INRIA

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
Noël Plouzeau [Assistant Professor Université de Rennes 1]

Visiting Scientist
Pierre-Alain Muller [Assistant Professor Université de Mulhouse]
Antoine Beugnard [Assistant Professor ENST de Bretagne]
Yves Le Traon [France telecom R&D]

Post-doctoral Fellow
Olivier Defour [until February 2005]
Olivier Barais [from December 2005]

Technical staff
Zoé Drey [Inria Associated Engineer]
Erwan Drezen [Inria (Project Carroll/Motor Carroll/Mutation) from June 2003 to March 2005]
Jean-Philippe Thibault [Inria (project RNTL ACCORD) since october 2002 until June 2005, (project FAMILIES) from October 2003 to June 2005]

Lecturer
Tewfik Ziadi [until September 2005]
Damien Pollet [since September 2005]

PhD Students
Erwan Brottier [CIFRE grant since November 2005]
Franck Chauvel [Brittany Council grant]
Franck Fleurey [MENRT grant]
Marouane Himdi [CIFRE grant]
Jacques Klein [INRIA grant]
Christophe Métayer [CIFRE grant]
Martin Monperus [DGA grant since October 2005]
Jean-Marie Mottu [MENRT grant since October 2005]
Sébastien Saudrais [INRIA grant]
Jim Steel [INRIA grant]

2. Overall Objectives

2.1. 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 (a.k.a. *product lines*) 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. Since these modeling languages and associated tools appear quite open-ended and very domain specific, there is a growing need for "tools for building tools for building software". Triskell is hence developping KerMeta as an original meta-meta modeling approach allowing the user to fully define his modeling languages (including dynamic semantics) and associated environments (including interpreters, compilers, importers/exporters, etc.) within Eclipse.

To avoid the pitfall of developping "tools for building tools for the sake of it", the Triskell project also has the goal of explicitly connecting research results to industrial problems through technology transfer actions. This implies, in particular, taking into account the industrial standards of the field, namely the Eclipse Modeling Framework EMF, the OMG's Meta-Object Facility MOF and Unified Modeling Language 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.; separation of concerns with aspects, 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 [47], [44]: 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 [50].

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 [42]. 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 concept also provides the bases needed to develop *software components*, for which Szyperski's definition [52] 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 [48]).

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 [51] 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) [53]. 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) [43].

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.*

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 [45] 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:

  1. processes are sequential : two events executed by the same process are causally ordered.
  2. 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* [54] 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, iether at requirement level or at design level.

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. Kermeta : Kernel Metamodeling

Keywords: *MDA, MOF, UML, model transformation.*

Participants: Franck Chauvel, Zoé Drey, Franck Fleurey, Jean-Marc Jézéquel, Pierre-Alain Muller, Jean-Philippe Thibault, Didier Vojtisek [correspondant].

Nowadays, object-oriented meta-languages such as MOF (Meta-Object Facility) are increasingly used to specify domain-specific languages in the model-driven engineering community. However, these meta-languages focus on structural specifications and have no built-in support for specifications of operational semantics. Triskell has developped the Kermeta language to explore the idea of using aspect-oriented modeling to add precise action specifications with static type checking and genericity at the meta level, and examine related issues and possible solutions [34].

Kermeta consists of an extension to the Essential Meta-Object Facilities (EMOF) 2.0 to support behavior definition. It provides an action language to specify the body of operations in metamodels. This action language is imperative and object-oriented.

Kermeta is used in several use cases:

- Kermeta give a precise semantic of the behavior of a metamodel which then can be simulated.
- Kermeta can be used as a model transformation language.
- Kermeta can be used as a constraint language.

The development environment built for the Kermeta language currently contains the following tools

- an interpreter that allows ametamodel to be executed.
- a texteditor, fully integrated within Eclipse, with syntax higlighting, code autocompletion.
- an Eclipse outline view, which allows navigation through the whole model and metamodel.
- various import/export transformations such as ecore2kermeta (kermeta text), kermat2ecore, kermeta2xmi (xmi version of your kermeta metamodel), xmi2kermeta, xmi2ecore.

Kermeta is one of the corner stone of UMLAUT NG. However, since it will be distributed as an Eclipse project (in the Eclipse GMT project), it has its own external visibility. Developped as an open source software under the terms of the EPL (Eclipse Public License), it has been first deposited to the APP (Agence de Protection des Programmes) in October 2005.

5.2. 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.

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 2005, UMLAUT NG was used within these projects in collaboration with industry:

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.

5.3. 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.

5.4. Requested : a toolbox for requirement simulation and testing

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

Participants: Yves Le Traon [correspondant], 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. Model Driven Engineering for Distributed Real Time Embedded Systems

Participants: Jean-Marc Jézéquel, Pierre-Alain Muller, Christophe Métayer.

