Dear reader,

This here is the third installment of the Fundamental Issues in Modelling, Verification and Evolution of Software (MoVES) newsletter. The goal of the newsletter is to highlight the work performed in the context of the project through the different workpackages. As with previous issues, this edition of the newsletter is divided in two parts, the first one focusing on the results achieved in the Programming Languages workpackage, and the second one on the work performed by the partners at the Vrije Universiteit Brussel (VUB).

The goal of workpackage 1 is to investigate novel abstraction techniques in programming languages that facilitate the mapping of abstract and high level models onto executable models. In this issue we present the results of collaborations on the topics of validation of compositions resulting from the use of aspect and context-oriented techniques, as well as more generic composition problems, such as the merging of changes in a code repository, and the composition of source code regularities into design regularities application. These results validate the relevance of the work performed in the package as well as the high degree of inter-institution collaboration. Testament of this is the work presented by two cotutelles, Nicolás Cardozo (UcL - VUB) and Verónica Uquillas (VUB - INRIA).

For the second part of this issue, we present a sample of the work currently carried out in the Software Languages Lab at the VUB. The different themes: human-computer interface, environmental mobile computing, workflow composition and multi-core programming, represent the different areas that the Software Languages Lab is currently exploring; all tackled from a programming language perspective.

We trust that you will find this issue entertaining as well as informative, for more information about the project, visit http://moves.vub.ac.be/.

Carlos Noguera and Dirk Deridder

Upcoming Events & Recent Joint Publications

- Recent and upcoming publications

- Upcoming events
Aspect-Oriented Programming (AOP) is a programming paradigm that supports the modular implementation of crosscutting concerns, i.e. concerns that are scattered throughout a system and tangled with other core concerns. In contrast with the other paradigms, AOP does not rely on explicitly called behavior: an aspect module encapsulates both the behavior of a concern and the specification of the places where this behavior should be executed. As a result, AOP improves the maintainability, reusability, and readability of software. What it does not improve however is the predictability of the resulting application: the behavior of an aspect can be triggered anytime. As a consequence, it is hard to anticipate how several aspects will interact with each other and whether or not these interactions will lead to undesired interferences. Techniques to handle the composition of aspects are therefore needed. Among the techniques that have been proposed, some focus on the detection of aspect interferences while the others focus on their resolution. Approaches to detect interferences are based on static analysis and model checking tools, which fail at handling the dynamic facet of the interferences. On the other hand, approaches to resolve these conflicts are typically not generic enough: each one handles a specific kind of interference. Moreover, in most cases, the code of the conflicting aspects has to be invasively modified in order to make these aspects aware of each other. As a result, the code to handle the composition of aspects is scattered over the aspects themselves and tangled with their fundamental purpose. Based on these observations, we proposed reflective AOP: aspects that have the ability to modularize concerns crosscutting the aspects themselves. Aspects are therefore empowered to reason about and act upon aspect computations. Consequently, an aspect can monitor interacting aspects at runtime and detect erroneous patterns considered as interferences. It can also impact the execution of these aspects to resolve the interferences, without any need to modify their implementation. Finally, because our approach is based on principled reflection concepts, it also supports the composition of composition aspects.

Recent years have seen a revival of dynamic languages in the setting of ambient intelligence, where adaptation to the context of execution is key for ubiquitous or pervasive applications. Among the family of dynamic languages, context-oriented languages are those who provide specific language constructs to deal with the dynamic adaptation of applications to the surrounding environment. These languages allow developers to associate specific behavior to particular contextual situations. Behavior becomes active freely whenever the associated contextual situations take place, making the interaction among behavior imperative. Existing technology lacks the appropriate abstractions to ensure that interaction between the behavioral adaptations maintain application predictability: consistency, correctness, and completeness. We provide three execution models to the development of context-aware applications to bridge the existing gap between the modeling and development processes. The models will provide programers with: 1) means to define and represent restrictions between contextual entities, 2) an overview of the run-time interaction between contextual entities, and 3) a definition of behavioral blocks (methods) and object entities. The introduction of these execution models enhances the predictability of the system by three major means. It allows the definition of different relations between contextual entities to avoid inconsistencies when they interact. It allows the developer to identify heterogeneous sources of information and to combine them in a sound way to ensure information reliability. And it allows to dynamically introduce fall-back mechanisms for the systems reaction to incomplete information. Reconciliation of model entities and programing entities is achieved by providing a direct mapping between the two of them. Each concept expressed in one of the models has the respective concept as a language feature. The introduction of a context manager, ensures the verification and validation of constraints imposed in the models (e.g. contracts between contextual entities, and exclusion and dependency rules among others), and the synchronization and co-evolution of the models.
Integrating changes that represent fixes, enhancements or new features are key software development activities. However, there is no adequate support to help release-masters to take decisions about the integration of published merged changes into the system release. Most of the tools are textual diffs that do not provide an overview of the changes (how changes are distributed, what groups of entities changed). At the same time, existing tools do not offer the possibility to understand detailed changes within their specific context.

