Editorial

Dear Reader,

The MoVES project on Modelling, Verification and Evolution of Software has entered its last year and this is already the fourth edition of the MoVES newsletter. In this month’s issue we focus on the work package on “Consistency Checking and Co-Evolution” and also seize the opportunity to zoom in on some of the research topics that are currently under active investigation at UCL university in Louvain-la-Neuve.

In particular, this newsletter contains contributions by Dr. Andy Kellens et al. on co-evolution of source code and structural design regularities (a collaboration between UCL and VUB), by Dr. Ragnhild Van Der Straeten et al. on consistency checking and inconsistency management in model-driven engineering (a collaboration between UMONS and VUB), by Dr. Anthony Cleve on a conceptual approach to database applications evolution (a collaboration between FUNDP and INRIA), by Arnaud Hubaux on inconsistency management in feature configuration workflows and by Anne Keller et al. on change impact analysis for UML model maintenance.

Dedicated contributions from UCL include research on language engineering for dynamic software evolution conducted by the RELEASeD research group headed by Prof. Kim Mens, research on constraint-based modelling at the Constraint Group lead by Prof. Yves Deville, research on Formal Verification of Complex Software Systems at the Louvain Verification Lab headed by Prof. Charles Pecheur, and finally research on Model-Driven System Engineering conducted by Prof. Axel van Lamsweerde and his team.

I wish you a pleasant reading and a productive research year!

Prof. Kim Mens

Upcoming Events & Recent Joint Publications

- Recent Joint Publications:

- Recent Events:
  - EVOLUMONS 2011: Research Seminar on Software Evolution, January 26, 2011 – UMONS, Mons, Belgium
The design and implementation of software systems are governed by a broad variety of so-called structural regularities: coding conventions, architectural guidelines, design rules, and so on. For a system to remain maintainable and comprehensible, or sometimes even to exhibit the correct behaviour, it is imperative that these regularities are adhered to in the source code. Unfortunately, as these regularities are often only implicitly known and poorly documented, respecting them consistently throughout the source code can be a daunting task. Our work aims at co-evolving regularities and source code by providing tool support for identifying, documenting, verifying and correcting regularities.

First, for an existing system, it might not immediately be clear which regularities are governing that system. To extract this information from the source code of a system, we are investigating novel approaches [1] based on association rule mining. Our approach returns association rules (i.e., “if ... then” rules) that express potentially interesting regularities in the system (e.g., “if a class inherits from UIAction, that class should override the methods performAction and undoAction”). Once the regularities governing a system are known, our IntensiVE tool suite [2] allows for the documentation and verification of these regularities. To this end, IntensiVE offers a set-theoretic formalism on top of the logic program query language SOUL. Our tool suite consists of a number of tools for verifying regularities, visualizing them, and for integrating the verification process with the development process (integration with unit testing frameworks and continuous integration tools). Currently, IntensiVE supports programs written in Smalltalk, Java, C(++) and COBOL. Finally, by means of the HEAL tool [3], we support the semi-automatic correction of identified violations of regularities. HEAL leverages the fact that we employ a logic formalism to document regularities and — by means of logic abduction — analyses the underlying cause of a violation. Based on this information, HEAL then proposes possible fixes for the violation, and automates these fixes.


Further Information: http://www.intensive.be/
Contact: akellens@vub.ac.be, Kim.Mens@uclouvain.be
A Conceptual Approach to Database Applications Evolution

Data-intensive applications generally comprise a database and a collection of application programs in strong interaction with the former. Such applications constitute critical assets in most enterprises, since they support business activities in all production and management domains. As any software systems, they usually have a very long life, during which they are subject to continuous evolution in order to meet ever-changing business and technical requirements.

The evolution of data-intensive applications is known as a highly complex, expensive and risky process. This holds, in particular, when the evolution involves database schema changes, which in turn have an impact on data instances and application programs.

