

MAP-I
Programa Doutoral em Informática

Model-Driven Software Engineering

Unidade Curricular em Paradigmas da Computação
Paradigms of Computation
(UCPC)

DIUM, FEUP

April 14, 2008

Abstract

This document describes a Ph.D. level course, corresponding to a Curriculum Unit credited with 5 ECTS. It corresponds to a joint DIUM-FEUP proposal for UCPC (Paradigms of Computation) in the joint MAP-i doctoral program in Informatics, organized by three portuguese universities (Minho, Aveiro, and Porto).

LECTURING TEAM

DIUM: João M. Fernandes, Luís S. Barbosa, Alcino Cunha
FEUP: João Pascoal Faria, Ana Paiva

Coordinator: João M. Fernandes

A. Programmatic Component

1. Theme, Justification and Context

Motivation: Models in Software Engineering

So-called model-driven approaches to software design have gained, in recent years, widespread acceptance in software engineering. This raises a number of questions to the non-initiated, namely:

- what does model-orientation actually mean?
- how does one record models?
- (and above all) why does model-orientation matter?

In classical disciplines, by model one means an abstraction of something of interest. Reality is so complex that any attempt to consider it in full detail is doomed to fail. Scientists and engineers have learnt by experience to deliberately abstract from that part of reality which is not relevant to their current particular perspective. Naturally, features which can be ignored in one particular context may turn up to be essential to another one. In a sense, a balance between sharp observation and economy of thought (and common sense) is what is required in abstract modeling.

These ideas are well expressed in the consensual definition given by J. Rothenberg in "The Nature of Modeling in Artificial Intelligence, Simulation, and Modeling", John Wiley and Sons, 1989, pp. 75-92,

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 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.

Concerning how to record abstract models, drawings and diagrams are the most popular choices. However, scientists have learnt that drawings alone are unreliable means of passing knowledge on to future generations and that some kind of formal, unambiguous notation is required. The emergence of mathematical notation from natural language in science and engineering is among the main advances in modern, learned societies.

In an internet site devoted to stimulating youths' interest towards engineering disciplines (<http://www.discoverengineering.org>) one can read: Engineering is the application of math and science to create something of value from our natural resources. The fact that software engineering is not enrolled in the list of engineering branches in the same site (17 as a whole) can lead some to think that the definition is not wide enough to include branches which are more concerned with services than with natural resource transformation, and which might turn up to be less mathdependent. However, when software engineering emerged in the late sixties, the intention was not quite that, cf. the following excerpt of the 1968 NATO conference on the subject, which took place in Garmisch, Germany:

In late 1967 the Study Group recommended the holding of a working conference on Software Engineering. The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.

So, clearly, it has been always the intention of software engineers to manufacture software from rigorous models of real problems and situations which can be animated, queried and reasoned about. Moreover, so-called model-oriented specification is a formal technique that has been in use for many years in methodologies such as eg. VDM, Z or B. So, model-orientation does matter and has a tradition in formal methods.

More recently, the term model-driven engineering (MDE) was coined to refer to the use of generative and transformational techniques for software engineering where system implementations are (semi-) automatically derived from models or specifications. It consists of systematically using "models" as primary engineering artifacts throughout the production lifecycle, but of course, everything depends on the underlying notion of what a model actually "is".

As mentioned above, mathematical modeling has a well-known tradition in both the exact sciences (e.g. physics, biology) and the social sciences (e.g. economics, sociology). In the context of software engineering, mathematical models are the essence of so-called model-oriented specification, a formal technique which has been in use for many years in methodologies such as e.g. VDM, Z or B. Within MDE, however, the word "model" does not necessarily refer to a piece of mathematics and purports solely the idea of abstracting irrelevant detail from a real problem, so as to make it tractable and machine-processable. Models are therefore regarded as first class entities, as expressed by the adagium "Everything is a Model".

Approach: what is Model-driven Software Engineering?

