3rd international software language engineering conference (SLE) : pre-proceedings, October 12-13, 2010, Eindhoven, the Netherlands

Eindhoven University of Technology

Department of Mathematics and Computer Science

3rd International Software Language Engineering Conference

Pre-Proceedings

October 12-13, 2010

Eindhoven, the Netherlands

Preface

We are pleased to present the proceedings of the Third International Conference on Software Language Engineering (SLE 2010). The conference will be held in Eindhoven, the Netherlands during October 12-13, 2010 and will be co-located with The Ninth International Conference on Generative Programming and Component Engineering (GPCE'10), and The Workshop on Feature-Oriented Software Development (FOSD).

An important goal of SLE is to integrate the different sub-communities of the software-language-engineering community to foster cross-fertilization and strengthen research overall. The Doctoral Symposium at SLE 2010 contributes towards these goals by providing a forum for both early and late-stage PhD students to present their research and get detailed feedback and advice from other researchers.

The SLE conference series is devoted to a wide range of topics related to artificial languages in software engineering. SLE is an international research forum that brings together researchers and practitioners from both industry and academia to expand the frontiers of software language engineering.

SLE's foremost mission is to encourage and organize communication between communities that have traditionally looked at software languages from different, more specialized, and yet complementary perspectives. SLE emphasizes the fundamental notion of languages as opposed to any realization in specific technical spaces. In this context, the term "software language" comprises all sorts of artificial languages used in software development including general-purpose programming languages, domain-specific languages, modeling and meta-modeling languages, data models, and ontologies. Software language engineering is the application of a systematic, disciplined, quantifiable approach to the development, use, and maintenance of these languages. The SLE conference is concerned with all phases of the lifecycle of software languages; these include the design, implementation, documentation, testing, deployment, evolution, recovery, and retirement of languages. Of special interest are tools, techniques, methods, and formalisms that support these activities. In particular, tools are often based on, or automatically generated from, a formal description of the language. Hence, the treatment of language descriptions as software artifacts, akin to programs, is of particular interest - while noting the special status of language descriptions, and the tailored engineering principles and methods for modularization, refactoring, refinement, composition, versioning, co-evolution, and analysis that can be applied to them.

The response to the call for papers for SLE 2010 was very enthusiastic. We received 79 full submissions from 108 initial abstract submissions. From these submissions, the Program Committee (PC) selected 25 papers: 17 full papers, five short papers, and two tool demonstration papers, resulting in an acceptance rate of 32%. To ensure the quality of the accepted papers, each submitted paper was reviewed by at least three PC members. Each paper was discussed in detail during the electronic PC meeting. A summary of this discussion was prepared by members of the PC and provided to the authors along with the reviews.

SLE 2010 would not have been possible without the significant contributions of many individuals and organizations. We are grateful to the organizers of GPCE 2010 and FOSD 2010 for their close collaboration and management of many of the logistics. This will allow us to offer SLE participants the opportunity to take part in three high-quality research events in the domain of software engineering. We also wish to thank our sponsors, ASML, InfoSupport, Jacquard, NWO, Software Improvement Group and Sogeti.

The SLE 2010 Organizing Committee, the Local Chairs, and the SLE Steering Committee provided invaluable assistance and guidance. We are also grateful to the PC members and the additional reviewers for their dedication in reviewing the large number of submissions. We also thank the authors for their efforts in writing and then revising their papers and we thank Springer for publishing the papers and the post-proceedings.

Mark van den Brand	**Brian Malloy**	**Steffen Staab**
General chair	Program co-chair	Program co-chair
Eindhoven University of Technology	Clemson University	University of Koblenz-Landau
The Netherlands	USA	Germany

Table of Contents

Martin Erwig: A Language for Software Variation Research

Managing variation is an important problem in software engineering that takes different forms, ranging from version control and configuration management to software product lines. In this paper, I present our recent work on the choice calculus, a fundamental representation for software variation that can serve as a common language of discourse for variation research, filling a role similar to lambda calculus in programming language research. After motivating the design of the choice calculus and sketching its semantics, I will discuss several potential application areas.