In domains such as automotive or avionics, real-time and embedded systems are getting ever more software intensive. The software cannot any longer be produced as a single chunk, and engineers are contemplating the possibility of componentizing it along the lines presented in Section 3.2. In this vision, any composite application is viewed as a particular configuration of components, selected at build-time and configured or re-configured at run-time. A software component only exhibits its provided or required interfaces. This defines basic contracts between components allowing one to properly wire them.

In real-time and embedded systems however, we have to take into account many extra-functional aspects, such as timeliness, memory consumption, power dissipation, reliability, performances, and generally speaking Quality of Service (QoS). These aspects can also be seen as contracts between the system, its environment and its users. These contracts must obviously be propagated down to the component level. One of the key desiderata in component-based development for embedded systems is thus the ability to capture both functional and extra-functional properties in component contracts, and to verify and predict corresponding system properties.

A contract is in practice taken to be a constraint on a given aspect of the interaction between a component that supplies a service, and a component that consumes this service. Component contracts differ from object contracts in the sense that to supply a service, a component often explicitly requires some other service, with its own contract, from another component. So the expression of a contract on a component-provided interface might depend on another contract from one of the component-required interfaces. For instance, the throughput of a component A doing some kind of computation on a data stream provided by component B clearly depends on the throughput of B.

It is then natural that people resort to modelling to try to master this complexity [32]. Since models of software have the same nature as the thing they models, this opens the possibility to automatically derive software from its model. This was recently be made quite popular with an OMG initiative called the Model Driven Architecture (MDA). The aim of this work is to show how MDA can be used in relation with real-time and embedded component based software engineering [18]. Building on Model Driven Engineering techniques, we show how the very same contracts expressed in a UML model can be exploited for (1) validation of individual components, by automatically weaving contract monitoring code into the components; and (2) 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.

In [32] we report on our experience with a model-driven architecture for distributed and embedded process-control based on the assembly of pre-defined components implemented for low-cost micro-controlers.

6.1.2. Model Composition and Weaving

Participants: Benoit Baudry, Franck Fleurey.

The aspect oriented modeling (AOM) approach provides mechanisms for separating crosscutting functionality from core functionality in design models. Crosscutting functionality is described by aspect models and the core application functionality is described by a primary model. The integrated system view is obtained by composing the primary and aspect models. In [39], we present a model composition technique that relies on signature matching: A model element is merged with another if their signatures match. A signature consists of some or all properties of an element as defined in the UML metamodel. The technique proposed in this paper is capable of detecting some conflicts that can arise during composition.

To implement the composition algorithm we have chosen to use our Kermeta language [34]. First, the language allows implementing composition by adding the algorithm in the body of the operations defined in the composition metamodel. Second, KerMeta tools are compatible with the Eclipse Modeling Framework (EMF) which allows us to use Eclipse tools to edit, store, and visualize models.

The apparent similarities between model composition and model transformations often lead to the following question: Is model composition a special type of model transformation? Answering this question can lead to useful insights that can be used to develop technologies that leverage the relationship between composition and transformation. In [25] we show that there are a number of ways to implement composition as transformations. We give an overview of these approaches in terms of their generality, ease of use, and ease of implementation. The insights gained from our initial analysis suggest that one can implement model composition as model transformations. This is important in that it indicates that research on model composition can leverage research on model transformations.

This work is done in collaboration with Robert France, Sudipto Ghosh and Raghu Reddy from Colorado State University (CSU). It was ignitiated during a visit from Benoit Baudry and Franck Fleurey to CSU. This visit was supported by INRIA and by the ARTIST network of excellence.

6.1.3. Weaving Behavioural Models

Participants: Jacques Klein, Jean-Marc Jézéquel, Noel Plouzeau.

Languages for aspect-oriented programming (AOP), such as AspectJ, are now popular, and the concepts used by the AOP community such as join points, pointcuts and advice are well-known. At the same time, in recent years, the aspect oriented software development (AOSD) approach has been developing itself beyond the programming activity. More particularly, the Early Aspects Initiative advocates the management

of crosscutting properties, i.e. aspects, at the early development stages of requirements engineering and architecture design to identify the impact of aspects as soon as possible. Some composition operators of aspects exist for these development stages, but they do not closely match standard AOP concepts (pointcuts, advice...). In [29], we propose an automatic way for weaving behavioural aspects given as scenarios. With these kinds of behavioural modelling languages, aspect weaving cannot always be performed at the abstract syntax level. In [28], we present the problems relating to the design of a semantic based aspect weaver for Hierarchical Message Sequence Charts (HMSCs).

