

Monterey Workshop 2004

October 4-6

CUE Workshop 2004

October 7

Software Engineering Tools: Compatibility and Integration

Sponsored by:

Army Research Office

National Science Foundation

EU funds under the CUE Initiative

Vienna, Austria

Program of

Monterey Workshop 2004

Software Engineering Tools: Compatability and Integration

Sponsored by:

Army Research Office
National Science Foundation

October 4-6, 2004

Hotel Sauerhof

Vienna, Austria

Workshop Chairs:

Thomas Henzinger, Ecole Polytechnique Federale de Lausanne

Zohar Manna, Stanford University

CUE Workshop

Sponsored by:

EU funds under the CUE Initiative

National Science Foundation

October 7, 2004

Hotel Sauerhof

Vienna, Austria

Workshop Chair:

Tom Maibaum, King's College, London

Motivation and Objectives

Software is the new physical infrastructure of the information age. It is fundamental to economic success, scientific and technical research and national security. Our current ability to construct the large and complex software systems demanded for continued economic and military success are inadequate.

A lack of tool integration has been recognized as a major impediment to the construction of complex computer systems, especially in the development of embedded systems, where the design of a single artifact may involve up to 50 different tools, including design, functional analysis, performance analysis, etc. tools. Tool interoperability is an important prerequisite for increasing software productivity and reliability.

The challenge is to create a platform that brings together the formal methods community and user community. Research assumptions should be constrained to facilitate the integration of such point solutions into a more complete and coupled overall engineering capability. This necessarily requires standardization at some levels, like we see in the hardware and middleware community, but can still allow a wide variety of representations and differences in expressive power at higher levels, thus catering to different application areas. The platform can support a variety of proof techniques, static and dynamic analyzers, type checkers, documentation tools, as well as lower level tools that support these techniques, such as decision procedures, constraint solvers, and SAT solvers.

The advantages of such a platform are many for both researchers and users: it provides researchers with exposure to practical case studies and the opportunity to have their methods and tools evaluated. It provides users with access to state-of-the-art methods without the added burden of extensive adaptation and training. newpage

Monterey Workshop Chairs

Thomas Henzinger, Ecole Polytechnique Federale de Lausanne

Zohar Manna, Stanford University

Program Committee

Bruce Krogh – Carnegie Mellon University

Matt Dwyer – Kansas State University

Tom Maibaum – King’s College, London, UK

Martin Wirsing – Ludwig Maximilians University Munich, Germany

Carlo Ghezzi – Polytechnic of Milan, Italy

Henny Sipma – Stanford University

Kane Kim – University of California at Irvine

Carlos Delgado Kloos – University Carlos III of Madrid, Spain

Lori Clarke – University of Massachusetts

Olaf Owe – University of Oslo, Norway

Insup Lee – University of Pennsylvania

Ugo Montanari – University of Pisa, Italy

Wolfgang Pree – University of Salzburg, Austria

Local Arrangements

Hermann Kopetz – Vienna University of Technology

Sibylle Kuster – Vienna University of Technology

Program of the Monterey Workshop

Sunday, October 3	
7:00–9:00 pm	Welcome Reception
Monday, October 4	
9:00	<i>Opening Remarks</i> David Hislop, Army Research Office
Session 1: User Perspectives on Tool Integration	
9:30	<i>Commercial Tool Integration: The Reactis Experience</i> Rance Cleaveland, SUNY at Stony Brook, Reactive Systems Inc.
10:10	Discussion
10:30	Coffee Break
10:50	<i>Integrated Tools for Integrated Methods</i> Alan Wassyng, McMaster University, Canada
11:10	<i>Post-mortem analysis of embedded systems: A case study</i> Purush Iyer, North Carolina State University
11:30	<i>TTP-Tools – The tool set of the Time-Triggered Architecture</i> Stefan Poledna, TTTech, Vienna
11:50	Discussion
12:15	Lunch
Session 2: Model Driven Architectures	
1:30	<i>Model-Integrated Computing</i> Janos Sztipanovits, Vanderbilt University
2:10	<i>Architecture-based Methods and Tools for Software System Design</i> Tom Maibaum, King's College, London
2:30	Discussion
3:00	Coffee break
3:20	<i>Architecture-based Verification</i> Lori Clarke, University of Massachusetts
3:40	<i>Evolution of Architecture Models</i> Bernhard Rumpe, Technical University of Braunschweig
4:00	<i>Model driven development of multimedia applications</i> Heinrich Hussman, Andreas Pleuss Ludwig-Maximilians-Universitaet, Munich
4:20	Discussion
5:00	Adjourn
Evening program: All participants are invited by TTTech for wine tasting at the vinery Urbanusschenke Familie Eitler Habsburgerstr. 62 a, 2500 Baden, Tel. 02252/ 20 9521	

