Vrije Universiteit Brussel

Research Descriptions 2007-2008

System and Software Engineering & Programming Technology Labs

Prof. Dr. Theo D'Hondt
Prof. Dr. Viviane Jonckers
Prof. Dr. Wolfgang De Meuter

Bram Adams

AOP and the Co-Evolution Between Source Code and Build System

My PhD work investigates how co-evolution between build system and source code causes many practical problems for people trying to integrate aspects into their development environment (in its widest sense).

Build systems need to evolve whenever the source code changes, otherwise source code is just plain text. On the other hand, rigid build systems obstruct source code evolution. Because of the radical difference between crosscutting relations and traditional module composition, aspects invalidate many proven workarounds to cope with this co-evolution. For source code configuration e.g., aspects can be assigned to individual source files or at the other extreme to the whole system, but flexibility is restricted by the lack of explicit dependencies between aspects. Incremental compilation on the other hand can be supported in the weaver itself or e.g. by carefully arranging the order of weaving, but again unforeseen side-effects can occur. No best practices have been established yet for five crucial responsibilities of the build system, although these issues are crucial for adoption of AOP.

To experiment with various techniques, we have developed a re(verse)-engineering framework for build systems (MAKAO) and an aspect language for C (Aspicere2). MAKAO exposes the build dependency graph for visualization, querying, filtering, verification and re-engineering. Aspicere2 is built on a logic pointcut language, generic advice and a link-time weaver. Recently, it has been extended with temporal pointcuts (cHALO), based on Charlotte Herzeel's HALO.

Bram.Adams@ugent.be

Build system, AOP for C, reverse engineering, logic/temporal pointcut languages

Dr. Thomas Cleenewerck

Modularizing Language Constructs: A Reflective Approach

Programming languages are continuously growing with new constructs so that programmers can express the problems within the language they are using. In order to grow language implementations more easily, we preserve the decomposition of language implementations into its language constructs by modularizing the syntax and semantics of language constructs in separate modules. The challenge is to preserve the modularization of constructs despite the fact that constructs closely interact with one another. In the linglet transformation system (LTS) the interactions are established by interaction strategies. The strategies capture a specific interaction pattern and are defined as extensions of a specifically tailored meta-object protocol (MOP). As such they can reflect upon the behavior of the different linglets and preserve the modularization of the language constructs.

We continue to explore the advantages modularization by studying language growth. An interesting domain of language research with respect to language growth are domain-specific languages and aspect languages. These languages are challenging to modularize because their constructs have complex invasive behavior. Furthermore, language growth is also inherent in graphical domain-specific languages: model-driven development techniques have made languages a part of the everyday development process. There is a mutual beneficial relationship between the modularization model advocated by LTS and the model-driven community that we are about to explore.

tcleenew@vub.ac.be

Language engineering, domain-specific languages, transformation systems, reuse, separation of concerns

Dr. Pascal Costanza

Context-Oriented Programming

Context-dependent behavior is becoming increasingly important for a wide range of application domains, from pervasive computing to common business applications. Unfortunately, mainstream programming languages do not provide sufficient mechanisms that enable software entities to adapt their behavior dynamically to the current execution context. This leads developers to adopt convoluted designs to achieve the necessary runtime flexibility. Context-oriented Programming addresses this problem by treating context explicitly and providing mechanisms to dynamically adapt behavior in reaction to changes in context, even after system deployment. Such behavioral variations may crosscut the entire system. Together with various researchers and users, we have developed the essential building blocks for Context-oriented Programming in recent years, as described for example in an overview article in Journal of Object Technology (March/April 2008). The research on Context-oriented Programming covers the range from the design and implementation of programming language extensions up to tool support for developing non-trivial context-aware applications.

Pascal.Costanza@vub.ac.be

Dynamic layer activation, dynamic scoping, dynamic software evolution, software composition, meta-programming, reflection

Coen De Roover

Behavioral Program Queries Using Logic Source Code Templates

The use of executable logics to query a program's implementation has gained significant momentum among researchers across software disciplines as diverse as quality assurance, program comprehension and refactoring. The efficacious identification of implementation parts causing faulty, well-known or suboptimal behavior is key to the success of logic meta programming in these disciplines. As similar behavior can be implemented by heterogeneous code, querying a behavioral rather than a syntactic program representation is a seemingly obvious strategy to improve identification efficacy. In reality, the arrival of behavioral program queries to an application programmer's toolbox is still a long way off.