Emma Söderberg and Görel Hedin: Automated Selective Caching for Reference Attribute Grammars

Reference attribute grammars (RAGs) can be used to express semantics as super-imposed graphs on top of abstract syntax trees (ASTs). A RAG-based AST can be used as the in-memory model providing semantic information for software language tools such as compilers, refactoring tools, and meta-modeling tools. RAG performance is based on dynamic attribute evaluation with caching.

Caching all attributes gives optimal performance in the sense that each attribute is evaluated at most once. However, performance can be further improved by a selective caching strategy, avoiding caching overhead where it does not pay off. In this paper we present a profiling-based technique for automatically finding a good caching configuration. The technique has been evaluated on a generated Java compiler, compiling programs from the Jacks test suite and the DaCapo benchmark suite.

Christoff Bürger, Sven Karol, Christian Wende and Uwe Aßmann: Reference Attribute Grammars for Metamodel Semantics

While current metamodelling languages are well-suited for the structural definition of abstract syntax and metamodelling platforms like the Eclipse Modelling Framework (EMF) provide various means for the specification of a textual or graphical concrete syntax, techniques for the specification of model semantics are not as matured. Therefore, we propose the application of reference attribute grammars (RAGs) to alleviate the lack of support for formal semantics specification in metamodelling. We contribute the conceptual foundations to integrate metamodelling languages and RAGs, and present *JastEMF* --- a tool for the specification of EMF metamodel semantics using JastAdd RAGs. The presented approach is exemplified by an integrated metamodelling example. Its advantages, disadvantages and limitations are discussed and related to metamodelling, attribute grammars (AGs) and other approaches for metamodel semantics.

Adrian Johnstone and Elizabeth Scott: Modelling GLL parser implementations

We describe the development of space-efficient implementations of GLL parsers, and the process by which we refine a set-theoretic model of the algorithm into a practical parser generator that creates practical parsers. GLL parsers are recursive descent-like, in that the structure of the parser's code closely mirrors the grammar rules, and so grammars (and their parsers) may be debugged by tracing the running parser in a debugger. While GLL *recognisers* are straightforward to describe, full GLL parsers present technical traps and challenges for the unwary. In particular, naïve implementations based closely on the theoretical description of GLL can result in data structures that are not practical for grammars for real programming language grammars such as ANSI-C. We develop an equivalent formulation of the algorithm as a high-level set-theoretic model supported by table-based indices, in order to then explore a set of alternative implementations which trade space for time in ways which preserve the cubic bound.

Markus Herrmannsdoerfer, Daniel Ratiu and Maximilian Koegel: Metamodel Usage Analysis for Identifying Metamodel Improvements

Modeling languages raise the abstraction level at which software is built by providing a set of constructs tailored to the needs of their users. Metamodels define their constructs and thereby reflect the expectations of the language developers about the use of the language. In practice, language users often do not use the constructs provided by a metamodel as expected by language developers. In this paper, we advocate that insights about how constructs are used can offer language developers useful information for improving the metamodel. We define a set of usage and improvement patterns to characterize the use of the metamodel by the built models. We present our experience with the analysis of the usage of seven metamodels (EMF, GMF, UNICASE) and a large corpus of models. Our empirical investigation shows that we identify mismatches between the expected and actual use of a language that are useful for metamodel improvements.

Benjamin Braatz and Christoph Brandt: Domain-Specific Modelling Languages with Algebraic Graph Transformations on RDF

Domain-specific modelling langugages (DSMLs), which are tailored to the requirements of their users, can significantly increase the acceptance of formal (or at least semi-formal) modelling in scenarios where informal drawings and natural language descriptions are predominant today. We show in this paper how the Resource Description Framework (RDF), which is a standard for the fundamental data structures of the Semantic Web, and algebraic graph transformations on these data structures can be used to realise the abstract syntax of such DSMLs. We examine a small DSML for IT infrastructures as an application scenario. From this scenario, we derive distributed modelling, evolution of language definitions, migration of legacy models and integration of modelling languages as key requirements for a DSML framework. RDF and transformation rules are then used to provide a solution, which meets these requirements, where all kinds of modifications—from simple editing steps via model migration to language integration—are realised by the single, uniform formalism of algebraic graph transformation.