Tuesday, October 5	
Session 3: Component-based Systems and Interface Theories Chair: Gabor Karsai	
8:30	<i>Interface-based Design</i> Tom Henzinger, Ecole Polytechnique Federale de Lausanne
9:10	<i>Threading by Appointment</i> Christoph Kirsch, University of Salzburg
9:30	<i>The Creol approach to open distributed systems</i> Olaf Owe, University of Oslo
9:50	Discussion
10:20	Coffee break
10:40	<i>Issues in Integration of TMO Programming and Specification Tools with Basic Object-Oriented Distributed Software Engineering Tools</i> Kane Kim, University of California, Irvine
11:00	<i>Compositional Periodic Interface for Real-Time Components</i> Insup Lee, University of Pennsylvania
11:20	<i>Priority Systems</i> Joseph Sifakis, Verimag, Grenoble
11:40	Discussion
12:15	Lunch
Session 4: Software Analysis and Design Tools	
1:30	<i>Multiple Interoperating Program Analyses</i> Martin Rinard, MIT
2:10	<i>Timing Prediction and Timing Predictability</i> Reinhard Wilhelm, University of Saarland, Saarbrucken
2:30	Discussion
3:00	Coffee break
3:20	<i>Model Checking, Abstraction and Symbolic Execution for Software</i> Sriram Rajamani, Microsoft Research
4:00	<i>Linking theories of concurrency</i> He Jifeng, UNU/IIST, Macau
4:20	Discussion
5:00	Adjourn
Evening Program	
7 pm	Workshop Dinner

Wednesday, October 6	
<p style="text-align: center;">Session 5: Open Tool Integration Frameworks Chair: Purush Iyer</p>	
8:30	<i>Black-box versus White-box Integration: Alternative Integration Strategies</i> Shmuel Katz, Technion
9:10	<i>The Interface is the Message</i> Shankar, SRI
9:50	Discussion
10:20	Coffee break
10:40	<i>Verification on the Web of Mobile Systems</i> Ugo Montanari, University of Pisa
11:00	<i>XML Language Composition and Integration of XML-Processing Tools</i> Tomasz Janowski, UNU/IIST, Macau
11:20	<i>(Testing) Tool Integration via Theory Integration</i> Bernhard Aichernig, UNU/IIST, Macau
11:40	Discussion
12:15	Lunch
<p style="text-align: center;">Session 6: Standardized Modeling Environments Chair: Joseph Sifakis</p>	
1:30	<i>Tool Integration Aspects in the Model-Driven Architecture</i> Gabor Karsai, Vanderbilt University
2:10	<i>Simulink integration patterns and lessons learned from integrating Giotto/TDL and Simulink</i> Wolfgang Pree, University of Salzburg
2:30	Discussion
3:00	Coffee break
3:20	<i>Integration of formal methods in MDA using the model bus</i> Xavier Blanc, University of Pierre & Marie Curie, Paris
3:40	<i>Basing a Modeling Environment on a General Purpose Theorem Prover</i> Myla Archer, Naval Research Lab
4:00	<i>Toward a more Formal Basis for Middleware</i> Chris Gill, Washington University at St Louis
4:20	Discussion
4:50	<i>Closing Remarks</i>