6.1.4. Timed-based contracts for components

Participants: Pierre-Alain Muller, Noël Plouzeau, Sébastien Saudrais.

In many application domains, contract-based specification for software components need to take time properties as well as other so called *extra-functional* properties (e.g. throughput, bandwidth consumption, etc).

[38] is a rigorous and automated approach for the behavioral validation of control software systems, based on metamodeling, model-transformations and process algebra. The work combines semi-formal object-oriented models with formal validation.

Triskell has also designed in the past three years a language for specifiying extra-functional properties on software components. This language allows the definition of extra-functional dependencies between required services and provided services.

Since the beginning of 2005, our team is working on a formal definition of this language. This formalization effort aims at supporting extra-functional property specification and verification through a complete chain of model transformation and validation that implements model driven engineering processes. The major difficulty to be tackled is the correct transformation of formal properties bound to the successive software models generated during the model transformation. The target platforms are real-time runtimes and frameworks such as the Giotto runtime model. The work also aims at maximing the compatibility with existing tool chain items from external partners.

This work is done in cooperation with the Jacquard project team at LIFL and with partners of the Artist2 network of excellence.

6.2. Model-Based Testing

6.2.1. Validation in a MDE context : Models and Model Transformations testing

Participants: Benoit Baudry, Franck Fleurey, Jim Steel, Jean-Marie Mottu, Yves le Traon.

Model-driven software development techniques raise the level of abstraction at which developers conceive and implement complex software systems. To ensure that the developed systems are of high quality, it is important for developers to evaluate the quality of the system models and the model transformations which manipulate them.

A systematic study of faults that occur in UML designs will help us develop better design evaluation techniques. While there is a large body of literature on fault models for system implementations (code), there is a lack of research on fault models in designs. In many cases, researchers reverse engineer the code to obtain views of the design. Our goal is to develop fault models for UML designs at various levels of abstraction, not just models of the code. There will be some overlap in the fault models for designs and code. However, there is a completely new set of fault types that arise from the use of different UML diagram types that can be used to represent different views of a system. In [27] we present a taxonomy of faults that occur in UML designs. We also describe a set of mutation operators for UML class diagrams. This work is part of a collaboration with Trung Dinh-Trong, Sudipto Gosh and Robert France from the Computer Science Department of the Colorado State University.

Once we can trust the models used, transformations allow automatic manipulations of these. It's an important feature for the models reuse. Thus, it is necessary to propose efficient techniques to validate the transformation programs. For every program, test process is composed of three steps: the test data generation,

their execution and the validation of the results produced by the oracle. If the execution has no specificities, the two other features depend on the program tested. With transformation programs, we need dedicated techniques because the data are particularly complex: the models. In [31], we study original solutions for model transformation testing. We introduce criteria to generate test data. We discuss an adaptation of the mutation testing which allows to evaluate the efficiency of the data generated. Finally, different issues for the oracle feature are analyzed. This work is part of a collaboration with Erwan Brottier from the Research and Development department of France Telecom.

6.3. Model-Driven Engineering

6.3.1. Weaving Executability into Object-Oriented Meta-Languages

Participants: Pierre-Alain Muller, Franck Fleurey, Jean-Marc Jézéquel.

Nowadays, object-oriented meta-languages such as MOF (Meta-Object Facility) are increasingly used to specify domain-specific languages in the model-driven engineering community. However, these meta-languages focus on structural specifications and have no built-in support for specifications of operational semantics. In [34] we explore the idea of using aspect-oriented modeling to add precise action specifications with static type checking and genericity at the meta level, and examine related issues and possible solutions. We believe that such a combination would bring significant benefits to the community, such as the specification, simulation and testing of operational semantics of metamodels. We present requirements for such statically-typed meta-languages and rationales for the aforementioned benefits.

6.3.2. Model Transformations

Participants: Jean-Marc Jézéquel, Pierre-Alain Muller, Franck Fleurey, Zoé Drey, Damien Pollet, Didier Vojtisek, 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 independently of any underlying technology, through model transformation techniques [35].

Domain specific languages for model transformation have recently generated significant interest in the model-driven engineering community. A number of approaches have been presented to model transformation, from graph-transformation-based techniques, to rule-based languages, to imperative languages. Each of these offer comparative advantages for different classes of model transformation problems.

One such approach is Tefkat, an implementation of a language designed specifically for the transformation of MOF models using patterns and rules. The language adopts a declarative paradigm, wherein users may concern themselves solely with the relations between the models rather than needing to deal explicitly with issues such as order of rule execution and pattern searching/traversal of input models. In [30], the language and its implementation are demonstrated using a provided example and highlight a number of language features used in solving the problem, a simple object-to-relational mapping.