First of all, it is unclear which generic behavioral representation suffices to answer a rich set of user-defined queries. As these representations approximate run-time behavior, special care must be taken in answering queries. Furthermore, the representations' intricate complexity hinders straightforward query definition. The efficacious resolution and straightforward definition of behavioral queries hence requires a specifically tailored executable logic. To ensure straightforward definition, our research introduces source code templates in logic queries to capture the prototypical implementation of the behavior that needs to be identified. Logics of qualified truth are available to demarcate specific execution contexts. To ensure identification efficacy, we resolve templates against behavioral representations. The resolution strategy relies on logics of quantified truth to handle approximations and to quantify the similarity between a template and its matches.

cderoove@vub.ac.be

Logic meta-programming, fuzzy/temporal logic, abstract interpretation, points-to analysis, code templates

Dr. Kris De Schutter

Aspect-Orientation and LMP for Revitalizing Legacy Software

We propose the use of Logic Meta-programming (LMP) and Aspect-oriented Programming (AOP) as a set of tools to tackle the ills of legacy business applications. The dynamics for such applications are characterized by a need for restructuring and integration at a much larger scale than was previously the case. This requires a non-trivial amount of human insight and experience, something which is hampered by a general lack of good documentation of these applications. Putting this software aside is simply not a reasonable option to its stakeholders. We therefore have to make modification of existing applications easier. This includes support for helping with the modification itself, but also to make up for missing knowledge about the system, which is crucial for proper evolution.

By embedding Aspect-oriented Programming in existing business environments we can empower software developers with a flexible toolchain while avoiding a steep learning curve. In using this toolchain, there is no requirement to move away from the existing development techniques; there is only the incentive to work with something that augments them. This can make for a faster turn-around based on available expertise. In addition, Logic Meta-programming can be used for expressing business concepts and architectural descriptions of business applications in a declarative way. This makes it possible to work with applications at a higher level of abstraction, which will allow better architectural descriptions to emerge. By making these descriptions available for practical use we can actively encourage development and understanding thereof.

Kris.De.Schutter@vub.ac.be

Aspect orientation, logic meta-programming, legacy software, maintenance, evolution

Dr. Jessie Dedecker

Languages for Ubiquitous Computing and Ambient Intelligence

Ambient-oriented programming is a programming paradigm whose properties are derived from the characteristics of hardware platforms for mobile computing. Mobile hardware devices are often provided with wireless networks facilities, allowing them to engage in collaboration with their environment. However, the autonomous nature of these devices as well as the volatile connections over their wireless infrastructure has its repercussions on the software that employs them. The most fundamental assumption of the Ambient-Oriented Programming paradigm is that languages should incorporate possible network failures at the heart of their programming model.

AmbientTalk is a distributed programming language that supports this paradigm and is designed as a language laboratory. The language offers a meta-object protocol that enables language designers to experiment with novel programming abstractions. The goal of these abstractions is to contribute to the rapid application development of Ubiquitous Computing and Ambient Intelligent applications.

jededeck@vub.ac.be

Ambient intelligence, ubiquitous computing

Peter Ebraert

Change-Oriented Advanced Round-Trip Engineering

Agile Software Development (AgSD) stresses a highly iterative and incremental development cycle and rapid prototyping in order to anticipate "change" during the different phases of software engineering. There exist however a large number of unaddressed challenges in the tool support for AgSD. We focus on three related issues. First, because certain software systems, such as critical systems, can not be stopped, changes need to be performed directly on running systems. Second, testing plays a centrol role in AgSD. Unit tests are automated pieces of code that invoke a different method and then check assumptions on the behavior of the element currently being tested. It has to be possible to easily write the tests and quickly run them, repeatedly and automatically. Finally, automated testing gives rise to refactoring, which are continuously applied in Agile methods such as XP in order to facilitate adding new functionality or to improve the design quality after a change took place.

Current tool support for AgSD does not sufficiently support runtime change propagation, automated testing or refactorings. Representing changes as first-class objects allows a change management system for alleviating those difficulties. We extend an existing approach called Advanced Round-Trip Engineering (ARTE) — which already provides limited support for runtime change propagation — with such a change management system. Change-Oriented Advanced Round-Trip Engineering is an ARTE approach in which changes can be specified, applied, undone and reasoned about on each of the three ARTE levels.

pebraert@vub.ac.be

Software evolution, dynamic adaptation, round-trip engineering

Sofie Goderis

DEUCE — Declarative User Interface Concerns Extrication

In order to survive in today's highly dynamic marketplace, companies must show a continuous and ever-increasing ability to adapt. This reflects on the adaptability requirements for the supporting software systems. Evolving a software system not only affects the source code responsible for the core application, but also the user interface. A problem with maintaining user interface (UI) logic is that it is entangled with the underlying application logic. The fact that the UI logic is scattered throughout the application logic makes adapting the UI logic and evolving the application logic cumbersome for the programmer.