Program of the CUE Workshop

Thursday, October 7	
9:00	<i>Integrating Tools for Practical Software Analysis</i> Zohar Manna and Henny Sipma
9:45	Discussion
10:30	Coffee break
11:00	Panel discussion: Architectures for Tool Integration Chair: Gianluigi Ferrari Panel members: Joseph Sifakis, Bernhard Steffen, Rance Cleaveland, Lin Huimin
12:30	Lunch
14:00	Panel discussion: On the Need for Improved Methods, Models, and Languages in Tool-Based Software Development Chair: Connie Heitmeyer Panel members: Manfred Broy, Insup Lee, Joseph Sifakis, Alan Wassyng
15:30	Coffee break
16:00	Summary Discussion Chair: Manfred Broy
17:00	Closing

List of Attendees

Bernhard Aichering	United Nations University – IIST, Macau
Myla Archer	Naval Research Lab, Washington DC
Xavier Blanc	University of Pierre & Marie Curie, Paris
Manfred Broy	Technical University Munich
Lori Clarke	University of Massachusetts, Boston, MA
Rance Cleaveland	State University of New York at Stony Brook and Reactive Systems, Inc
GianLuigi Ferrari	University of Pisa
Chris George	United Nations University – IIST, Macau
Chris Gill	Washington University at St Louis
Helen Gill	National Science Foundation, Washington DC
Klaus Havelund	Kestrel Technology / NASA Ames Research Center
He Jifeng	United Nations University – IIST, Macau
Connie Heitmeyer	Naval Research Lab, Washington DC
Tom Henzinger	Ecole Polytechnique Federale de Lausanne
David Hislop	Army Research Office
Heinrich Hussmann	Andreas Pleuss Ludwig-Maximilians-Universitaet, Munich
Purush Iyer	North Carolina State University, Raleigh, NC
Tomasz Janowski	United Nations University – IIST, Macau
Gabor Karsai	Vanderbilt University, Nashville, TN
Shmuel Katz	Technion, Haifa
Kane Kim	University of California at Irvine, CA
Christoph Kirsch	University of Salzburg
Hermann Kopetz	Vienna University of Technology
Fabrice Kordon	University of Pierre & Marie Curie, Paris
Insup Lee	University of Pennsylvania
Lin Huimin	Institute of Software, Beijing
Tom Maibaum	King’s College, London
Zohar Manna	Stanford University, CA
Bertrand Meyer	Swiss Federal Institute of Techology, Zurich
Ugo Montanari	University of Pisa
Olaf Owe	University of Oslo

Klaus Pohl	University Duisburg, Essen
Stefan Poledna	TTTech, Vienna
Wolfgang Pree	University of Salzburg, Austria
Sriram Rajamani	Microsoft Research, Redmond, WA
Martin Rinard	MIT, Boston, MA
Bernhard Rumpe	Technical University of Braunschweig
Shankar	SRI, Menlo Park, CA
Josef Sifakis	Verimag, Grenoble
Henny Sipma	Stanford University, CA
Neeraj Suri	Technical University of Darmstadt
Janos Sztipanovits	Vanderbilt University, Nashville, TN
Alan Wasslyng	McMaster University, Canada
Reinhard Wilhelm	University of Saarland, Saarbrücken

ABSTRACTS

Commercial Tool Integration: The Reactis Experience

Rance Cleaveland

Department of Computer Science SUNY at Stony Brook
CEO Reactive Systems Inc. Falls Church, VA, USA

Reactis is a model-based testing and validation tool for control software developed by Reactive Systems, Inc., a company started. Using Reactis, control-system developers can generate test data from, and check properties of, models of controllers given in the Simulink / Stateflow notations of The MathWorks, Inc., the market leader in control-system design tools. A number of automotive, aerospace and consumer-electronics companies use Reactis in their software-development processes.

In building Reactis, we had to confront a number of integration issues with The MathWorks' Simulink / Stateflow tool set. In this talk I will outline our approaches to these.

Integrated Tools for Integrated Methods

Alan Wassying

McMaster University

There is a natural symbiotic relationship between software development methods and software tools. It is our task to make sure that the relationship is mutually beneficial. Tool integration should start with method integration. It is time we concentrated on building software development methods that are integrated over all aspects of the development life-cycle, at least within specific application domains. To support such methods we need tools that are just as integrated. This paper presents ideas for such integrated methods and tools, with examples drawn from an industrial safety-critical project.