We define Model-Driven Software Engineering (or MDSE) as the application of model-driven engineering to software engineering, i.e., it consists of systematically using models as primary engineering artifacts throughout the software engineering lifecycle.

The most common types of models of software systems used today are UML models (with roots in several diagramming notations) and several types of formal models (with stronger roots in mathematics). The current trend in MDSE is accompanied by an increasing convergence between UML models and formal models.

The best known MDE initiative is the Object Management Group (OMG) called Model-Driven Architecture (MDA), which is a registered trademark of OMG. Another related acronym is Model-Driven Development (MDD) which is an OMG trademark. MDA is intended to support model-driven engineering of software systems. A gentle introduction to MDA can be found in the book, "MDA Explained: The Model Driven Architecture – Practice and Promise", by Anneke Kleppe, Jos Warmer and Wim Bast, from Addison-Wesley, 2003. The basic idea behind the MDA initiative is to use (UML) models not only as analysis and design documents but also as the basis for code generation. The benefits are increased productivity (via automated transformations from models to code as well as between models at different levels of abstraction), portability (with the distinction between Platform Independent Models and Platform Specific Models), interoperability (via PSM bridges) and maintainability (since models are easier to maintain than code). In order to be able to perform those transformations in an automated way, the models have to be of greater rigor and precision. Stronger roots and convergence with more formal notations can be observed in several components of the recent UML 2.0 standard: the Object-Constraint Language (or OCL) has roots in formal specification languages of the VDM and Z family; behavior state machine diagrams are now very close to the more formal SDL notation used in the telecommunication industry; the semantics of activity diagrams is now based on Petri Nets.

However, one should not reduce the more fundamental MDE trend to the MDA initiative. According to Jean Bézivin in his seminal paper "On the Unification Power of Models",

the MDA initiative is a particular variant of a new global trend called MDE (Model Driven Engineering)

Such is the scope of the present course.

Course Context

In such a context this course proposes an approach to Software Engineering based on the central role of models, their construction, validation, verification, transformation

and reuse. It seeks to lay the foundations for a sound discipline of Model-driven Software Engineering, instead of adopting a particular formalism, method or notation. It is organized around three main ideas:

- Sound principles - as enounced above
- Eclecticism - several approaches and notations are addressed rather than a single unifying one
- Pragmatism - the course goes beyond first principles into practical application of useful techniques and tools

More specifically, the goal of this course is to address the fundamental concepts and techniques of MDE, without forgetting its realization in the MDA initiative.

Besides the benefits already mentioned in the MDA initiative, the convergence of formal-based and UML-based models and methods, opens the way for automated verification and validation of the models (for earlier defect detection), the development of systems (or parts of systems) that are correct by construction, and the automatic generation of test cases from models for conformity testing of the final system. Existing legacy systems can also benefit from this trend by employing automated reverse-engineering techniques.

MDSE is an area of very active research, where theory and pragmatics, rigor and agility meet to achieve increased quality and productivity in the development of software intensive systems.

Brief Mention to CMU Related Courses

Carnegie Mellon University offers related courses in the Master of Software Engineering, which act simultaneously as core courses in the PhD Program in Software Engineering:

Models of Software Systems: “Scientific foundations for software engineering depend on the use of precise, abstract models for characterizing and reasoning about properties of software systems. This course considers many of the standard models for representing sequential and concurrent systems, such as state machines, algebras, and traces. It shows how different logics can be used to specify properties of software systems, such as functional correctness, deadlock freedom, and internal consistency. Concepts such as composition mechanisms, abstraction relations, invariants, non-determinism, inductive definitions and denotational descriptions are recurrent themes throughout the course.” Topics: foundations; state machines; Z; refinement and abstraction; concurrency (in FSP); model checking; Petri Nets; UML