Recent studies show, in particular, that schema evolutions may have a huge impact on the database queries occurring in the programs, reaching up to 70% query loss per new schema version. Evaluating the impact of database schema changes on related programs typically requires sophisticated techniques, especially in the presence of dynamically generated queries. The latter also severely complicate the adaptation of the programs to the new database schema. Without reliable methods and tools, the evolution of data-intensive applications rapidly becomes time-consuming and error-prone.

In our recent joint work between INRIA Lille and the University of Namur, we addressed the problem of automated co-evolution of database schemas and programs. Building on our previous work in the field, we present a comprehensive approach that supports the rapid development and the graceful evolution of data-intensive applications. The proposed approach combines the automated derivation of a relational database from a conceptual schema and the automated generation of a data manipulation API that provides programmers with a conceptual view of that relational database. This conceptual API can be re-generated in order to mitigate the impact of successive schema evolutions and to facilitate their propagation to the program level.


Contact: rvdstrae@vub.ac.be
Further Information: Recent collaboration between FUNDP and INRIA

Inconsistency Management in Model-driven Engineering

Model-driven engineering (MDE) is an approach to software engineering where the primary assets are models. Each model is expressed in a certain modeling language. The description of a software system consists of several models each describing particular aspects of the system. Models can be refined, evolved into a new version, and can be used to generate executable code. The ultimate goal is to raise the level of abstraction and to develop and evolve complex software systems by manipulating models only. The manipulation of models is achieved by means of model transformations.

One of the challenges of manipulating models is to manage inconsistencies between these models. A model is considered to be inconsistent if it contains undesirable patterns, which are specified by so-called inconsistency rules. These patterns and rules can reveal and capture problems of lexical, structural, behavioural or visual nature. Detection of inconsistencies invariably consists in traversing the model in order to detect these undesirable patterns. In the past we conducted research in the area of inconsistency detection [1].

Inconsistency management not only consists of detecting inconsistencies but also of handling them. We are investigating two approaches that allow us to generate inconsistency resolutions automatically. (1) The first one is based on the usage of a SAT solver. Models and consistency rules are expressed in a high-level specification language. The resulting specification is passed on to an off-the-shelf SAT solver. This brute-force approach reveals the complexity of the inconsistency resolution problem. For each inconsistency, lots of resolutions, in some cases even an infinite set of resolutions are possible. Techniques need to be developed to instruct the SAT solver very carefully. (2) Another approach is to use automated planning, a technique stemming from the Artificial Intelligence domain. Jorge Pina Puissant, a Ph.D. student of UMONS supervised by Prof. Tom Mens is working on this topic. In a first experiment, an “off-the-shelf” progression planning tool was used to assess the scalability of using the planning technique in the domain of inconsistency resolution. However, this forward search did not scale and the development of a custom-built regression planning in Prolog is initiated. The results look promising [2].

In the context of a project of the "Communauté française" aiming at strengthening the relations between young French and Walloon research teams, we are comparing existing approaches that deal with model inconsistency resolution. The partners of this project are Prof. Tom Mens, Prof. Xavier Blanc (University of Bordeaux) and dr. Ragnhild Van Der Straeten. The objective is to gain insight in current approaches and to identify challenges for future research.


Contact: rvdstrae@vub.ac.be
Further Information: Ongoing collaboration between VUB and UMONS
Change Impact Analysis for UML Model Maintenance

Software maintenance is likely the most costly phase in the software life-cycle. The software system to be maintained consists of numerous inter-dependent artifacts that, inevitably, each undergo various changes during maintenance. What makes planning and executing these changes difficult, is that each change may have severe ‘ripple effects’ to other points of the system that are difficult to assess due to the inter-dependent nature of the artifacts. The goal of our work is to introduce a light-weight and accurate model-based change impact analysis technique that helps estimate the cost of the changes in terms of time, labor and money, reduce potential errors due to unexpected side-effects, and finally, efficiently implement the change.