Post-mortem analysis of embedded systems: A case study

Raoul Jetley and Purush Iyer

North Carolina State University

We report on our experience in using tools that are available for analyzing C programs, slicing tools and software-model checkers, to analyze a large (approx 30KLOC), but not complete,- system which is part of a medical device. Our goal is post-mortem analysis – going from bug reports to cause of those bugs. Clearly, there are no tools that are currently available that could carry out the task automatically. Our report, therefore, details the steps that were carried out manually, with the aid of available tools, to identify the cause of bugs reported to the manufacturer of our case study. We conclude with an analysis of what future tools should address to make automation of post-mortem analysis possible.

TTP-Tools – The tool set of the Time-Triggered Architecture

Stefan Poledna

TTTech

The Time-Triggered Architecture provides a framework for the efficient development of highly dependable embedded applications. As all relevant system activities are triggered by the progression of time, a dedicated tool approach is necessary that integrates well with existing methodologies and tools.

TTP-Tools is a specific toolset addressing these requirements from high-level systems integration, code generation and configuration for middleware components and a time-triggered operating systems, as well as download and visualization capabilities.

For highly dependable and safety relevant systems the TTP-Tools in conjunction with its run-time components provide a platform that allows the transparent and fault-tolerant execution of application code. Based on tool settings the replication degree and voting scheme can be selected per application.

This presentation will elaborate the TTP-Tools architecture, its scheduling capabilities and integration concepts.

Model-Integrated Computing

Janos Sztipanovits

Vanderbilt University

Computing and communication is increasingly used as universal system integrator in physical systems. Embedded computers are surrounded by physical processes: they receive their inputs from sensors and send their outputs to actuators. Embedded computing devices, viewed from their sensor and actuator interfaces, act like physical processes with dynamics, noise, fault, size, power and other physical characteristics. The role of the embedded software is to "configure" the computing device so as to meet physical requirements. It is not surprising that using current software technology, logical/functional composability does not imply physical composability. In fact, physical properties are not composable, rather, they appear as cross-cutting constraints in the development process. The effects of cross-cutting constraints can be devastating for the design. Meeting specifications in one part of the system may destroy performance in others, and, additionally, many of the problems will surface at system integration time. Consequently, we need to go beyond conventional software technology to a model-based system/software design technology, which addresses the design of the whole system with its many interdependent physical, functional and logical aspects.

This talk describes Model-Integrated Computing (MIC), which comprises modeling, model analysis and model transformation technologies as foundation for embedded software composition. The primary goals are to provide an overview of the core elements of the MIC software infrastructure: meta-languages, meta-programmable tools and tool architectures, and to discuss research directions on building semantic foundation for MIC.

Architecture Based Methods and Tools for Software Systems Design

Tom Maibaum

King's College

The field of Software Architecture has made a lot of progress over the past decade in raising the level of abstraction at which software systems are designed. Existing Architecture Description Languages have various shortcomings, resulting from their 'premature' birth, being used to define the subject rather than having their requirements defined by the perceived needs of the subject. We are embarking on building languages, methods and tools to support architecture based design (and implementation) of systems. The ADL supporting abstract design, tentatively called Wren, supports a notion of component that completely externalises the definition of interaction and incorporates directly concepts of hierarchical design. Refinement and implementation notions, including code generation, will be based completely on transformational concepts, as will the generation of various other 'representations, such as documentation, traceability, etc.

Architecture-based Verification

Shangzhu Wang, George S. Avrunin, and Lori A. Clarke

University of Massachusetts

Architectural-based verification is particularly important for distributed systems since it is often difficult to determine the appropriate interactions among the components. An approach is needed in which it is easy to experiment with different interaction models among the components in order to get early feedback about the impact on the overall behavior of the resulting systems. One of the problems that currently arise with this approach, however, is that the semantics of the components are inextricably intertwined with the semantics of the interaction model. For example, a simple change from an asynchronous send to a synchronous send requires that the component now be able to block and process an acknowledgement. Of course, more complex interactions models that support features, such as priorities and bounds, require more extensive component modifications. We have been developing a building block approach to architectural design that separates a component's behavior from its interaction models as well as from the connectors. This leads to an architectural design where it is indeed possible to experiment with alternative interaction models without having to modify the components. We combine this approach with finite state verification so that developers can make architectural modifications and quickly receive feedback about the impact of these changes on important system properties.