In our approach we aid a programmer to cope with the complexity of UI development. To do so the UI and application code should be separated as much as possible. This idea has been applied to many software engineering areas. With respect to user interfaces, it has been applied, amongst others, by the Model-View-Controller metaphor and 3-tiered systems. Both approaches however, when put to practice, still result in entangled and scattered code and don't help the developer to cope with several UI concerns separately.

We are currently implementing a framework to support our approach, called DEUCE. The concerns are described declaratively by means of facts and rules. A reasoning mechanism puts the concerns together.

Sofie.Goderis@vub.ac.be

Separation of concerns, declarative meta programming, software evolution

Elisa Gonzalez Boix

Distributed Memory Management in Mobile Ad Hoc Networks

In mobile ad hoc networks, distributed programming is substantially complicated by the intermittent connectivity of the devices in the network and the lack of any centralized coordination facility. Within this context, my research focuses on the repercussions of such hardware phenomena on distributed memory management. Due to the frequent disconnections in mobile networks, references to remote objects that do tolerate network failures are much more suitable as they remain valid during a disconnection. However, because it is impossible to distinguish a transient network failure from a permanent (network or machine) failure, the lifetime of the remote object reference should be limited such that the remote object can eventually be reclaimed if the network failure persists. Memory management based on leasing provides a robust mechanism to manage reclamation of remote objects in mobile ad hoc networks. Rather than providing a general leasing framework, we explore a language approach such that low-level memory management concerns can be abstracted away as much as possible. I have designed dedicated language support that integrates leasing directly into the remote object reference abstraction, leading to the concept of a leased object reference. However, applying the leasing semantics on each remote object reference still places a considerable burden on developers. I am currently investigating software engineering techniques to implement dedicated language support for leasing based on the usage analysis of our language constructs for leasing in mobile ad hoc applications.

egonzale@vub.ac.be

Ambient intelligence, distributed memory management, ambient-oriented programming, language abstractions

Kris Gybels

Aspect-Oriented Programming with a SOUL

My research interests are the foundations of Logic Meta Programming (LMP) and its applications to Aspect-Oriented Software Development. Logic meta programming is the use of a programming language based on the logic paradigm (such as SOUL) for writing programs about programs, such as design recovery tools, code refactoring browsers or "pointcuts" in Aspect-Oriented Programs (AOP). While the use of a logic language for meta programs that are not necessarily written in a logic language has advantages due to the declarative nature of logic programs, it also raises a number of research questions such as how to properly represent programs as data in the logic language, and how to integrate the logic and non-logic language to create a linguistic symbiosis. Ideally, such a symbiosis is transparent, allowing interaction between the two languages while not making it obvious that a boundary between languages is crossed. I have applied this to business rule integration, where it allows the object-oriented parts of the software to be freely replaced by logic rules and vice-versa. My application of LMP to AOP revolves around the design of pointcut languages, which are used in AOP to describe which runtime events to intercept in a program. I have created an advanced pointcut language, CARMA, which, by using sophisticated features of LMP, helps decrease the coupling of these descriptions to the rest of the program. I am doing further research on the design of this language, and the question of whether techniques such as Inductive Logic Programming can be used to automatically mine programs for pointcuts.

Kris.Gybels@vub.ac.be

Aspect-oriented software development, logic meta-programming, linguistic symbiosis

Charlotte Herzeel

Reflective Foundations

In the past, I have developed a logic-based aspect language called HALO that incorporates temporal reasoning. It eases writing pointcuts that capture part of the execution history of a program, and expands on the theme of related approaches such as tracematches and state-based aspects. HALO is especially powerful due to its support for language symbiosis and reflective features which allow the aspect language to invoke methods of the base language, and vice versa.

My subsequent investigations on applying HALO to event-based systems, parallel programming models and transactional systems revealed that a better understanding of computational reflection as a foundation for my work is necessary. Unfortunately, computational reflection is still a poorly understood concept: It is perceived as hard to implement, understand and use. The earliest, but still most complete attempt to come up with a conceptual framework for computational reflection is 3-Lisp, and I am currently working on framing ideas from 3-Lisp and its successors in a modern setting. Though seemingly forgotten, pragmatic subsets of reflective programming systems are actually in wide use and we believe that a better understanding of the 3-Lisp model and its successors is a key to better understand programming techniques such as meta-object protocols, open implementation and aspect-oriented programming in general.

caherzee@vub.ac.be

Reflection, Lisp

Dr. Isabel Michiels

BEHAVE — Verifying and Documenting Design Invariants in Software

Today's software systems have to evolve rapidly, especially because of the quickly evolving business requirements. Supporting this process of change has up until now mainly resulted in tool support for modularizing systems better and localizing those parts sensitive to change.

