉZÉQUEL. *Model-driven generative approach for concrete syntax composition*, in "Proc. of OOPSLA Workshop on Best Practices for Model-Driven Development", 2004.

[39] S. PICKIN, J.-M. JÉZÉQUEL. *Using UML Sequence Diagrams as Basis for a Formal Test Description Language*, in "Proc. of Fourth International Conference on Integrated Formal Methods IFM2004, Canterbury, Kent, England", LNCS, n° to be published, Springer, April 2004, –.

[40] J. STEEL, M. LAWLEY. *An MDA Approach to Testing the Tarzan Model Transformation Engine*, in "Proceedings of ISSRE04 (International Conference on Software Reliability Engineering), St Malo, France", To appear, November 2004.

[41] Y. L. TRAON, B. BAUDRY. *Optimal Allocation of Testing Resources*, in "Proc. of the SIVOES-Modeva workshop, SIVOES (Specification Implementation and Validation Of Embedded Systems)-MoDeVa (Model Design and Validation)", 2004.

[42] T. ZIADI, L. HÉLOUËT, J.-M. JÉZÉQUEL. *Revisiting Statechart Synthesis with an Algebraic Approach*, in "26th International Conference on Software Engineering (ICSE 04), Edinburgh, UK", ACM, May 2004.

Miscellaneous

[43] J. BÉZIVIN, M. BLAY, M. BOUZHEGOUB, J. ESTUBLIER, J.-M. FAVRE, S. GÉRARD, J.-M. JÉZÉQUEL. *Rapport de Synthèse de l'AS CNRS sur le MDA*, November 2004, CNRS.

[44] J.-M. JÉZÉQUEL. *Perspectives on Model Driven Engineering*, January 2004, Invited presentation to the 11e Rencontre INRIA-Industrie - "L'ingénierie du logiciel", INRIA - Rocquencourt, France.

Bibliography in notes

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

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

[47] A. BEUGNARD, J.-M. JÉZÉQUEL, N. PLOUZEAU, D. WATKINS. *Making Components Contract Aware*, in "IEEE Computer", vol. 13, n° 7, July 1999.

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

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

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

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

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

[53] J.-M. JÉZÉQUEL, B. MEYER. *Design by Contract: The Lessons of Ariane*, in "Computer", vol. 30, n° 1, January 1997, p. 129–130.

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

[55] B. MEYER. *Reusability: The Case for Object-Oriented Design*, in "IEEE SOFTWARE", n° 3, March 1987, p. 50–64.

[56] B. MEYER. *Applying "Design by Contract"*, in "IEEE Computer (Special Issue on Inheritance & Classification)", vol. 25, n° 10, October 1992, p. 40–52.

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

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

[59] 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, Bad Honnef", G. R. W. BRAUER (editor)., vol. 255, Springer-Verlag, september 1986, p. 325-392.