To validate our work, we evaluate precision and recall of the impact obtained with our technique compared to the actual changes recorded. The change scenario chosen is changes that occur in order to resolve inconsistencies between different models (corrective maintenance). The inconsistencies studied are obtained by checking UML models with consistency rules such as defined by [2]. Up to now, validation was performed on two case studies which together contain approximately 5686 model elements on which 3287 inconsistencies are resolved. The validation returned a mean precision and recall of (0.77, 0.95) and (0.97, 0.93). The figure on the right shows one example of the impact compared to the actual changes. Without going into detail, the figure illustrates two important points. On the one hand, the figure shows cases of how the impact predicts the number of changed elements rather accurately, as well as cases where it is above and below. We want to stress that the impact is not always only a worst case assessment and therefore it is necessary to evaluate both precision and recall. Secondly, we emphasize the wide range of the impact function. The smallest impact is 1, e.g., for Res1 and the highest impact we encountered was 160 model elements for Res4.


Further Information: http://win.ua.ac.be/akeller/
Contact: Anne.Keller@ua.ac.be, Serge.Demeyer@ua.ac.be

University in Focus – University of Louvain (UCL)

Ensuring the safety and reliability of nowadays complex software systems is universally acknowledged as a very difficult and costly issue, and has been a major challenge of the discipline for decades. To address this challenge, the Louvain Verification Lab (LVL) at UCL investigates principles, tools and applications of formal analysis and verification of computer systems. The following themes have particularly benefited of exchanges within the MoVES consortium.

**Symbolic Model Checking of Asynchronous Systems**

Model checking is an efficient technique for verifying properties on reactive systems. BDD-based symbolic model checking is a well-known approach to deal with the state space explosion problem in model checking, though BDDs can still suffer from space blow-up. More recently, bounded model checking using SAT-based procedures has been used as a very successful alternative to BDDs. However, these approaches give poor results when it is applied to models with a lot of asynchronism. On the other hand, partial-order reduction (POR) methods reduce the number of states explored by model-checking, by avoiding to explore different equivalent interleavings of concurrent events. We are developing, implementing and experimenting with algorithms which combine partial-order reduction and symbolic model checking to allow more efficient verification on models featuring asynchronous processes.

**Human-Computer Interaction Verification**

In the context of critical applications, like autopilot in civil aviation, confusions occurring when using these can lead to dangerous situations. Human-Computer Interaction analysis follows system observability issues, abstraction and simulation which are well studied in the formal methods field. We are working on generating interfaces for critical systems, satisfying some controllability properties, using formal techniques.


Further Information: http://lvl.info.ucl.ac.be
Contact: charles.pecheur@uclouvain.be
Computing platforms are more connected to their physical and logical environment than ever before. If equipped with higher levels of context-driven adaptability, software systems can become smarter with respect to this environment and to user needs, exhibiting emergent properties, resiliency in face of perturbations, and generally fitting better in their technical ecosystem.

**Programming in isolation** Unfortunately, most software systems do not meet the high adaptability expectations that stem naturally from their connectedness to their environment. Most applications exhibit fixed functionality and are seldom aware of their changing execution context to adapt their behaviour accordingly. Many chances of delivering improved services to users and network peers are thus missed. We hypothesise that a major reason for this lack of adaptability is the unavailability of appropriate context-aware programming abstractions, methodologies and tool sets. Current programming technology does not put programmers in the right state of mind, nor does it provide adequate tools, to program applications that are smart with respect to their environment.

**Towards a mindset shift: programming with context** New paradigms are needed that help overcoming the limiting vision of current approaches by putting programmers in the right state of mind to build dynamically adaptable applications from the ground up. By taking the context into account, new opportunities are unveiled for delivering improved services according to the physical and logical circumstances in which those services are used. Starting from the previous observation, we have been developing programming technology that permits the natural expression of adaptable behaviour according to changing contexts. This new programming technology allows flexible software compositions that can change at run time, thereby enabling dynamic software evolution. Dynamic evolution constitutes a radical, yet powerful way of achieving software adaptability to context. This endeavour has lead us to the development of the Ambience and Subjective-C languages.