But what about those parts of a software system that are not allowed to change when software evolves? In our approach, we offer support for documenting and verifying design invariants of a system. Design invariants represent the laws of your software system that must continue to hold through every evolution cycle. They are difficult to capture as they cross-cut an entire system. As an example of an invariant, consider a banking application: it should always be the case that, if a transaction error occurs, the outcome is in favor of the customer.

BEHAVE offers a platform to specify and verify a high-level behavioral model representing such a design invariant. The main contribution of our approach is that the behavioral models are specified using a declarative formalism which renders the models machine-verifiable but also understandable to developers. More specifically, BEHAVE lets you specify a behavioral model of the invariant in terms of selective high-level run-time events using temporal logic programming.

Isabel.Michiels@vub.ac.be

Crosscutting concerns, declarative meta-programming, dynamic analysis, lightweight verification

Stijn Mostinckx

Fault-Tolerance Abstractions in Ambient-Oriented Programming

The far-reaching introduction of small, mobile and often dedicated computer hardware in our everyday life has a significant impact on the development of distributed systems. As these devices are becoming an integral part of our everyday environment, it becomes ever more interesting to develop software that leverages the network formed by these devices.

One of the key issues when developing such "ambient-oriented" software, is the ability for it to cope with the inevitable failures encountered as devices go in and out of communication range. Dealing with such failures requires a family of dedicated abstractions offering transactional guarantees in a fleeting environment.

The Ambient-Oriented Exception Handling Model provides a basis from which such advanced abstractions can be constructed.

smostinc@vub.ac.be

Ambient-oriented programming, exception handling, asynchronous communication

Christophe Scholliers

Software Engineering Techniques for Data Sharing in a Mobile Network

Current-day applications for mobile phones and PDAs are often limited to miniature versions of standard desktop applications such as browsers, calendar and word processing applications. Only a fraction of the available applications allows the mobile devices to interact directly with their environment. One of the reasons for this is that even for the simplest interactions between mobile devices, the implementation needs to deal with a lot of problems that are inherent to a pervasive computing environment (e.g. frequent disconnections, dynamic discovery, etc.).

Last year, I have been involved in the development of (a concrete implementation of) the Fact Space Model, an extension to the tuple space model which provides fine-grained control over the effects of disconnections. Using a declarative language, every device can specify how it will adjust its behavior in response to dynamic changes in its environment. Currently, I am working on an extension of this model that allows applications to work on shared and replicated data. The use of weak data replication in a mobile environment ensures that there is no need to stop an ongoing application when certain data cannot be synchronized.

cfscholl@vub.ac.be

Ambient computing, data replication, language design, CRIME

Stijn Timbermont

Modular Virtual Machines for Ambient Intelligence

In an Ambient Intelligent setting, various kinds of hardware are involved, each with its own characteristics. To allow abstraction over the different devices, a standard virtual machine approach is not feasible because it is not possible to abstract over all the devices at the same time. Instead, each device will require an individually adapted virtual machine.

This research proposes a new way to develop virtual machines for Ambient Intelligence. Instead of manually modifying the virtual machine for of a particular device, the virtual machine can be generated by composing reusable modules. The first step is to determine the set of modules that can be used to build a virtual machine. The next step is to find implementation and composition techniques that guarantee the correctness and efficiency of the generated virtual machine.

Stijn.Timbermont@vub.ac.be

Ambient intelligence, virtual machines, language engineering

Jorge Vallejos

Distributed Context-Dependent Adaptations

Within the domain of Ambient Intelligence (AmI), my research focuses on the capacity of software applications to adapt to their dynamically reconfigurable environments. The main idea is to identify the properties of context-dependent adaptations and establish a set of specific requirements of distribution for such adaptations, derived from concrete AmI scenarios. In my research, I claim that context-dependent adaptations occur dynamically and within a delimited scope of action. In addition, these adaptations should be consistently combined with the default behaviour of the application, and clearly modularised to avoid the entanglement between the adaptations and the application behaviour. To also cope with the effects of distribution, behavioural adaptations should take into account the context of all the applications involved in an interaction, have an unambiguous scope of action even in the presence of concurrent interactions, and finally protect the privacy of the interacting applications. Currently I investigate amongst others: - Context-dependent adaptations using role-based models. - Rule-based systems for context reasoning and role selection. - Role-based communications. - Combinations of actor and role models for the development of context-dependent applications.