Another approach is presented in [14] as an architecture for manipulating models which is independent 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.

This approach is based on a MOF-level interface 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.

Thus, while the adopted QVT specification has normalized some scheme of model transformation language; several different model transformation language paradigms are likely to co-exist in the near future, ranging from imperative to declarative (including hybrid). It remains nevertheless questionable how model transformation specific languages compare to more general purpose languages, in terms of applicability, scalability and robustness. [20] is a general overview of the different model transformation techniques, including concept and terminology presentation. In [35] we report on our specific experience in applying an executable meta-language such as Kermeta to the model transformation field.

6.3.3. Model Typing for Improving Reuse in Model-Driven Engineering

Participants: Jim Steel, Jean-Marc Jézéquel.

Where object-oriented languages deal with objects as described by classes, model-driven development uses models, as graphs of interconnected objects, described by metamodels. A number of new modeling languages have been and continue to be developed for this model-based paradigm, both for model transformation and for general programming using models. Many of these use single-object approaches to typing, derived from solutions found in object-oriented systems, while others use metamodels as model types, but without a clear notion of polymorphism. Both of these approaches lead to brittle and overly restrictive reuse characteristics.

The contribution presented in [40] presents a simple extension to object-oriented typing to better cater for a model-oriented context, including a simple strategy for typing models as a collection of interconnected objects. Using a simple example it is shown how this extended approach permits more flexible reuse, while preserving type safety.

Going forward, the presence of a well-founded type system for models will allow developers to reason about how various model-driven engineering artifacts interconnect, from models and metamodels to model transformations and programs, to repository and modelling tools.

6.3.4. Code Generation from UML Models with Semantic Variation Points

Participants: Franck Chauvel, Jean-Marc Jézéquel.

UML semantic variation points provide intentional degrees of freedom for the interpretation of the metamodel semantics. The interest of semantic variation points is that UML now becomes a family of languages sharing lot of commonalities and some variabilities that one can customize for a given application domain. This works [26] propose to reify the various semantic variation points of UML 2.0 statecharts into models of their own to avoid hardcoding the semantic choices in the tools. We do the same for various implementation choices. Then, along the line of the OMG's Model Driven Architecture, these semantic and implementation models are processed along with a source UML model (that can be seen as a PIM) to provide a target UML model (a PSM) where all semantic and implementation choice are made explicit. This target model can in turn serve as a basis for a consistent use of code generation, simulation, model-checking or test generation tools.

6.3.5. Bridges between Models and Text/Hypertext

Participants: Jim Steel, Pierre-Alain Muller, Jean-Marc Jézéquel.

In [36] we use HUTN as a bridge between ModelWare and GrammarWare, to generate parsers and editors for DSLs defined under the shape of metamodels. We describe the problems that we have encountered with the ambiguities of the current HUTN specification and discuss how this specification may be fixed to be usable with grammar-driven tools.

[21] is about platform independent Web application modeling and development in the context of model-driven engineering. Web applications are represented via three independent but related models (business, hypertext and presentation). A kind of action language (based on OCL and Java) is used all over these models to write methods and actions, specify constraints and express conditions.

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, Olivier Barais, 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.(http://www.aosd-europe.net/)

Project duration: 2004-2008

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, Pierre-Alain Muller, Benoit Baudry, Didier Vojtisek.

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. Artist2 gathers 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 is implementing 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 addresses 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”.

The Triskell team is taking part in two Artist2 clusters: the *Modelling and Components* cluster (led by Bengt Jonsson, at Uppssala university, Sweden) and the Adaptive Real Time Middleware (led by Giorgo Buttazzo, Italy).

The current cooperation topics within the Components cluster are the use of various formalisms for timed behaviour descriptions, the definition of an architecture for interconnecting simulation and verification platforms for these behaviours. Within the Adaptive Real Time cluster, Triskell is participating in the common definition of quality of service dictionary, in the context of middleware runtimes.

7.3. FAMILIES (ITEA Eureka)

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

Participants: Jean-Marc Jézéquel, Yves Le Traon, Jacques Klein, 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. For this work package, Triskell has organized SPLC-EUROPE 2005 (9th International Software Product Line Conference), Rennes, September 2005, which attracted over 130 persons from all over the world, including 70 people from industry.

7.4. MUTATION 2 (carroll)

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

Participants: Benoit Baudry, Erwan Drézen, Franck Fleurey, Yves Le Traon, Didier Vojtisek.

MUTATION 2 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 2 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 2 are:

- defining rules in order to formalize requirements,
- defining rules in order to formalize the detailed software design,
- automatic test generation from a requirements model,
- providing a low cost training for an industrial team.