Analysis of Software Artifacts: “Our ability to build, maintain, and reuse software systems relies on our ability to analyze effectively the products of software development. This course will address all kinds of software artifacts - specifications, designs, code, etc. - and will cover both traditional analyses, such as verification and testing, and promising new approaches, such as model checking, abstract execution and new type systems. The focus will be the analysis of function (for finding errors in artifacts and to support maintenance and reverse engineering), but the course will also address other kinds of analysis (such as performance and security). Various kinds of abstraction (such as program slicing) that can be applied to artifacts to obtain simpler views for analysis will play a pivotal role. Concern for realistic and economical application of analysis will also be evident in a bias towards analyses that can be applied incrementally. The course emphasizes the fundamental similarities between analyses (in their mechanism and power) to teach the students the limitations and scope of the analyses, rather than the distinctions that arose historically (static vs. dynamic, code vs. spec). The course will balance theoretical discussions with lab exercises in which students will apply the ideas they are learning to real artifacts.”

ACM Computing Classification System subjects covered:

- D. Software / D.2 SOFTWARE ENGINEERING / D.2.2 Design Tools and Techniques
- D. Software / D.2 SOFTWARE ENGINEERING / D.2.4 Software/Program Verification
- D. Software / D.2 SOFTWARE ENGINEERING / D.2.5 Testing and Debugging
- D. Software / D.2 SOFTWARE ENGINEERING / D.2.11 Software Architectures
- H. Information Systems / H.5 INFORMATION INTERFACES AND PRESENTATION / H.5.2 User Interfaces

2. Objectives and Learning Outcomes

This course aims at introducing a sound approach to the principles, methods and pragmatics of Software Engineering, understood as

... that form of Engineering that applies the principles of computer science and mathematics to achieve cost-effective solutions to software problems
[CMU/SEI-90-TR-003]

More specifically it intends to cover, both from the foundational and the methodological point of view, the construction, analysis, design, classification, animation, validation and verification of models for software systems at different levels of abstraction and concern. As a second objective the course aims at providing the conceptual tools for the use of models in all phases of the software process, with a particular emphasis on test planning and code generation. The course is not intended as an introductory survey to Model-driven Software Engineering, but as an opportunity of exposing students to cutting-edge research topics in this area, although presented in a coherent and integrated way. It is placed at a similar level and cover overlapping material with advanced modules in doctoral programs at leading academic institutions.

Upon successful completion of this curricular unit, students should be able:

- to explain the need for describing software systems with models, as a way to abstract from the system's complexity and to reason about its properties;
- to clearly differentiate from behavioral/dynamic and informational/structural models;
- to identify the expressiveness and limitations of different modeling frameworks (like UML, VDM, Petri nets, ERDs, DFDs, Problem Frames) and to describe the type of system views they better fit.
- to use models for the activities (analysis, design, implementation, testing, maintenance) associated with the development of large software systems.
- to create, analyze, validate, verify and transform sets of models with the adequate detail for the software system under development, based on a multiple view approach.
- to describe/implement model transformations for automating the software process, namely to generate final code from the models.

3. Course Contents

1. Principles of model-driven software engineering

This first course unit is intended to provide a motivation and context for MDSE and introduce its main principles. Topics to be addresses include, but are not restricted to a notion of abstraction, model quality, process vs. product, cost effectiveness, reuse, patterns, selection of modeling frameworks based on the modelling purpose, model consistency, model continuity, views and multiview approaches and modeling complexity.

2. Models

This is a central unit in the course whose purpose is to study several modelling frameworks in Software Engineering. Such study encompasses, for each case, foundations, associated methodologies and modeling notations. Models to be considered in this unit are classified with respect to

- the degree of mathematical precision involved, leading to a basic distinction between formal and semi-formal (often referred to as visual) models;
- their purpose, distinguishing between functional, data and control/dynamic/behavioral models;
- the underlying computation model, distinguishing between state-oriented, activity-oriented, structure-oriented, data-oriented, and heterogeneous;
- their domain of application, to capture functional or non-functional properties.