Kacper Bąk, Krzysztof Czarnecki and Andrzej Wąsowski: Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled

We present *Clafer*, a meta-modeling language with first-class support for feature modeling. We designed Clafer as a concise notation for meta-models, feature models, mixtures of meta- and feature models (such as components with options), and models that couple feature models and meta-models via constraints (such as mapping feature configurations to component configurations or model templates). Clafer also allows arranging models into multiple specialization and extension layers via constraints and inheritance. We identify four key mechanisms allowing a meta-modeling language to express feature models concisely and show that Clafer meets its design objectives using a sample product line. We evaluated Clafer and how it lends itself to analysis on sample feature models, meta-models, and model templates of an E-Commerce platform.

Peter Pirkelbauer, Damian Dechev and Bjarne Stroustrup: Support for the Evolution of C++ Generic Functions

The choice of requirements for an argument of a generic type or algorithm is a central design issue in generic programming. In the context of C++, a specification of requirements for a template argument or a set of template arguments is called a *concept*.

In this paper, we present a novel tool, TACE[1], designed to help programmers understand the requirements that their code de facto imposes on arguments and help simplify and generalize those through comparisons with libraries of well-defined and precisely-specified concepts. TACE automatically extracts requirements from the body of template functions. These requirements are expressed using the notation and semantics developed by the ISO C++ standards committee. TACE converts implied requirements into concept definitions and compares them against concepts from a repository. Components of a well-defined library exhibit commonalities that allow us to detect problems by comparing requirements from many components: Design and implementation problems manifest themselves as minor variations in requirements. TACE points to source code that cannot be constrained by concepts and to code where small modifications would allow the use of less constraining concepts. For people who use a version of C++ with concept support, TACE can serve as a core engine for automated source code rejuvenation.

[1] template analysis and concept extraction

Davide Di Ruscio, Ralf Lämmel and Alfonso Pierantonio: Automated co-evolution of GMF editor models

The Eclipse Graphical Modeling (GMF) Framework provides the major approach for implementing visual languages on top of the Eclipse platform. GMF relies on a family of modeling languages to describe abstract syntax, concrete syntax as well as other aspects of the visual language and its implementation in an editor. GMF uses a model-driven approach to map the different GMF models to Java code. The framework, as it stands, lacks support for evolution. In particular, there is no support for propagating changes from the domain model (i.e., the abstract syntax of the visual language) to other editor models. We analyze the resulting co-evolution challenge, and we provide a solution by means of GMF model adapters, which automate the propagation of domain-model changes. These GMF model adapters are special model-to-model transformations that are driven by difference models for domain-model changes.

Markus Herrmannsdoerfer, Sander D. Vermolen and Guido Wachsmuth: An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models

Modeling languages and thus their metamodels are subject to change. When a metamodel is evolved, existing models may no longer conform to it. Manual migration of these models in response to metamodel evolution is tedious and error-prone. To significantly automate model migration, operator-based approaches provide reusable coupled operators that encapsulate both metamodel evolution and model migration. The success of an operator-based approach highly depends on the library of reusable coupled operators it provides. In this paper, we thus present an extensive catalog of coupled operators that is based both on a literature survey as well as real-life case studies. The catalog is organized according to a number of criteria to ease assessing the impact on models as well as selecting the right operator for a metamodel change at hand.

Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio: JTL: a bidirectional and change propagating transformation language

In Model Driven Engineering bidirectional transformations are considered a core ingredient for managing both the consistency and synchronization of two or more related models. However, while non-bijectivity in bidirectional transformations is considered relevant, current languages still lack of a common understanding of its semantic implications hampering their applicability in practice.