The three successive parts of MUTATION 2 are:

- definition of a language dedicated to the description of requirements and its associated methodology; a user guide and some examples will also to be written,
- defining coverage critera and identofying necessray UML extensions,
- applying the proposed concepts on a real case provided by THALES; it should allow to evaluate the improvement in terms of productivity.

This project has already produced some results such as

- requirements formalization as a model with an associated textual syntax (called Requirements Description Language)
- detailed design formalization,
- test objectives generation according to several criteria,

- prototypes that support the underlying technologies,
- prototypes pre-evaluation on a real system provided by THALES.

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

7.5. MDE Standards for Aerospace (carroll)

Keywords: *AADL*, *MDA*, *OMG*, *UML*, *methodology*.

Participants: Jean-Marc Jézéquel, Benoit Baudry, Pierre-Alain Muller, Didier Vojtisek.

"MDE Standards for Aerospace" is a lightweight project developed by CEA/LIST (LLSP), THALES Research and Technology, and INRIA acting as an expert group for CNES. This project aims at assessing the suitability for the Aerospace domain of Model-Driven Engineering standards. It started in March 2005 and was completed in November 2005.

7.6. Amadeus

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

Participants: Jean-Marc Jézéquel, Pierre-Alain Muller, Antoine Beugnard, 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.

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 [3].

8. Other Grants and Activities

8.1. International working groups

8.1.1. ERCIM Working Group on Software Evolution

Numerous scientific studies of large-scale software systems have shown that the bulk of the total software-development cost is devoted to software maintenance. This is mainly due to the fact that software systems need to evolve continually to cope with ever-changing software requirements. Today, this is more than ever the case. Nevertheless, existing tools that try to provide support for evolution have many limitations. They are (programming) language dependent, not scalable, difficult to integrate with other tools, and they lack formal foundations.

The main goal of the proposed WG (http://w3.umh.ac.be/evol/) is to identify a set of formally-founded techniques and associated tools to support software developers with the common problems they encounter when evolving large and complex software systems. With this initiative, we plan to become a Virtual European Research and Training Centre on Software Evolution.

Triskell contributes to this working group on the following points:

- re-engineering and reverse engineering
- model-driven software engineering and model transformation
- impact analysis, effort estimation, cost prediction, evolution metrics
- traceability analysis and change propagation
- family and product-line engineering

8.1.2. 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 participates to the new Executable UML foundation RFP. This RFP will standardize a subset of UML2.0 with a more precise execution semantic.
- 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/).

8.1.3. Collaboration with foreign research groups:

- University of Oslo, Norway. Collaboration on the SWAT project (Semantics-preserving Weaving - Advancing the Technology) with Øystein Haugen and Birger Møller-Pedersen. The goal of this formal collaboration is to identify basic mechanisms behind the mechanisms we find in generics, aspect orientation, family modeling and generative programming, in general what mechanisms we should have in order to produce models/programs from generic models/programs or from fragments of models/programs.
- Colorado State University (CSU), USA. Collaboration on several issues related to model-driven development with Robert France and Sudipto Ghosh. More precisely we have collaborated on model composition for aspect-oriented modelling, model transformation and model validation with testing. Franck Fleurey and Benoit Baudry visited CSU in summer 2005, this visit was funded by INRIA as part of the “mini prjet INRIA” program. Robert France visited Triskell in 2003, will come back in June 2006 and should spend 6 months in sabbatical in 2006-07. To institutionalize our collaboration, we have set up a “Equipe associée” (associated team) called MATT between CSU and Triskell on Model-driven engineering: Aspects, Transformations and Test (see http://www.irisa.fr/triskell/matt for details).
- 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 [49].

- Software engineering 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.B. Meyer's team), Switzerland on Trusted Components. B. Meyer came to Rennes several times in the past few years.
- Distributed Systems Technology Centre, Brisbane, Australia. Triskell has collaborated and published with the Pegamento team on rule-based approaches to model transformation.

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

9.1.1.2. Pierre-Alain Muller

is an Associate Editor of the following journals:

- Journal on Software and System Modeling: SoSyM

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":

- Adnan Bader, May 2005, Monash University (referee);
- Arnaud Thiefaine, June 2005, université de Paris 6 (referee);
- Damien Pollet, June 2005, université de Rennes (adviser);
- Jean-Charles Tournier, July 2005, INSA Lyon (referee);
- Olivier Barais, November 2005, université de Lille (referee);
- Arnaud Cuccuru, November 2005, université de Lille (referee);
- German Vega, December 2005, université de Grenoble (referee) ;
- Selma Matougui, December 2005, université de Rennes (adviser);

9.1.2.2. Yves Le traon

was in the examination committee of the following PhD thesis:

- Yves Souchard, November 2005, LEG-INPG (referee);
- Céline Bigot, June 2005, CNAM-CEA (referee);

9.1.2.3. Pierre-Alain Muller

was in the examination committee of the following PhD thesis:

- Damien Pollet, June 2005, université de Rennes.