Although our long-term goal is to find a common set of building blocks for the most widely used interaction models, we present here some preliminary results based on the investigation of one particular interaction model - Message Passing. Variants of message passing semantics are investigated and formally defined in terms of a set of reusable ports. An example is presented to show how the interactions of a message passing system can be specified using ports and how finite-state verification techniques can be used to find a sound design.

Evolution of Architecture Models

Bernhard Rumpe

Technical University of Braunschweig

Today an increasing part of newly developed systems replace legacy systems. Many of those are being replaced, because even small changes of functionality have become unmanageable due to old technology, complex architecture and missing documentation. Easy change of systems, therefore, becomes increasingly important due to faster changes of requirements as well as technologies both in the business and in the embedded systems domain. Models and their extensive use during development have become an important technique to improve reusability and adaptability of developed parts of a system.

In this talk we introduce the foundational concept of model engineering and demonstrate an approach to use model engineering for evolution of software architectures. We show how such software evolution can be managed on an architectural level provided that reliability ensuring mechanisms, such as test models exist. The key idea of the approach is to use model artifacts to describe the architecture of a system and others to model tests. Both can be analyzed for consistency and can be animated in order to derive test results.

An architecture is evolved using systematic refactoring techniques and thus becomes a lot easier when regression tests allow to repeatedly check the correctness of each evolution step and thus retain reliability over the evolutionary process.

Model driven development of multimedia applications

Heinrich Hussmann

Andreas Pleuss Ludwig-Maximilians-Universitaet, Munich

The development of multimedia applications is an area which is still missing much of the achievements of software engineering. The used development style is often dominated by the individual platform (e.g. authoring tool), and an abstract architecture of the application is in most cases not existing. On the other hand, multimedia applications and applications with multimedia elements are taking over an increasing part of the software market. Therefore, there is a need for development methods focusing on the specific characteristics of multimedia applications, which are integration of media objects, high relevance of the user interface, and a high degree of interaction. In this paper we propose to employ the Model-Driven Architecture (MDA) paradigm for multimedia applications. Based on existing modeling languages for multimedia, we propose an object-oriented visual modelling of multimedia application architectures focusing on a practical usability for developers. These models are platform-independent in the MDA terminology of the OMG and enable code generation for multiple platforms. Thereby is not mandatory for the developer to provide complete models or to include application details to get valuable code. Application details like concrete behavior or user interface design can be alternatively included in the model, inserted directly into the generated code or produced by other specialized tools. We also briefly report on a prototypical implementation of the proposed approach generating code for the target platforms Flash/ActionScript or SVG/JavaScript.

Interface-based Design

Tom Henzinger

Ecole Polytechnique Federale de Lausanne and University of California at Berkeley

Interfaces play a central role in the component-based design of software systems. Good interface design is based on two principles. First, an interface should expose enough information about a component so to make it possible to predict if two or more components work together properly, by looking only at their interfaces. Second, an interface should not expose more information about a component than is required by the first principle. The technical realization of these two principles depends, of course, on our interpretation of what it means for two or more components to "work together properly." A simple interpretation is offered by typed programming languages: a component that implements a function and a component that calls that function are compatible if the function definition and the function call agree on the number, order, and types of the parameters. We present richer notions of interfaces, which expose in addition to type information, also temporal information about components. For example, the interface of a file server with the two methods `open-file` and `read-file` may stipulate that `read-file` must not be called before `open-file` has been called. Symmetrically, the interface of a client specifies the possible sequences of `open-file` and `read-file` calls during its execution, so that a compiler can check if the file server and the client fit together. Such behavioral interfaces, which expose temporal information about a component and at the same time impose temporal requirements on the environment of the component, can be specified naturally using an automaton-based language. The notion of compatibility can be refined further, by exposing real-time information and resource-use information, leading to push-down, timed, and resource interfaces. For instance, resource interfaces can be used to ensure that no two components simultaneously access a unique resource.