The Unified Model Language (UML) will be adopted as a representative of a semi-formal modeling framework. As the OMG specification states, UML is “a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system”. Although in itself it does not specify any methodological or design process, its role as a (collection of inter-related of) semi-formal notations in supporting software development, from business processes or global architectures down to database schema, and reusable software components, became more and more fundamental, almost a de facto standard, to the whole Software Engineering discipline. Topics to be addressed include: UML diagrams; OCL; action semantics and executable UML; extensibility mechanisms; UML meta-model. On the formal side, on the other hand, the course will cover both

- state-oriented (e.g. VDM and Spec#), and
- behavior-oriented methods (e.g. Petri nets and temporal logics)

A last topic to be addressed in this unit concerns model integration, within UML and between UML and other modeling frameworks.

3. Model Quality

If a project uses poor models, it risks encountering problems such as misunderstanding, building the wrong product, large testing effort and ultimately a low quality product. Hence, assuring the quality of models is of key importance for completing projects successfully in a MDSE approach. This course unit is concerned with model quality analysis and enforcement. In particular, the following topics will be considered:

- Quality attributes, e.g., completeness, consistency, non-ambiguity, conciseness, adequacy, esthetics, verifiability, maintainability, compliance

- Quality metrics
- Quality strategies, e.g., constructive and analytical

4. Model Verification and Validation

As a follow-up of the unit on model quality, this course unit introduces the principles and techniques that can be used to promote and analyze aspects of software quality during the software development life-cycle. The unit reviews the state of the art in software verification and validation, associated to the correct development of models (get the model right) and the analysis of their soundness, adequacy and general sensibleness (get the right model) , respectively. Particular topics to be covered include: V&V in the Software Engineering life-cycle; models for reasoning about software; classification, expression and enforcing of properties; static and dynamic analysis.

The emphasis is placed on practical static analysis techniques and tools to perform model based verification and validation of software, in terms of assigned invariants, pre- and post-conditions. These include both model checking (explicit vs. symbolic, timed, stochastic) and theorem proving (firstorder, higher-order), as well as their integration. It also presents an overview of dynamic analysis techniques, based on model prototyping, and a comparison with its static counterpart, although a deeper coverage will be left for the model-based testing unit. At this stage it is not pertinent to choose the exact tools to use in the course. Nevertheless, likely candidates are model checkers such as SMV, SPIN, Uppaal or PRISM, and theorem provers such as the Larch prover, PVS or HOL.

5. Model Transformation

This course unit covers the fundamental issue of model transformation within the two basic types of models considered in the course — visual (like UML) and formal (like VDM, Petri nets or ASM). Particular emphasis will be placed on the following topics:

- Stepwise refinement of both function-, data- and behavior-oriented models.
- Transformation of analysis models into design models making explicit the envisaged software architecture.
- Transformation of platform independent into platform specific models.
- Code generation (i.e., transformation of design models into code), especially for behavior-intensive systems, and its current limitations
- Coupled software transformations understood as the simultaneous transformation of several models preserving consistency relations holding between them. This area has strong connections to the viewupdate problem, i.e, the ability to translate updates on a view of a data source into updates on the data source itself, which is extremely relevant in MDSE.

- Model constraints (vulg. invariants, “business rules”) and their role in model transformation, i.e., principles for reasoning about constraints, including constraint logics and calculi that make these logics operational.

6. Model-based Testing

Model-based testing, a fundamental issue in MDSE, is a software testing technique in which test cases are derived (semi-)automatically from a model of the system under test (SUT). It allows checking the conformity between the implementation and the model of the SUT, introducing more systematization and automation into the testing process. This course unit will start by explaining the model-based testing process and its phases. Existing techniques to support each phase of the testing process will be reviewed. In particular, it will present techniques for

- Input data generation.
- Test case generation.
- Checking test coverage and adequacy.
- Dealing with the state explosion problem.
- Checking conformity between the model and an implementation.