9.1.3. Conferences

9.1.3.1. Jean-Marc Jézéquel

was Conference Chair of the following conference:

- SPLC-EUROPE 2005 (9th International Software Product Line Conference), Rennes, September 2005. This conference attracted over 130 persons from all over the world, including 70 people from industry.

9.1.3.2. Jean-Marc Jézéquel

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

- ECOOP 2005 The 19th European Conference on Object Oriented Programming, Scottish Exhibition and Conference Centre, Glasgow, Scotland, 25-29 July 2005
- Special session: Model Driven Engineering of 31th EUROMICRO Conference, Porto, Portugal, August 30st - September 3rd, 2005
- MODELS 2005 The 8th International Conference on Model Driven Engineering Languages and Systems (formerly the UML series of conferences) Jamaica, 2-7 October 2005
- ECMDA-FA'05 European Conference on Model Driven Architecture - Foundations and Applications, November 7-10, 2005, Nurenberg, Germany
- QoSA 2005 First International Conference on the Quality of Software Architectures, Net.ObjectDays, September, 20-22, 2005, Erfurt, Germany
- JC 2005 4eme Conférence Francophone autour des Composants Logiciels, Le Croisic, France, 7 et 8 avril 2005

9.1.3.3. Yves Le Traon

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

- ISSRE 2005- the 16th IEEE International Symposium on Software Reliability Engineering, Chicago, November 2005
- 2nd int. workshop on Model design and Validation (MoDeVa 2005), Jamaica, October 2005
- 1ères journéees sur l'Ingénierie Dirigées par les Modèles (IDM 05), Paris, June 2005

9.1.3.4. Noël Plouzeau

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

- 2ème Journée Francophone sur le Développement de Logiciels Par Aspects (JFDLPA 2005), Septembre 2005

9.1.3.5. Pierre-Alain Muller

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

- MODELS 2005 The 8th International Conference on Model Driven Engineering Languages and Systems (formerly the UML series of conferences) Jamaica, 2-7 October 2005
- Special session: Model Driven Engineering of 31th EUROMICRO Conference, Porto, Portugal, August 30st - September 3rd, 2005
- VL/HCC'05, IEEE Symposium on Visual Languages and Human-Centric Computing, Dalas, Texas, USA, 20-24 September 2005

9.1.4. Workshops

J.-M. Jézéquel gave an invited talk at the DGA seminar on "Complex Software Systems engineering", November 2005.

P.-A. Muller was Programme Chair of the 1ères journéees sur l'Ingénierie Dirigées par les Modèles (IDM 05), Paris, June 2005 [12]. He presented an invited conference on *Model Transformations* at the Artist2 summer school, septembre 2005. He was also responsible of setting up and moderating a panel on "Meta-modeling architecture" at MODELS'2005, and another one on "Model-Driven Engineering" at LMO'05.

B. Baudry was co-organizer with Christophe Gaston (CEA) and Sudipto Ghosh (CSU) of the MoDeVa workshop in conjunction with MoDELS'05. He was also Workshop Chair for SPLC-EUROPE 2005 (9th International Software Product Line Conference), Rennes, September 2005.

9.2. Teaching

Jean-Marc Jézéquel teaches OO Analysis and Design with UML (Iup3 and Diic2) at Ifsic [13], 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 and Component Based Design to students of the first and second year of Mastère Sciences et Technologies, mention Informatique (Ifsic).

Benoit Baudry teaches software testing (Iup3, Diic3, Master1).

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

9.3. Miscellaneous

- J.-M. Jézéquel is a member of the Steering Committee of the MODELS/UML Conferences series. He is appointed to the board of the Committee of Projects of INRIA Rennes.
- P.-A. Muller is a member of the Steering Committee of the MODELS/UML Conferences series. He holds an elected position at the scientific council of the Université de Haute Alsace.

- J.-M. Jézéquel is the writer of the chapter on Design Patterns in the *Encyclopédie de l'informatique* to be published by Vuibert in the second half of 2005 [19].
- N. Plouzeau is the writer of the chapter on Software Components in the encyclopedia mentioned above [23].

10. Bibliography

Major publications by the team in recent years

[1] B. BAUDRY, F. FLEUREY, J.-M. JÉZÉQUEL, Y. LE TRAON. *Automatic Test Cases Optimization: a Bacteriologic Algorithm*, in "IEEE Software", vol. 22, n° 2, March 2005, p. 76–82.