---

**Further Information:** [http://ambience.info.ucl.ac.be/](http://ambience.info.ucl.ac.be/)

**Contact:** sebastian.gonzalez@uclouvain.be

---

Constraint-based approaches are well-suited for modeling complex problems. In the context of the MoVES project, the Constraint Group at UCL is mostly interested in the design of modeling abstractions dedicated to specific classes of problems. For such classes of problems, we focus on the design of specialized modeling languages, allowing the synthesis of efficient algorithms, either based on constraint programming or on constraint-based local search.

**Modeling graph matching problems** Matching problems, such as subgraph isomorphism, are complex problems that have many applications (pattern recognition, bioinformatics, ...). Constraint-based models have been used for solving various graph matching problems. New filtering techniques have been developed [1] as well as a modeling language dedicated to matching problems.

**Modeling of scheduling problems** We developed a modeling language for scheduling problems [2]. Scheduling algorithms are synthesized from high-level models. Specific filtering techniques have also been developed for just-in-time scheduling and for the open-shop problems.

**Introduction of graph objects in constraint-based local search (CBLS) models** Graphs and trees have been developed on top of the Comet CBLS modeling language. We also showed how such objects can be used for solving complex constrained optimum path problems [3].

---

**Further Information:** [http://becool.info.ucl.ac.be/](http://becool.info.ucl.ac.be/)

**Contact:** yves.deville@uclouvain.be
This group developed a multi-view modelling framework and tool to inter-relate the goals, objects, agents, scenarios, operations, and domain properties of a system to be developed or re-engineered. The framework provides a graphical modeling language, a lightweight formalism for precise model specification, an optional real-time temporal logic for model analysis, a systematic method for model elaboration, and various dedicated techniques for goal refinement, operationalization, conflict management, hazard analysis, agent responsibility assignment, and goal mining from scenarios. Recent developments that relevant to MOVES include the following:

**Model Synthesis** A new interactive technique for synthesizing behavior models, captured by labeled transition systems, from scenarios captured by message sequence charts. Compared with others, the synthesizer requires no additional input such as state or flowcharting information. The technique extends known learning techniques for grammar induction. It is incremental on training examples and requires the end-user to interactively classify as positive or negative additional scenarios that are generated by the synthesis procedure. Experience with this approach shows that the number of such scenario questions may become fairly large in complex applications. The technique has therefore been optimized by injecting additional information into the synthesizer, when available, in order to constrain induction and prune the inductive search space. Such information may include global definitions of fluents that link interaction events and atomic assertions; declarative properties of the domain; behavior models of external components; and goals that the software system is expected to satisfy.

**Reasoning about alternative options** Model building often raises multiple options regarding the target system. A lightweight quantitative technique has been developed to support this. The aim is to determine the overall score of each option with respect to all leaf soft goals in the system’s goal refinement graph, taking their respective importance into account. The option with highest score is then selected.

**Analysis of safety-critical medical processes** Decision-based process models capture processes where the application of specific tasks and their sequencing depend on explicit decisions based on the state of the environment in which the process operates. Decision-based processes in areas such as healthcare are often critical in terms of timing, resources, and environmental consequences (e.g., radiotherapy or chemotherapy processes). A variety of tool-supported techniques for analyzing models of such processes were developed. The analyses allow us to highlight incomplete decisions, non-deterministic decisions, inaccurate decisions due to inaccurate or outdated information about the environment, violations of temporal constraints, and inadequate resource usage. The analyses differently instantiate the same generic fixpoint algorithm for propagating abstract decorations through the model graph. The language of Guarded High-Level Message Sequence Charts is used for modeling decision-based processes. To enable these analyses, this language is extended with timing constructs, preconditions, and state variables tracking environment quantities. These extensions are grounded on the formal semantic framework of labeled transition systems and fluents introduced earlier.