jvallejo@vub.ac.be

Context-oriented programming, ambient-oriented programming, actors, layers, roles, symbiosis

Tom Van Cutsem

Ambient References: Object Designation in Mobile Ad Hoc Networks

My research is part of the "ambient-oriented programming" research domain of the Programming Technology Lab. In the context of my PhD research, I co-designed and developed the "ambient-oriented" programming language AmbientTalk/2. It is an object-oriented, concurrent and distributed language designed specifically for writing applications for so-called mobile ad hoc networks (MANETs). A MANET is a network composed of mobile devices that communicate through wireless communication links. In such networks, where there is only intermittent connectivity between nodes and infrastructure is very scarce, traditional distributed programming language abstractions do not scale. The goal of our research is to develop novel distributed coordination abstractions that raise the level of abstraction for the programmer.

In particular, I am investigating how the classic abstraction of a remote object reference (well-known in distributed object systems such as e.g. Java RMI) can be extended to be scalable in a MANET. We do this by reusing techniques developed for loosely-coupled event-driven systems and by casting them in an object-oriented form. The concrete language abstractions developed for my doctoral thesis are termed "ambient references". They are an object-oriented anonymous and asynchronous communication channel for many-to-many interactions between objects in a wireless, mobile network.

Aside from ambient-oriented programming, my research interests include programming language design in general, with a particular emphasis on computational reflection and object composition abstractions.

tvcutsem@vub.ac.be

Coordination abstractions, events, actors, concurrent and distributed languages, AmbientTalk

Mathieu Braem

Aspect-Oriented Workflow Patterns for Web Service Composition

In current composition languages for web services, there is insufficient support to explicitly separate crosscutting concerns, which leads to compositions that are hard to maintain or evolve. A similar problem in object-oriented languages is being tackled by aspect-oriented programming, and some work has been started to apply these techniques to web service composition languages as well. We identified some key problems with these approaches and formulated some improvements on the current work. We started implementing these features in Padus, an aspect-oriented language to instrument WS-BPEL, the most well known language for web service composition.

In future work we plan to further investigate aspect-oriented programming in a workflow context and aim to define an extension to generic workflow patterns, that can in the end be mapped back to concrete languages, such as e.g. YAWL and WS-BPEL.

mbraem@vub.ac.be

Aspect-oriented programming, workflow languages, web service composition

Bruno De Fraine

Improving Language Facilities for the Deployment of Reusable Aspects

In the AOSD community, there is an increasing interest in the development of reusable implementations of typical crosscutting concerns, such as security, synchronization, profiling, etc. To employ a reusable aspect in a concrete application, deployment logic has to be written that specifies where and how to apply the new behavior, and how the interaction with the base application and the other aspects in the system is organized. Although the deployment logic might specify a crucial part of the application functionality, current AOP approaches provide only inferior means for its specification. We identify a number of issues regarding the reuse of deployments, their dynamic invocation and their integration with the rest of the system.

The Eco AOP model addresses these shortcomings by organizing deployment logic as procedures that employ different reusable building blocks of aspectual behavior as first-class values. More concretely, this means that pointcuts, advices and combinations strategies are passable and returnable as parameters to and from pieces of deployment logic, and that the deployment logic can be dynamically invoked with these entities as runtime values. EcoSys realizes the Eco model as Java AOP framework, and allows the development of deployment logic as standard Java code. Contrary to other AOP frameworks, EcoSys exploits the Java 5 Generics feature to provide static type checking of deployments, similar to language-based AOP approaches. As a byproduct of this research, a novel type system for the AOP pointcut/advice mechanism is being developed.

Bruno.De.Fraine@vub.ac.be

Aspect-oriented programming, language engineering, software reuse, generic typing

Dr. Dirk Deridder

Concept-Centric Coding

Concept-Centric Coding (C3) aims at providing programmer support to cope with software evolution in an agile context. It prescribes to complement an application's source code with explicit knowledge about the application and its problem domain. This domain knowledge is described in a concept graph, which is more than a mere ontology. In addition to the domain concepts, it contains the source code entities of the application itself, thereby connecting the application's domain knowledge with its implementation, and keeping this link up to date when the application evolves. Moreover the domain knowledge actively contributes to the runtime functionality of the application. Hence it becomes possible to adapt an applications' behavior by modifying the corresponding domain concepts.