Torch is a visual tool for understanding source-code changes. It is the result of a joint research between the Software Languages Lab at the Vrije Universiteit Brussel and the RMoD Team at INRIA Lille/University of Lille 1. Torch supports integrators in taking decisions about the integration of changes before performing the actual merging, and offers developers a means to understand and control their changes before publishing them. Torch provides an overview of how a Smalltalk program was changed, and aims at aiding its users in understanding these changes.

Torch characterizes the changes based on structural information, authorship and symbolic information. It offers a dashboard that presents different structural representations of source code changes using visualizations. The Dashboard speeds up the access to the textual information of changes by using a diff as a fly-by-help. By combining graphical and textual data, Torch brings semantic information to changes exploration.

Torch has been picked up by the Pharo Smalltalk community and is actively used in the integration process of various Pharo projects.

Further Information: http://soft.vub.ac.be/torch
Contact: Verónica Uquillas-Gómez, vuquilla@vub.ac.be

Independently of the modularization techniques used to implement the abstractions of an application, maintaining its source code is a challenging task. First, changes inevitably introduce undesired and unnoticeable design degradation. Second, knowledge of the high-level abstractions of the application and their mapping to source code is seldom documented or this documentation is lost over time. Our work analyzes the usefulness of diverse characteristics of source code entities as indicators of design abstractions to tackle both maintainability issues mentioned above. By finding source code indicators of high-level abstractions we aim at recovering lost design knowledge. Deviations to these high-level abstractions are considered possible design degradations. Our approach is based on the assumption that developers implicitly or explicitly rely on naming schemes, implementation patterns, idioms, usage protocols, etc., to reflect their design decisions in the source code. These regularities consist of a set of characteristics shared by source code entities implementing a high-level abstraction. To detect these regularities our approach relies on association rule mining. Association rule mining takes as input the set of characteristics describing each source code entity of the analyzed application. Its output is a set of rules of the type "if a class belongs to the hierarchy of Action, then it always (100% likelihood) implements the method performAction" which indicates the implementation of the command pattern.

Inherently, all the regularities and deviations found by our technique are correct; nevertheless, they do not necessarily convey design decisions nor design degradations. Furthermore, rules by themselves are not good enough to support maintenance tasks because they tend to be redundant and might not provide novel information. Therefore, most of our work lays in finding novel ways to filter redundant and obvious information, as well as grouping related rules into concise regularities. However, the usefulness of a regularity depends on the context of its usage. For that reason, we integrated the detection of source code regularities in an IDE to log the context (maintenance task, source code entity that is being maintained, level of experience of the developer with the application and with the programming language) and the usefulness of the regularities (whether or not changes done fix deviations or increase the likelihood of rules). Analyzing this information would provide quantitative evidence of the impact of the approach, along with hints to create more accurate filters.

Further Information: http://www.info.ucl.ac.be/~alozano/
Contact: Angela Lozano, angela.lozano@uclouvain.be
Complex Event Processing for Multimodal Human Computer Interaction

Over the past few years, multi-touch user interfaces emerged from research prototypes into mass market products. This evolution has been mainly driven by innovative devices such as Apple’s iPhone or Microsoft’s Surface tabletop computer. Unfortunately, there seems to be a lack of software engineering abstractions in existing multi-touch development frameworks. Many multi-touch applications are based on hard-coded procedural low level event processing. This leads to proprietary solutions with a lack of gesture extensibility and cross-application reusability.

We present Midas, a declarative model for the definition and detection of multi-touch gestures where gestures are expressed via logical rules over a set of input facts. The advantage of such an approach is that the programmer no longer needs to be concerned about how to derive gestures but only about describing the gesture itself. Complex gestures, which are extremely hard to implement in traditional approaches, can be expressed in one or multiple rules which are easy to understand. The use of a rule language has the benefit that the developed gestures are reusable and easy to compose. Furthermore, a strong connection to application-level entities allows developers to activate and deactivate gestures depending on their graphical context.

Our current focus is to vastly improve the performance of set-based reasoning in order to support larger time-frames. With this greater efficiency we foresee an extension of spatio-temporal operators for multimodal input devices (e.g. combining speech with multi-touch, interactive pens and more). Lastly we plan to provide software engineering abstractions in order to solve conflict situations for clashing gestural input.

Further Information: http://soft.vub.ac.be/soft/research/midas
Contact: Lode Hoste, lhoste@vub.ac.be

Participatory Sensing for Sustainable Urban Life

Data on environmental pollution in cities is remarkably limited. Pollution maps are typically created through computer simulations based on general statistics, backed up by only limited amounts of measurements in the field. Current measuring methods are also expensive and thus not very scalable. The resulting maps are hard to interpret by citizens because they lack spatio-temporal detail as well as semantic and subjective meta-data. A people-centric approach to environmental monitoring is important, not only for assessing individual exposure but also for awareness building. Most citizens lack access to tools to estimate the quality of their daily environment and how it is affected by their behaviour. Therefore it is hard to sensibilise the community or set up local actions towards policy makers.