We formally capture the requirements on interfaces by axiomatizing interface theories. For example, the axiom of "independent implementability" of interfaces stipulates that if A and B are compatible interfaces, and A' is a component that conforms to interface A , and B' is a component that conforms to interface B , then the composition $A' \text{---} B'$ of the two components conforms to the composite interface $A \text{---} B$. For selected interface formalisms, such as behavioral, timed, and resource interfaces, we show that they satisfy the axioms of interface theories, and we discuss the following algorithmic problems:

1. Compatibility: given two interfaces, are they compatible?
2. Conformance: given an interface and a component, does the component conform to the interface?
3. Synthesis: given an interface theory and a component, what is its interface?

In particular, we show that the compatibility checking of interfaces amounts to solving a game in which the interfaces and the unknown environment represent players. Furthermore, we show that the conformance relationship between a component and its interface must be a contravariant one, which treats inputs and outputs differently. This distinguishes interface conformance from most notions of refinement used in formal verification.

The work reported here is joint with Arindam Chakrabarti, Luca de Alfaro, and Marielle Stoelinga.

Threading by appointment

Christoph Kirsch

University of Salzburg

We propose a concurrent programming model called threading-by-appointment (TAP). Unlike traditional threads, TAP threads can only communicate with system components or other threads by appointment. For example, a TAP thread cannot simply try to access some shared resource. Instead, a TAP thread must make an appointment with the shared resource in advance. Only at the time of the appointment, the thread can actually access the shared resource. The desired start time and duration of an appointment may either be provided by the thread or computed by the TAP runtime system. The duration of an appointment must be finite. Instantaneous appointments are possible but only allow a single atomic operation per appointment. TAP threads cannot deadlock because the duration of appointments is finite. Race conditions may occur for making appointments but can be avoided by making appointments independently of the system's progress. Threading-by-appointment makes the process of determining the time instants for system interaction explicit and may therefore help to increase the determinism of concurrent applications as well as the efficient use of resources. Interesting questions arise such as how to schedule appointments as opposed to how to multiplex resources such as the CPU, and how to define structured programming elements that support threading-by-appointment.

The Creol approach to open distributed systems

Olaf Owe

University of Oslo

The talk considers a formal object-oriented model for distributed computing. Object orientation appears as a leading framework for concurrent and distributed systems. However, the synchronization of the RPC communication model is unsatisfactory in many distributed systems. Asynchronous message passing gives better control and efficiency in this setting, but lacks the structure and discipline of traditional object-oriented methods. The integration of the message concept in the object-oriented paradigm is unsettled, especially with respect to inheritance and redefinition.

We propose an approach combining asynchronous method calls and conditional processor release points, which reduces the cost of waiting for replies in the distributed environment while avoiding low-level synchronization constructs such as explicit signaling. Even the lack of replies to method calls in unstable environments need not lead to deadlock in the invoking objects. This property seems attractive in asynchronous, open, or unreliable environments. Furthermore, the approach allows active and passive behavior (client and server) to be combined in concurrent objects in a natural way.

We consider the integration of these constructs with a mechanism for multiple inheritance within a small object-oriented language, Creol. The language constructs are formally described by an operational semantics defined in rewriting logic, which is executable as a language interpreter in the Maude tool. The use of the language constructs is illustrated by an example of a peer-to-peer network.

Issues in Integration of TMO Programming and Specification Tools with Basic Object-Oriented Distributed Software Engineering Tools

Kane Kim

University of California at Irvine

The TMO (Time-triggered Message-triggered Object) programming and specification scheme is intended to facilitate RT distributed programming and software engineering in a form which software engineers in the vast business software field can adapt to with relatively small efforts. It is also aimed for enabling system engineers to confidently produce certifiable RT computing systems for safety-critical applications, i.e., the systems possessing the reliability which will be demanded by the general public in the 21st century. Its support tools can be based on well-established OO programming languages such as C++ and JAVA and on ubiquitous commercial RT operating system kernels or even on MS Windows. In addition, the TMO scheme facilitates an attractively simple approach to parallel and distributed RT simulation. However, exploration of all these potentials is still at an early stage.