The 'Concept to Code Browser' (CoBro), is the core tool suite supporting the C3 approach. It is closely integrated with Smalltalk and the VisualWorks development environment. Moreover it is implemented in symbiosis with Smalltalk allowing a programmer to transparently invoke and manipulate concepts as if they were plain Smalltalk objects. This enables a non-obtrusive synergy between the concept level and the code level. Part of the power of CoBro is realized by its (partial) metacircular implementation. In particular, CoBro itself is implemented in terms of domain concepts using the C3 approach. Consequently, it can be adapted, even at runtime, by changing its concepts, thus leading to a highly extensible and flexible tool-base.

Dirk.Deridder@vub.ac.be

Software evolution, domain modeling, flexible architectures, meta-programming, software variability, co-evolution

Oscar González

High-Level Business Processes Monitoring, Measurement and Control

Aiming to improve their flexibility, companies organize themselves around (partially automated) business processes, which facilitate the integration of human and technological resources and manage the flow and control between them. In order to improve processes and assure quality, it is important to install a monitoring activity. Even though several monitoring solutions are already available, these approaches are typically implemented with low-level mechanisms, which require specific knowledge about the process implementation that business experts do not necessarily have. Furthermore, even for users with technical skills the implementation of monitoring requirements remains a tedious and complex task because the monitoring statements crosscut the executable specifications and involve information that is not necessarily available in one specific location.

The main goal of this research is the definition of a high-level domain-specific language for expressing monitoring, measurement and control specifications on business processes. The language is targeted at experts that need to assess the quality of a business process. Therefore we provide specialized abstractions at the domain level instead of at the code level. Among others it contains constructs for gathering basic real time and historic process measurements, for creating new high-quality concepts, and for applying control actions over the process domain and monitoring domain. Eventually the goal is to support domain experts in the creation of monitoring requirements without having to know the specifics of the underlying implementation.

ogonzale@vub.ac.be

Business process management, domain-specific languages, aspect-oriented programming, monitoring, measurement, control

Niels Joncheere

Workflows for Client/Server Based Computer Aided Engineering

Computer aided engineering (CAE) deals with optimizing products through analysis and simulation. For example, CAE is used for determining properties of physical objects based on their design parameters (such as determining the maximum stress level of a bridge based on the thickness of its beams). In CAE, engineers specify simulations by describing workflows whose activities correspond to the application of certain algorithms on certain data sets. Each workflow is typically executed a number of times with varying parameters in order to determine which parameter values yield an optimal result. Although there are already languages available for specifying such CAE workflows, they lack a number of important features, which mostly boil down to a lack of separation of concerns: each workflow is a single monolithic specification, with no support for modularizing sub-workflows or crosscutting concerns. They also lack support for client/server scenarios in which algorithms are available to workflows as (web) services.

The goal of our research is to design and implement a service based workflow system that better addresses the requirements of the CAE community by providing a simple but powerful modularization mechanism, which allows subdividing workflows in sub-workflows. These sub-workflows can then be called from other workflows using block tasks, or can be attached to workflows in an aspect-oriented way. Furthermore, we have identified which control flow, resource and data patterns are of interest in the CAE context, and will design a language that natively supports these patterns.

njonchee@vub.ac.be

Aspect-oriented programming, computer aided engineering, modularization, web services, workflow

Dr. Andy Kellens

Verification and Co-Evolution of Structural Regularities

The topic of this research is the documentation, verification and co-evolution of structural regularities in the source code of software. We put forward the model of Intensional Views as a generic approach to document different kinds of structural regularities as well as the interactions between such regularities. By checking conformance of these constraints with respect to the source code, we can detect evolution conflicts and provide support for co-evolving the documentation of the structural regularities and the source code of the system. To provide support for this model of intensional views, we implemented IntensiVE, a tool suite that extends the IDE of VisualWorks Smalltalk.

In order to illustrate the broad applicability of our approach, we documented a variety of different types of structural regularities and assessed to which extent they are supported by our model and tool suite. More specifically, our approach supports naming conventions, coding conventions, programming idioms, design patterns, anti-patterns and framework documentation. Our work is also applicable in the context of aspect-oriented programming. We showed that the fragile pointcut problem, an open evolution problem in the AOSD community, is caused by the tight coupling of aspect-oriented programs with the structural regularities that are prevalent in an application. To address this problem, we integrated our approach with an aspect-language and demonstrated how this approach can alleviate the fragility of pointcuts.

Andy.Kellens@vub.ac.be

Co-evolution, structural regularities, aspect-oriented programming, tool support

Eline Philips

Orchestration of Mobile Devices

My research has its roots in the field of ambient computing and more specific the coordination of ambient devices in a pervasive environment. The recent and ongoing miniaturization of computational devices and the standardization of bluetooth and wifi have caused a revolution in every day life of the western society. These co-operating ambient devices, have characteristics which are significantly different from conventional devices. For instance the connections in a MANET can not be assumed stable and furthermore is the use of a centralized server practically impossible.