In this paper, the Janus Transformation Language (JTL) is presented, a bidirectional model transformation language specifically designed to support non-bijective transformations and change propagation. In particular, the language propagates changes occurring in a model to one or more related models according to the specified transformation regardless of the transformation direction. Additionally, whenever manual modifications let a model be non reachable anymore by a transformation, the *closest* model which approximate the ideal source one is inferred. The language semantics is also presented and its expressivity and applicability are validated against a reference benchmark. JTL is embedded in a framework available on the Eclipse platform which aims to facilitate the use of the approach, especially in the definition of model transformations.

Abraham Bernstein: Software Engineering and the Semantic Web: A match made in heaven or in hell?

The Semantic Web provides models and abstractions for the distributed processing of knowledge bases. In Software Engineering endeavors such capabilities are direly needed, for ease of implementation, maintenance, and software analysis.

Conversely, software engineering has collected decades of experience in engineering large application frameworks containing both inheritance and aggregation. This experience could be of great use when, for example, thinking about the development of ontologies.

These examples---and many others---seem to suggest that researchers from both fields should have a field day collaborating: On the surface this looks like a match made in heaven. But is that the case?

This talk will explore the opportunities for cross-fertilization of the two research fields by presenting a set of concrete examples. In addition to the opportunities it will also try to identify cases of fools gold (pyrite), where the differences in method, tradition, or semantics between the two research fields may lead to a wild goose chase.

Vadim Zaytsev and Ralf Lämmel: A Unified Format for Language Documents

We have analyzed a substantial number of language documentation artifacts, including language standards, language specifications, language reference manuals, as well as internal documents of standardization bodies. We have reverse-engineered their intended internal structure, and compared the results. The Language Document Format (LDF), was developed to specifically support the documentation domain. We have also integrated LDF into an engineering discipline for language documents including tool support, for example, for rendering language documents, extracting grammars and samples, and migrating existing documents into LDF. The definition of LDF, tool support for LDF, and LDF applications are freely available through SourceForge.

Einar W. Høst and Bjarte M. Østvold: Canonical method names for Java

Programmers rely on the conventional meanings of method names when writing programs. However, these conventional meanings are implicit and vague, leading to various forms of ambiguity. This is problematic since it hurts the readability and maintainability of programs. Java programmers would benefit greatly from a more well-defined vocabulary. Identifying synonyms in the vocabulary of verbs used in method names is a step towards this goal. By rooting the meaning of verbs in the semantics of a large number of methods taken from real-world Java applications, we find that such synonyms can readily be identified. To support our claims, we demonstrate automatic identification of synonym candidates. This could be used as a starting point for a manual canonicalisation process, where redundant verbs are eliminated from the vocabulary.

Sebastián González, Nicolás Cardozo, Kim Mens, Alfredo Cádiz, Jean-Christophe Libbrecht and Julien Goffaux: Subjective-C: Bringing Context to Mobile Platform Programming

Thanks to steady advances in hardware, mobile computing platforms are nowadays much more connected to their physical and logical environment than ever before. To ease the construction of adaptable applications that are smarter with respect to their execution environment, the context-oriented programming paradigm has emerged. However, up until now there has been no proof that this emerging paradigm can be implemented and used effectively on mobile devices, probably the kind of platform which is most subject to dynamically changing contexts. In this paper we study how to effectively realise core context-oriented abstractions on top of Objective-C, a mainstream language for mobile device programming. The result is Subjective-C, a language which goes beyond currently existing context-oriented languages by providing a rich encoding of context interdependencies. Our initial validation cases and efficiency benchmarks make us confident that context-oriented programming can become mainstream in mobile application development.