In this presentation, a brief review of the underlying design paradigm and the essence of the scheme, the tools (including middleware supporting fault-tolerant execution of TMOs) and application demonstrations built so far, will be given first. Thereafter, some of major issues in integrating the TMO programming and specification tools with other well established basic OO distributed software engineering tools will be discussed.

Compositional Periodic Interface for Real-Time Components

Insup Lee and Insik Shin

University of Pennsylvania

As embedded systems become more complex, it becomes necessary to design complex embedded systems through component-based design. For real-time embedded systems, a challenging problem is to develop simple yet effective interfaces that support compositional real-time guarantees in a hierarchy of components. Recent studies addressed this problem using a periodic resource interface (PRI) that abstracts the collective temporal resource requirement of a component with a single periodic task model. The principal techniques to develop the PRI include to calculate the minimum resource supply of a periodic resource and to extend traditional scheduling theories with such calculations. We describe the framework and identify interesting research and development challenges.

Priority Systems

Joseph Sifakis

Verimag, Grenoble

We present a framework for the incremental construction of deadlock-free systems meeting given safety properties. The framework borrows concepts and basic results from the controller synthesis paradigm by considering a step in the construction process as a controller synthesis problem.

We show that priorities are expressive enough to represent restrictions induced by deadlock-free controllers enforcing safety properties. We define a correspondence between such restrictions and priorities and provide compositionality results about the preservation of this correspondence by operations on safety properties and priorities. Finally, we provide examples illustrating application of the results to the incremental construction of deadlock-free systems.

Multiple Interoperating Program Analyses

Martin Rinard

Massachusetts Institute of Technology

Over the years researchers have developed many program analyses that can extract or verify a range of interesting program properties. Two problems, however, complicate the deployment of these analyses: diversity (the fact that different analyses are suitable for different properties) and scalability (the need to analyze programs of substantial size). There is a need for an analysis framework that will allow multiple analyses to cooperate to analyze a single sizable program, with each analysis operating on only that part of the program for which it is best suited. Such a framework holds out the promise of solving both the diversity problem (by allowing different analyses to cooperate to analyze the same program) and the scalability problem (by allowing each analysis to operate only on a manageable part of the program).

In this talk we discuss some general issues that arise in such a framework and present some of our experiences developing both programs and analyses in the context of an initial general program analysis framework that supports multiple analyses.

Timing Prediction and Timing Predictability

Reinhard Wilhelm

University of Saarland, Saarbrücken

Hard real-time systems need methods and tools to determine safe and precise bounds on execution times. Modern computer architectures and current system-design methods endanger the predictability of the systems timing behavior. The talk will report about the state of the art in determining bounds on execution times, on threats to the predictability, and propose a discipline Design for Timing Predictability.

Model Checking, Abstraction and Symbolic Execution for Software

Sriram Rajamani

Microsoft Research

Over the past few years, there has been growing interest in using techniques such as predicate abstraction, symbolic execution, model checking, and abstraction-refinement to prove properties about programs written in common programming languages like C. I will give an overview of the issues involved in applying these techniques to software, and also how compiler techniques such as data-flow analysis and pointer analysis interact with these techniques in building a usable verification system. I will also discuss current limitations of these approaches, in terms of scale and usability, and speculate on possible solutions.

Linking theories of concurrency

He Jifeng

United Nations University – IIST

In the study of process calculi, there are two standard approaches to the definitions of similarity or equivalence of processes. The first is bisimulation, an equivalence relation based on the structural operational semantics of the calculus. The second is refinement, an ordering defined as inclusion of the sets of observations that may be made of the behaviour of each process.

The advantages of bisimilarity are simplicity, efficiency and variety. Bisimulation is based directly on an operational semantics, which provides an abstract model of implementation; it thereby appeals directly to the operational intuition of programmers, which is especially useful when diagnosing errors in a program. Bisimulation admits simple proofs using co-induction. And for particular programs, proofs can often be replaced by mechanical checking, because bisimulation is a direct description of an algorithm that can be used by a model checker.