In the past, I was one of the developers of a logic coordination language CRIME, which gives the user fine-grained control over the effects of disconnection. This language has incorporated a set of primitives in the core of its design in order ease the implementation of context aware applications. My current research investigates the extension of the CRIME programming model in order to specify the orchestration between several mobile devices and is inspired by workflow languages.

ephilips@vub.ac.be

Ambient computing, service oriented architectures, workflows

Mario Sánchez

Executable Models for Constructing Extensible Workflow Applications

Workflow languages and applications are currently used in a lot of contexts where there is a need to coordinate the work performed by humans or the execution of several applications. The problem with current languages to describe workflows, is that each one has certain limitations in its expressiveness and in its capacity to evolve. This creates problems especially to maintain and introduce new requirements into some applications. Our work is then directed towards the development of new models and tools to build very extensible workflow-based applications.

Our proposal is based on three main ideas. The first one is that in workflows it is possible to identify and separate concerns, in a similar way to what is done in aspect-oriented programming; some sample concerns are Control, Time, Data and Resources. The second idea is that these concerns can be implemented using extensible, executable models, which should run in a synchronized way. The third idea is that the executable models can be implemented using a coordination model based on synchronized state machines. The base element of this coordination model is called an 'open object': it offers a synchronization mechanism based on event passing and method calls, as well as several advantages towards extensibility. Following these ideas, we are currently working on the detailed specification of the open objects' structure and behavior, and the definition of dedicated models to represent some of the more common concerns

marsanch@ssel.vub.ac.be

Workflow languages, coordination models, separation of concerns

Dr. Ragnhild Van Der Straeten

Inconsistency Management in Model-Driven Engineering

One of the important challenges in current-day MDE is the ability to manage model inconsistencies. When designing models in a collaborative and distributed setting, it is very likely that inconsistencies in and between the models will arise because: (i) different models may be developed in parallel by different persons; (ii) the interdependencies between models may be poorly understood; (iii) the requirements may be unclear or ambiguous at an early design stage; (iv) the models can be incomplete because some essential information is still unknown. In a model evolution context the ability to deal with inconsistent models becomes even more crucial as models are continuously subject to changes.

The global objective of this research is to develop an inconsistency management framework for the definition, detection and handling of inconsistencies in the context of object-oriented models with special focus on UML models. The precise definition and detection of inconsistencies is enabled using declarative formalisms. Description Logics and graph transformations have been investigated. Currently the focus of our research is on inconsistency resolution. Different relations can exist between inconsistency resolutions. The nature of these relations and the kind of application under construction can lead to different resolution strategies. As a proof of concept we implement our framework called RACOoN (originally implemented in the UML case-tool Poseidon) as an Eclipse plug-in.

rvdstrae@vub.ac.be

Inconsistency management, model-driven engineering, logic-based languages, software evolution

Dennis Wagelaar

Platform Ontologies for the Model Driven Architecture

The Model-Driven Architecture (MDA) allows for the deployment of software applications on a variety of different platforms. The MDA models software in a platform-independent model (PIM) that is transformed to a platform-specific model (PSM), given a platform model (PM). Currently, models are transformed directly from PIM to PSM, without using a platform model. The model transformations implicitly assume a platform model. This makes it much easier to write model transformations, since one only has to deal with the limited scope of targeting a single, assumed platform. It is unclear, however, whether a model transformation can be used for other platforms. The only safe assumption is that each targeted platform requires its own corresponding set of model transformations.

We propose a separate platform model, which can be used to automatically select and configure a number of reusable model transformations for a given platform. This platform model is expressed in OWL-DL. Platform constraints can be defined for each model transformation. This way, the model transformations can be reused over a well-defined class of platforms. Concrete platforms are modelled separately and refer to the same platform vocabulary model. An automatic DL reasoner can be used to verify whether a concrete platform satisfies the platform constraints of a model transformation. In addition, it can determine which platform constraint is most platform-specific.

Dennis.Wagelaar@vub.ac.be

Model-driven engineering/architecture, software product lines, platform modeling, ambient intelligence

Andrés Yie

Multi-Step Concern Refinement

A Model-Driven Software Product Line (MD-SPL) uses metamodels, models, and transformations to create a family of products. These products are generated from a high-level model (or business model), which is refined using a sequence of model-to-model transformations. The sequence of transformations is a Model Refinement Line (MRL). However, an MD-SPL must evolve to introduce new crosscutting concerns, such as security or logging, in the generated applications. Since the MRL transformations are fragile and complex, the original MRL must be preserved or reused when the new concerns are added to the MD-SPL.