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

[3] 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), St Malo", November 2004.

[4] 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.

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

[6] J.-M. JÉZÉQUEL. *Model Driven Engineering for Distributed Real Time Embedded Systems*, S. GÉRARD, J.-P. BABAU (editors). , chap. Real Time Components and Contracts, Hermes Science Publishing Ltd, London, 2005.

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

[8] J.-M. JÉZÉQUEL. *Reifying Variants in Configuration Management*, in "ACM Transaction on Software Engineering and Methodology", vol. 8, n° 3, July 1999, p. 284–295.

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

[10] G. SUNYÉ, A. L. GUENNEC, J.-M. JÉZÉQUEL. *Using UML Action Semantics for Model Execution and Transformation*, in "Information Systems, Elsevier", vol. 27, n° 6, July 2002, p. 445–457.

[11] Y. L. TRAON, T. JÉRON, J.-M. JÉZÉQUEL, P. MOREL. *Efficient OO Integration and Regression Testing*, in "IEEE Trans. on Reliability", vol. 49, n° 1, March 2000, p. 12–25.

Books and Monographs

[12] S. GÉRARD, J.-M. FAVRE, P.-A. MULLER, X. BLANC (editors). *IDM05, Actes des 1ères Journées sur l'Ingénierie Dirigée par les Modèles*, n° ISBN 2-7261-1284-6, June 2005, http://planetmde.org/idm05/actes.pdf.

[13] J.-M. JÉZÉQUEL, N. PLOUZEAU, Y. L. TRAON. *Développement de logiciel à objets avec UML*, September 2005, Polycopié IFSIC C119, version 1.6, 146 pages.

Doctoral dissertations and Habilitation theses

[14] D. POLLET. *Une architecture pour les transformations de modèles et la restructuration de modèles UML*, Ph. D. Thesis, Université de Rennes 1, June 2005, http://www.irisa.fr/bibli/publi/theses/2005/pollet/pollet.html.

Articles in refereed journals and book chapters

[15] B. BAUDRY, F. FLEUREY, J.-M. JÉZÉQUEL, Y. LE TRAON. *Automatic Test Cases Optimization: a Bacteriologic Algorithm*, in "IEEE Software", vol. 22, n° 2, March 2005, p. 76–82.

[16] 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)", vol. 15, n° 2, June 2005, p. 73-96.

[17] J. BAYER, S. GÉRARD, O. HAUGEN, J. MANSELL, B. MOLLER-PEDERSEN, J. OLDEVIK, P. TESSIER, J.-P. THIBAULT, T. WIDEN. *Families Research Book*, LNCS, chap. A Unified Conceptual Model for Product Family Variability Modelling, n° to be published, Springer Verlag, 2005.

[18] J.-M. JÉZÉQUEL. *Model Driven Engineering for Distributed Real Time Embedded Systems*, S. GÉRARD, J.-P. BABAU (editors). , chap. Real Time Components and Contracts, Hermes Science Publishing Ltd, London, 2005.

[19] J.-M. JÉZÉQUEL. *Encyclopédie Vuibert de l'informatique*, chap. Patrons de conception, Vuibert, 2005.

[20] P.-A. MULLER. *Model Driven Engineering for Distributed Real Time Embedded Systems*, S. GÉRARD, J.-P. BABAU, J. CHAMPEAU (editors). , chap. Model Transformations, Hermes Science Publishing Ltd, London, 2005.

[21] P.-A. MULLER, P. STUDER, F. FONDEMENT, J. BEZIVIN. *Independent Web Application Modeling and Development*, in "Software and System Modeling", vol. 4, n° 4, November 2005, p. 424–442.

[22] C. NEBUT, Y. LE TRAON, J.-M. JÉZÉQUEL. *Families Research Book*, LNCS, chap. System Testing of Product Families: from Requirements to Test Cases, n° to be published, Springer Verlag, 2005.

[23] N. PLOUZEAU. *Encyclopédie Vuibert de l'informatique*, chap. Composants logiciels, Vuibert, 2005.

[24] T. ZIADI, J.-M. JÉZÉQUEL. *Families Research Book*, LNCS, chap. Product Line Engineering with the UML: Products Derivation, n° to be published, Springer Verlag, 2005.

Publications in Conferences and Workshops

[25] B. BAUDRY, F. FLEUREY, R. FRANCE, R. REDDY. *Exploring the Relationship between Model Composition and Model Transformation*, in "Aspect Oriented Modeling (AOM) Workshop, Montego Bay, Jamaica", October 2005.

[26] F. CHAUVEL, J.-M. JÉZÉQUEL. *Code Generation from UML Models with Semantic Variation Points*, in "Proceedings of MODELS/UML'2005, Montego Bay, Jamaica", L. BRIAND, S. KENT (editors). , LNCS, vol. to be published, Springer, October 2005.