Our goal is to design, create, validate and deploy software tools to facilitate environmental measurement surveys conducted by citizens through participatory sensing and social tagging.

Participatory sensing appropriates everyday mobile devices such as cell phones to form interactive sensor networks. This people-centric data collection places measuring tools literally in the hands of citizens who measure their individual exposure and obtain immediate feedback, contributing to their awareness of the ecological conditions they live in. While participatory sensing does not reach the level of accuracy of state-of-the-art environmental monitoring stations, we expect to alleviate this by collecting massively more data, of higher spatio-temporal granularity and by calibrating devices and correlating data.

Social tagging augments quantitative measurements with qualitative meta-data in the form of tags, open-ended keywords entered through mobile phones. This provides contextual, semantic and subjective knowledge that facilitates analysis and interpretation.

We primarily work on the NoiseTube project (initiated at the Sony Computer Science Lab in Paris), which applies these concepts and technologies to the problem of urban noise pollution. The system consists of a noise measuring application for cell phones and a central Web-based “community memory” where data can be analysed and plotted on dynamic maps.

Under the BrusSense moniker we also work to extend this community-driven approach to monitoring beyond noise pollution by looking at atmospheric pollution and urban microclimates. Our main focus is to establish a proof-of-concept systems and set up a case study in the Brussels Region.

Contact: Matthias Stevens, mstevens@vub.ac.be
Uniform Modularization of Workflow Concerns

Workflow management systems have become a popular means of automating processes in many domains, ranging from high-level business process management to low-level web service orchestration. Realistic workflows consist of several concerns, i.e., parts that are relevant to a particular concept, goal, or purpose, which are combined in order to achieve some desired behavior. State-of-the-art workflow languages provide insufficient means of specifying these concerns as separate modules. This leads to monolithic workflow specifications, in which it is hard to add, maintain, remove or reuse individual concerns. Although most workflow languages allow decomposing workflows into sub-workflows, this mechanism is typically aimed at grouping activities instead of facilitating the independent design, evolution and reuse of concerns. Moreover, a workflow can only be decomposed into sub-workflows according to one dimension, and concerns that do not align with this decomposition end up scattered across the workflow and tangled with one another. Such concerns are called crosscutting concerns, and this problem has given rise to aspect-oriented programming for workflows.

We aim to facilitate the independent design, evolution and reuse of all – i.e., not only crosscutting – workflow concerns, by providing a powerful modularization mechanism that allows concerns to be specified in isolation of each other, and subsequently composing these independently specified concerns according to workflow-specific connection patterns. We develop a workflow framework named Unify, which is based on a number of meta-models that allow expressing both the workflow concerns and their composition. We define a clear semantics for both the workflow concerns (using Petri nets) and their composition (using Graph Transformation). The framework is instantiated towards state-of-the-art workflow languages such as WS-BPEL, and its implementation is compatible with existing tool chains.

RoarVM: A Manycore VM for Smalltalk and Beyond

The RoarVM is our research vehicle developed together with IBM Research in a project on non-deterministic programming for manycore systems. It is the first single-image Smalltalk VM for manycore processors, tested on chips with up to 64 cores. As part of our collaboration with IBM Research the VM was open sourced and it is now available in a version that is compatible with Squeak and Pharo Smalltalk.

From here on our research will focus on bringing support for a wide range of different concurrency models to virtual machines. Today, VMs like the JVM are increasingly popular and host a wide range of different languages. This enables software engineers to solve their problems with domain-specific languages while benefiting from the advances in just-in-time compiler and garbage collection technology. Therefore, we are investigating the design of a multi-language VM for manycore systems. With respect to concurrency, today's virtual machines do not offer sufficient abstraction mechanisms to enable language designers to benefit from the available hardware parallelism easily. Implementing concurrent languages or libraries is tedious since VMs do not expose fundamental concepts like locality or encapsulation. Thus, we investigate how the various forms of hardware restrictions can be abstracted in a VM model. A VM on a multi- or manycore system needs detailed information with respect to locality to enable optimal performance by utilizing the available computational resources but more importantly the available memory bandwidth.

Furthermore, language designer devise a wide range of strategies to control or avoid shared mutable state in all kind of different ways. The Actor model gained increasing attention over the last years, but less restricted models and alternative approaches are used and investigate, too. However, implementing guaranteed encapsulation on top of existing VMs is a challenge either with respect to performance or the complexity of the necessary type systems. Thus, our goal is to provide a flexible model of encapsulation inside the VM.

With the fundamental notions of locality and encapsulation a virtual machine can provide real abstraction from the various different hardware concurrency models and make the next step to provide the necessary mechanisms for languages in a manycore world.
Partners

Universiteit Antwerpen
FUND P
UNIVERSITY OF NAMUR
Katholieke Universiteit Leuven
Université de Mons
UNIVERSITÉ DE LIÈGE
Université de Liège
INRIA, Centre de recherche Lille - Nord Europe
Imperial College London
TU Delft