We propose an approach to add new crosscutting concerns to the applications produced, while keeping the original MRL as unchanged as possible. This approach adds a new MRL that refines a high-level model of the new concern. This concern model is related with the application business model in the original MRL with a set of pointcuts expressing the relationships among them. Afterwards, the concern model refinement is performed in parallel with the original MRL. In every refinement step, the application model, the concern model, and the pointcuts are refined. Finally, the low-level application and concern models are composed, and the application is generated.

ayiegarz@ssel.vub.ac.be

Model-driven engineering, software product lines, aspect-oriented software development

PhD Dissertations

A Goal-Driven Approach for Documenting and Verifying Design Invariants
Isabel Michiels, 2007

Modularizing Language Constructs: A Reflective Approach
Thomas Cleenewerck, 2007

Maintaining Causality Between Design Regularities and Source Code
Andy Kellens, 2007

Connecting High-Level Business Rules with Object-Oriented Applications: An Approach Using Aspect-Oriented Programming and Model-Driven Engineering
María Agustina Cibrán, 2007

Advanced Round-Trip Engineering: An Agile Analysis-Driven Approach for Dynamic Languages
Ellen Van Paesschen, 2006

A Concept-Centric Environment for Software Evolution
Dirk Deridder, 2006

Ambient-Oriented Programming
Jessie Dedecker, 2006

Modularizing Advanced Transaction Management
Johan Fabry, 2005

Inconsistency Management in Model-Driven Engineering: An Approach Using Description Logics
Ragnhild Van Der Straeten, 2005

Integrative Composition of Program Generators
Johan Brichau, 2005

Combining Aspect-Oriented and Component-Based Software Engineering
Wim Vanderperren, 2004

Hybrid Aspects for Integrating Rule-Based Knowledge and Object-Oriented Functionality
Maja D'Hondt, 2004

Move Considered Harmful: A Language Design Approach to Mobility and Distribution for Open Networks
Wolfgang De Meuter, 2004

Progressive Mobility
Luk Stoops, 2004

Creation of an Intelligent Concurrency Adaptor In Order To Mediate the Differences Between Conflicting Concurrency Interfaces
Werner Van Belle, 2003

PhD Dissertations

Automated Support for Framework-Based Software Evolution
Tom Tourwe, 2002

Component Composition Using Composition Patterns and Scenarios
Bart Wydaeghe, 2001

UML-SDL Round-Trip Engineering Through Incremental Translation of Changes
Kurt Verschaeve, 2001

A Logic Meta-Programming Approach to Support the Co-Evolution of Object-Oriented Design and Implementation
Roel Wuyts, 2001

Automating Architectural Conformance Checking By Means Of Logic Meta-Programming
Kim Mens, 2000

Multi-Paradigm Design
James O. Coplien, 2000

A Formal Foundation for Object-Oriented Software Evolution
Tom Mens, 1999

An Approach to Incermental Fuzzy Modelling of Dependencies in Complex Physical Systems
Kamreddine Ouliddren, 1998

A Novel Approach to Architectural Recovery in Evolving Object-Oriented Systems
Koen De Hondt, 1998

Type-Oriented Logic Meta-Programming
Kris De Volder, 1998

Documenting Reuse and Evolution with Reuse Contracts
Carine Lucas, 1997

ZYPHER — Tailorability as a Link from Object-Oriented Software Engineering to Open Hypermedia
Serge Demeyer, 1996

Open Design of Object-Oriented Languages: A Foundation for Specialisable Reflective Language Frameworks
Patrick Steyaert, 1994

The Synthesis of "Safe" Fuzzy Controllers Based on Reinforcement Learning
Ann Nowé, 1994

From Specification to Implementation
Ludo Cuypers, 1993

Contact Information

Programming Technology Lab *(Prof. Dr. Theo D'Hondt, Prof. Dr. Wolfgang De Meuter)*
Vrije Universiteit Brussel, DINF-PROG, Pleinlaan 2, B-1050 Brussels, Belgium
http://prog.vub.ac.be/

System and Software Engineering Lab *(Prof. Dr. Viviane Jonckers)*
Vrije Universiteit Brussel, INFO-SSEL, Pleinlaan 2, B-1050 Brussels, Belgium
http://ssel.vub.ac.be/

This brochure was created in the context of the
Interuniversity Attraction Poles Programme - Belgian State - Belgian Science Policy
http://www.belspo.be/