At the end of this unit, the students should be able to apply model-based testing to a medium size software system selecting among the existing techniques the ones considered more adapted for the kind of model used.

7. Re-engineering

This unit will introduce techniques for the reverse engineering of software systems, with a particular emphasis on code slicing and strategic programming. The reverse engineering of the user interface code of software written in Java/Swing will be used as a case study.

8. Case-study

A challenging case study will be proposed in each instantiation of this course. This unit has a double purpose: not only to act as a case study discussion in strict sense but also to raise a research challenge in Software Engineering. For the previous edition (07/08) of MAP-i, the chosen area was human-computer interaction. In the 08/09 edition, a different case study will be selected.

4. Teaching Methods and Student Assessment

The course is organized around a number of lectures with the double purpose of covering the contents and introducing research challenges. Specific lecture notes for each course unit will be written.

- 50% based on written assignments (e.g., a research paper analysis) proposed on the end of each course unit.
- 50% based on an individual survey monograph.

5. Basic Bibliographic References

- Berard B, Bidoit M, Finkel A, Laroussinie F, Petit A, Petrucci L, Schnoebelen P. *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer, 1999.
- Bjorner D. *Software Engineering (Abstraction and Modelling; Specification of Systems and Languages; Domains, Requirements and Software Design)*, Springer, 2006.
- Clarke EM, Grumberg O, Peled DA. *Model Checking*. MIT Press, 2000.
- Douglass BP. *Real Time UML: Advances in the UML for Real-Time Systems*, Addison Wesley, 2004.
- Fitzgerald J, Larsen PG, Mukherjee P, Plat N, Verhoef M. *Validated Designs for Object-oriented Systems*, Springer, 2005.
- Jensen K, Kristensen LM. *Coloured Petri Nets: Modeling and Validation of Concurrent Systems*, Springer, 2008. To appear.
- Kleppe A, Warmer J, Bast W. *MDA Explained: The Model Driven Architecture – Practice and Promise*, Addison-Wesley, 2003.
- Stevens P, Pooley R. *Using UML: Software Engineering with Objects and Components*, Addison-Wesley, 2000.
- Utting M, Legeard M. *Practical Model-Based Testing: A Tools Approach*, Morgan Kaufmann, 2007.
- Wieringa RJ. *Design Methods for Reactive Systems: Yourdon, StateMate, and the UML*, Morgan Kaufmann Publishers, 2003.

B. Lecturing Team

1. Team Presentation

This course is supported by a team involving researchers from both the University of Porto, FEUP (João Pascoal Faria and Ana Paiva), and the University of Minho, DI/CCTC (João M. Fernandes, Luís Soares Barbosa, and Alcino Cunha). This course was offered in the first edition of the MAP-i programme and was attended by 15 students.

All team members are working, and have worked actively in the past few years, on topics that are directly related to the subjects covered by this course, as detailed below.

2. Coordinator

The coordinator of the unit is João M. Fernandes.

3. Short Presentation of Team Members

In the sequel we introduce a brief presentation of each team member, which includes, for each of them, up to 5 key publications related to the scientific area in which this course is proposed. **All CVs are supplied in separate PDF documents.**

João M. Fernandes is associate professor at the Department of Informatics of Minho University, and a researcher member of CCTC. His scientific research activities are centered around the areas of systems modelling and development of software for embedded systems. In the systems modelling area, his interests lie on the usage of the UML and High-level Petri nets, as specification notations for highly complex software systems, and in studying and applying model-driven development approaches. In the embedded software area, his attention focus on issues related to the methodological approach to follow, namely the requirements capture, the software process model, the methods used for development, and the transition between phases. He is a co-founder and regular editor for the International Workshop Series on Model-based Methodologies for Pervasive and Embedded Software (MOMPES). He is a funding member of the IFIP Working Group 10.2 (Embedded Systems). In 2002/03, he was a postdoctoral researcher at TUCS (Turku, Finland), and in 2006/7 he was an invited assistant professor at University of Aarhus (Denmark).