[27] T. DINH-TRONG, S. GHOSH, R. FRANCE, B. BAUDRY, F. FLEUREY. *A Taxonomy of Faults for UML Designs*, in "Model Design and Validation (MoDeVa) Workshop, Montego Bay, Jamaica", October 2005.

[28] J. KLEIN, J.-M. JÉZÉQUEL. *Problems of the Semantic-based Weaving of Scenarios*, in "In Aspects and Software Product Lines: An Early Aspects Workshop at SPLC-Europe 05, Rennes", September 2005.

[29] J. KLEIN, J.-M. JÉZÉQUEL, N. PLOUZEAU. *Weaving Behavioural Models*, in "In First Workshop on Models and Aspects, Handling Crosscutting Concerns in MDSD at ECOOP 05, Glasgow", July 2005.

[30] M. LAWLEY, J. STEEL. *Practical Declarative Model Transformation With Tefkat*, in "Model Transformations In Practice Workshop, Montego Bay, Jamaica", October 2005.

[31] J.-M. MOTTU, B. BAUDRY, Y. LE TRAON, E. BROTTIER. *Génération Automatique de Test pour les Transformations de Modèles*, in "1ère Journées sur l'Ingénierie Dirigée par les Modèles, Paris", June 2005.

[32] P.-A. MULLER, D. BRESCH. *Model-Driven Architecture for Distributed and Embedded Process-Control*, in "5ème Colloque sur l'Enseignement des Technologies et des Sciences de l'Information et des Systèmes (CETSIS), Nancy", October 2005.

[33] P.-A. MULLER, C. DUMOULIN, F. FONDEMENT, M. HASSENFORDER. *The TopModL Initiative*, in "UML 2004 Satellite Activities, Montego Bay, Jamaica", March 2005.

[34] P.-A. MULLER, F. FLEUREY, J.-M. JÉZÉQUEL. *Weaving Executability intoObject-Oriented Meta-Languages*, in "Proceedings of MODELS/UML'2005, Montego Bay, Jamaica", L. BRIAND, S. KENT (editors). , LNCS, vol. to be published, Springer, October 2005.

[35] P.-A. MULLER, F. FLEUREY, D. VOJTISEK, Z. DREY, D. POLLET, F. FONDEMENT, P. STUDER, J.-M. JÉZÉQUEL. *On Executable Meta-Languages applied to Model Transformations*, in "Model Transformations In Practice Workshop, Montego Bay, Jamaica", October 2005.

[36] P.-A. MULLER, M. HASSENFORDER. *HUTN as a Bridge between ModelWare and GrammarWare*, in "WISME Workshop, MODELS / UML'2005, Montego Bay, Jamaica", October 2005.

[37] C. NEBUT, F. FLEUREY. *Une méthode de formalisation progressive des exigences basée sur un modèle simulable*, in "Langages et Modèles à Objets: LMO'05 (L'Objet logiciel, bases de données, réseaux, RSTI série l'Objet Vol. 11 N° 1-2/2005), Bern, Switzerland", February 2005, p. 145-158.

[38] A. RASSE, J.-M. PERRONNE, P.-A. MULLER, B. THIRION. *Using Process Algebra to Validate Behavioral Aspects of Object-Oriented Models*, in "MODEVA Workshop, MODELS / UML'2005, Montego Bay, Jamaica", October 2005.

[39] R. REDDY, R. FRANCE, S. GHOSH, F. FLEUREY, B. BAUDRY. *Model Composition - A Signature-Based Approach*, in "Aspect Oriented Modeling (AOM) Workshop, Montego Bay, Jamaica", October 2005.

[40] J. STEEL, J.-M. JÉZÉQUEL. *Model Typing for Improving Reuse in Model-Driven Engineering*, in "Proceedings of MODELS/UML'2005, Montego Bay, Jamaica", L. BRIAND, S. KENT (editors). , LNCS, vol. to be published, Springer, October 2005.

[41] T. ZIADI, J.-M. JÉZÉQUEL. *Manipulation de Lignes de Produits Logiciels : Une Approche Dirigée par les Modèles*, in "1ère Journées sur l'Ingénierie Dirigée par les Modèles, Paris", June 2005.

Bibliography in notes

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

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

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

[45] 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.

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

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

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

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

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

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

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

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

[54] G. WINSKEL. *Event Structures*, in "Petri Nets: Applications and Relationships to Other Models of Concur-

rency, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef", W. BRAUER, W. REISIG, G. ROZENBERG (editors). , vol. 255, Springer-Verlag, september 1986, p. 325-392.