Colin Atkinson, Bastian Kennel and Björn Goß: The Level-agnostic Modeling Language

As an alternative modeling infrastructure and paradigm, multi-level modeling addresses many of the conceptual weaknesses found in the four level modeling infrastructure that underpins traditional modeling approaches like UML and EMF. It does this by explicitly distinguishing between linguistic and ontological forms of classification and by allowing the influence of classifiers to extend over more than one level of inst*ant*iation. Multi-level modeling is consequently starting to receive attention from a growing number of research groups. However, there has never been a concrete definition of a language designed from the ground-up for the specific purpose of representing multi-level models. Some authors have informally defined the "look and feel" of such a language, but to date there has been no systematic or fully elaborated definition of its concrete syntax. In this paper we address this problem by introducing the key elements of a language, known as the Level-Agnostic Modeling Language (LML) designed to support multi-level modeling.

Raphael Mannadiar and Hans Vangheluwe: Debugging in Domain-Specific Modelling

An important obstacle to the wide-spread adoption of model-driven development approaches in industry is the lack of proper debugging facilities. Software debugging support is provided by a combination of language and Integrated Development Environment (IDE) features which enable the monitoring and altering of a running program's state. In Domain-Specific Modelling (DSM), debugging activities have a wider scope: *designers* debug model transformations (MTs) and synthesized artifacts, while domain-specific *modellers* debug their models, unaware of generated artifacts. This work surveys the state-of-the-art of debugging in the context of DSM and proposes a mapping between debugging concepts (e.g., *breakpoints*, *assertions*) in the software and DSM realms.

Markus Herrmannsdoerfer: COPE - A Workbench for the Coupled Evolution of Metamodels and Models

Model-driven software development promises to increase productivity by offering modeling languages tailored to a problem domain. Consequently, an increasing number of modeling languages are built using metamodel-based language workbenches. In response to changing requirements and technologies, the modeling languages and thus their metamodels need to be adapted. Manual migration of existing models in response to metamodel adaptation is tedious and error-prone. In this paper, we present our tool COPE to automate the coupled evolution of metamodels and models. To not lose the intention behind the adaptation, COPE records the coupled evolution in an explicit history model. Based on this history model, COPE provides advanced tool support to inspect, refactor and recover the coupled evolution.

Bruno Barroca, Levi Lucio, Vasco Amaral, Roberto Felix and Vasco Sousa: DSLTrans: A Turing Incomplete Transformation Language

In this paper we present DSLTrans: a visual language and a tool for model transformations[2]. We aim at tackling a couple of important challenges in model transformation languages --- transformation termination and confluence. The contribution of this paper is the proposition of a transformation language where all possible transformations are guaranteed to be terminating and confluent by construction. The resulting transformation language is simple, turing incomplete and includes transformation abstractions to support transformations in a software language engineering context. Our explanation of DSLTrans includes a complete formal description of our visual language and its properties.

[2] This work has been developed in the context of project BATIC3S partially funded by the Portuguese FCT/MCTES ref. PTDC/EIA/65798/2006, the doctoral grant ref. SFRH/BD/38123/2007, the post doctoral grant ref. SFRH/BPD/65394/2009 and the Luxembourguese FNR/CORE project MOVERE ref. C09/IS/02

Adrian Johnstone and Elizabeth Scott: Translator generation using ART

ART (Ambiguity Resolved Translators) is a new translator generator tool which provides fast generalised parsing based on an extended GLL algorithm and automatic generation of tree traversers for manipulating abstract syntax. The input grammars to ART comprise modular sets of context free grammar rules, enhanced with regular expressions and annotations that describe disambiguation and tree modification operations using the TIF (Tear-Insert-Fold) formalism. ART generates a GLL parser for the input grammar along with an *output grammar* whose derivation trees are the abstract trees specified by the TIF tree modification operations.

Jean-Marie Favre, Dragan Gasevic, Ralf Lämmel, and Ekaterina Pek: Empirical language analysis in software linguistics

Software linguistics is the science of software languages. In this short paper, we sketch the general discipline of software linguistics, but our focus is on one part of it: empirical analysis of software languages. Such analysis is concerned with understanding language usage on the grounds of a corpus. In this short paper, we sketch a survey on empirical language analysis, and we argue that the research method of content analysis is needed for a thorough survey.