Key Publications:

- Machado RJ, Fernandes JM, Monteiro PA, Rodrigues H. Transformation of UML

Models for Service-Oriented Software Architectures, 12th IEEE International Conference on the Engineering of Computer Based Systems (ECBS 2005), Greenbelt, Maryland, USA, pp. 173-182, IEEE Computer Society Press, Apr/2005.

- Machado RJ, Ramos I, Fernandes JM. Specification of Requirements Models, Engineering and Managing Software Requirements, Aurum A., Wohlin C. (eds.), chap. 3, pp. 47-68, Springer, Jul/2005.
- Machado RJ, Fernandes JM, Monteiro PA, Rodrigues H. Refinement of Software Architectures by Recursive Model Transformations, 7th International Conference on Product Focused Software Process Improvement (PROFES 2006), Amsterdam, The Netherlands, LNCS 4034, pp. 422-428, Springer, Jun/2006.
- Fernandes JM, Lilius J, Truscan D. Integration of DFDs into a UML-based Model-Driven Engineering Approach, Software and Systems Modeling (SoSyM) 5(4):403-428, Springer, Dec/2006.
- Fernandes JM, Jørgensen JB, Tjell S. Requirements Engineering for Reactive Systems: Coloured Petri Nets for an Elevator Controller, 14th Asia-Pacific Software Engineering Conference (APSEC 2007), Nagoya, Japan, IEEE Computer Society Press, pp. 294-301, Dec/2007.

Luis Soares Barbosa is associate professor at the Department of Informatics of Minho University, and a researcher member of CCTC. His research area is program semantics and calculi applied to systems understanding and rigorous software construction. In particular, most of his work has been devoted, concerns the development of formal models and calculi for software components, services and architectures. Since 2003 he has been the coordinator of a FCT project on Program Understanding and Re-engineering as well as of a cooperation project with School of Mathematics of Peking University on Formal Models for Software Components. From 1990 to 2004 he coordinated, as an invited lecturer, a Seminar on Systems Design at the Faculty of Engineering of the University of Bristol, UK. He was also an invited lecturer in Mathematics and Computer Science PhD programs at Tartu University, Estonia (2003), and Peking University, China (2006).

Key Publications:

- Sun M, Naixiao Z, Barbosa LS. On Semantics and Refinement of UML Statecharts, Proc. of 2nd IEEE Int. Conf. on Software Engineering and Formal Methods, Cuelar J, Liu Z (eds), pp. 164-173, IEEE Computer Society Press, 2004.
- Cruz A, Barbosa LS, Oliveira JN. From Algebras to Objects: Generation and Composition, Journal of Universal Computer Science 11(10):1580-1612, 2005.

- Barbosa LS, Sun M, Aichernig BK, Rodrigues N. On the Semantics of Componentware: a Coalgebraic Perspective, in *Mathematical Frameworks for Component Software: Models for Analysis and Synthesis*, He J, Liu Z (eds), Series on Component-Based Software Development, World Scientific, 2006.
- Rodrigues N, Barbosa LS. Program Slicing by Calculation, *Journal of Universal Computer Science* 12(7):828-848, 2006.
- Barbosa LS, Oliveira JN. Transposing Partial Components: an Exercise on Coalgebraic Refinement, *Theoretical Computer Science* 365(1-2):2-22, 2006.