Lennart C. L. Kats, Karl T. Kalleberg and Eelco Visser: Interactive Disambiguation of Meta Programs with Concrete Object Syntax

In meta-programming with concrete object syntax, meta programs can be written using the concrete syntax of manipulated programs. Quotations of concrete syntax fragments and anti-quotations for meta-level expressions and variables are used to manipulate the abstract representation of programs. These small, isolated fragments are often ambiguous and must be explicitly disambiguated with quotation tags or types, using names from the non-terminals of the object language syntax. Discoverability of these names has been an open issue, as they depend on the (grammar) implementation and are not part of the well-known concrete syntax of a language. Based on advances in interactive development environments, we introduce *interactive disambiguation* to address this issue, providing real-time feedback and proposing quick fixes in case of ambiguities.

Arnaud Hubaux, Quentin Boucher, Herman Hartmann, Raphaël Michel and Patrick Heymans: Evaluating a Textual Feature Modelling Language: Four Industrial Case Studies

Feature models are commonly used in software product line engineering as a means to document variability. Since their introduction, feature models have been extended and formalised in various ways. The majority of these extensions are variants of the original tree-based graphical notation. But over time, textual dialects have also been proposed. The *textual variability language* (TVL) was proposed to combine the advantages of both graphical and textual notations. However, its benefits and limitations have not been empirically evaluated up to now. In this paper, we evaluate TVL with four cases from companies of different sizes and application domains. The study shows that practitioners can benefit from TVL. The participants appreciated the notation, the advantages of a textual language and considered the learning curve to be gentle. The study also reveals some limitations of the current version of TVL.

Nils Bandener, Christian Soltenborn, and Gregor Engels: Extending DMM Behavior Specifications for Visual Execution and Debugging

Dynamic Meta Modeling (DMM) is a visual semantics specification technique targeted at behavioral languages equipped with a metamodel defining the language's abstract syntax. Given a model and a DMM specification, a transition system can be computed which represents the semantics of that model. It allows for the investigation of the model's behavior, e.g. for the sake of understanding the model's semantics or to verify that certain requirements are fulfilled. However, due to a number of reasons such as tooling and the size of the resulting transition systems, the manual inspection of the resulting transition system is cumbersome.

One solution would be a visualization of the model's behavior using animated concrete syntax. In this paper, we show how we have enhanced DMM such that visual execution and debugging can be added to a language in a simple manner.

Nicolas Genon, Patrick Heymans and Daniel Amyot: Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation

BPMN 2.0 is an OMG standard and one of the leading process modelling notations. Although the current language specification recognises the importance of defining a visual notation carefully, it does so by relying on common sense, intuition and emulation of common practices, rather than by adopting a rigorous scientific approach. This results in a number of suboptimal language design decisions that may impede effective model-mediated communication between stakeholders. We demonstrate and illustrate this by looking at BPMN 2.0 through the lens of the Physics of Notations, a collection of evidence-based principles that together form a theory of notation design. This work can be considered a first step towards making BPMN 2.0's visual notation more cognitively effective.

Sebastian Thore Erdweg and Klaus Ostermann: Featherweight TeX and Parser Correctness

TeX (and its LaTeX incarnation) is a widely used document preparation system for technical and scientific documents. At the same time, TeX is also an unusual programming language with a quite powerful macro system. Despite the wide range of TeX users (especially in the scientific community), and despite a widely perceived considerable level of ``pain'' in using TeX, there is almost no research on TeX. This paper is an attempt to change that.

To this end, we present Featherweight TeX, a formal model of TeX which we hope can play a similar role for TeX as Featherweight Java did for Java. The main technical problem which we study in terms of Featherweight TeX is the parsing problem. As for other dynamic languages performing syntactic analysis at runtime, the concept of ``static'' parsing and its correctness is unclear in TeX and shall be clarified in this paper. Moreover, it is the case that parsing TeX is impossible in general, but we present evidence that parsers for practical subsets exists.

We furthermore outline three immediate applications of our formalization of TeX and its parsing: a macro debugger, an analysis that detects syntactic inconsistencies, and a test framework for TeX parsers.