Alcino Cunha is assistant professor at the Department of Informatics of Minho University, and a researcher member of CCTC. He received his Ph.D. degree in 2005. His general research areas are program calculation, functional programming, data refinement, and coupled software transformation, where he has developed extensive work, partially documented in the 5 selected publications mentioned below

Key Publications:

- Cunha A, Pinto JS. Point-free program transformation. *Fundamenta Informaticae*, 66(4):315-352, 2005.
- Cunha A, Pinto JS, Proença J. A framework for point-free program transformation. In A. Butterfield, C. Grelck, and F. Huch, editors, *Selected Papers of the 17th International Workshop on Implementation and Application of Functional Languages*, volume 4015 of LNCS, pp. 1–18. Springer-Verlag, 2006
- Cunha A, Oliveira JN, Visser J. Type-safe Two-level Data Transformation. 14th International Symposium on Formal Methods (FM 2006), Hamilton, Ontario, Canada, LNCS 4085, pp. 284-299, Springer, Jul/2006.
- Cunha A, Visser J. Transformation of Structure-Shy Programs - Applied to XPath Queries and Strategic Functions. *Partial Evaluation and Program Manipulation (PEPM 2007)*, Nice, France, pp. 11-20, ACM Press, Jan/2007.
- Berdaguer P, Cunha A, Pacheco H, Visser J. Coupled Schema Transformation and Data Conversion for XML and SQL. *Practical Aspects of Declarative Languages (PADL 2007)*, Nice, France, LNCS 4354, pp. 290-304, Springer, Jan/2007.

Ana Paiva is assistant professor at Engineering Faculty of Porto University. She has been developing research work on model-based GUI testing in collaboration with Foundation of Software Engineering research group within Microsoft Research. She had the opportunity to extend Microsoft's model-based testing tool, Spec Explorer, in order to make it more adapted for GUI testing. She had to face with several problems concerned with the implementation and automation of the model-based testing process. In particular, she developed a method for modeling GUIs, a technique to cope with the

state space explosion problem, and a tool to map model actions with simulated user actions on a GUI implementation for checking conformity between the GUI model and its implementation. She is a member of the CYTED network on software verification and validation (REVVIS). Her input will be valuable specially in modeling techniques, model-based testing and in case study construction.

João Pascoal de Faria is assistant professor at the Department of Electrical and Computer Engineering of FEUP, and a researcher at INESC Porto. He has active research work in the area of model-based testing (one of the major topics of the program proposed), based both on formal models and UML models, since 2003. He has more than 10 years of experience in teaching and supervising students' projects in object oriented analysis and design with UML and previous notations, and a 4-year experience in teaching Software Testing. He has a 10-year experience in the development and maintenance of a rapid application development (RAD) tool between 1989 and 1999 that is still in use, and currently has a strong interest in combining formal and traditional methods for rapid and rigorous application development. He is the proposal coordinator of a research project currently submitted to FCT on An Automated Model-based User Interface Testing Environment.

Key Publications (together with Ana Paiva):

- Paiva A, Faria JP, Vidal R. Specification-based Testing of User Interfaces, 10th Design Specification and Verification of Interactive Systems Workshop (DSV-IS 2003), Funchal, Portugal, LNCS 2844, pp. 139-153, Springer, Jun/2003.
- Paiva A, Faria JP, Vidal R. Automated Specification-based Testing of Interactive Components with AsmL, QUATIC 2004, Porto, Oct/2004.
- Paiva A, Faria JP, Vidal R, Tillmann N. Modeling and Testing Hierarchical GUIs, 12th International Workshop on Abstract State Machines (ASM 2005), Paris, France, pp. 329-344, Mar/2005.
- Paiva A, Faria JP, Vidal R, Tillmann N. A Model-to-implementation Mapping Tool for Automated Model-based GUI Testing, 7th International Conference on Formal Engineering Methods (ICFEM 2005), Manchester, UK, LNCS 3785, pp. 450-464, Springer, Nov/2005.
- Paiva A, Faria JP, Vidal R. Towards the Integration of Visual and Formal Models for GUI Testing, 3rd Workshop on Model Based Testing (MBT 2007) at ETAPS 2007, Braga, Portugal, 2007.