The advantages of refinement are abstraction, expression power, coarseness and completeness. The observational semantics exploits the familiar abstraction of set theory. This permits a process to be specified by an arbitrary mathematical description of the observations of its behaviour: the specification does not have to be encoded in the syntax of the calculus. New operators can often be defined in the model, without invalidating theorems proved by normal mathematical properties of sets. Refinement permits proofs of more equations than bisimilarity; and each additional algebraic law may be useful for program optimisation and for reasoning about program correctness.

Of course, when bisimulation and refinement are reconciled, all their distinctive advantages can be combined and exploited whenever the occasion demands. This talk shows how to link these two approaches.

Black-box versus White-box Integration: Alternative Integration Strategies

Shmuel Katz

Technion, Haifa

Top-down black-box and bottom-up white-box approaches to the integration of multiple software tools are presented and compared. In the black-box approach, component tools are not influenced in any way by the integration, and only the interfaces to the tools are used to achieve integration. The VeriTech project to translate among formal methods verification tools is shown to exemplify this approach. In VeriTech, the language interfaces of distinct model-checking or theorem-proving tools are connected by translations that make use of an internal intermediate language. Additional information is gathered to show how differences among the languages are treated for a particular instance of translation. In the bottom-up white box approach to integration, modules of existing (or proposed) tools are generalized and made into robust reusable components. These become the basis of the integration framework, and are used to reimplement (and often improve) existing tools, and to develop new ones. This approach is demonstrated through the design of the Common Proof Environment (CPE) project. In CPE a generic collection of static analysis and verification modules are developed, and used to construct specific verification tools for a variety of programming and design languages. The generic collection includes dataflow and parsing components, as well as type checkers and other modules used by many of the full-fledged analysis tools. Syntactic and semantic issues specific to a tool are treated by writing adaptors that connect to the generic components.

In both approaches, basic questions of differing semantic assumptions, as well as distinct interfaces and built-in functionalities must be treated. The advantages and disadvantages of each strategy will be analyzed and compared.

The Interface is the Message

Shankar

SRI

Decision procedures are usually constructed as black-box routines that return Yes or No on a given conjecture. Based on our experience with systems such as PVS, SAL, and ICS, we argue that the interface to a decision procedure must allow a rich range of interactions including those for the assertion and retraction of information, querying, and the creation and deletion of contexts. We outline both the mathematical and engineering challenges in building inference modules that can be composed to yield effective deductive tools.

Verification On The Web Of Mobile Systems

Ugo Montanari

University of Pisa

The vast majority of current available verification environments have been built by sticking to traditional architectural styles: centralized and without dealing with interoperability and dynamic reconfigurability. In this talk we present a verification toolkit whose design and implementation exploits the Web service architectural paradigm (joint work with Gianluigi Ferrari, Stefania Gnesi, Roberto Raggi, Gianluca Trentanni, and Emilio Tuosto).

XML Language Composition and Integration of XML-Processing Tools

Tomasz Janowski

United Nations University – IIST

XML namespaces provide a simple syntactic mechanism to compose new XML languages from the existing XML languages, such that every element (of a document in this new language) belongs to at most one namespace. Thanks to this property, tools designed to process a composite XML language can rely on the tools for processing the component XML languages, supporting software reuse and language substitution.

During this talk, we explain the mechanism of XML language and tool composition using some concrete examples, and present a simple mathematical model to capture its meaning. Then we propose a new XML language to provide a high-level, language-neutral description of the composition of XML-processing tools using namespaces, define its semantics in terms of the formal model, and discuss the on-going implementation.

(Testing) Tool Integration via Theory Integration

Bernhard K. Aichernig

United Nations University – IIST

A main challenge in tool integration is the integration of theories. Different tools work on different languages and/or on different semantic domains which limits their synergies. In addition, opportunities for integration might be overlooked since results in one area of research are not realized by others. Test case generation is no exception.

In this talk we are going to present a fault-based testing theory that translates to different semantic domains. It is based on the general concept of an implementation relation and therefore can be applied to several frameworks and tools where such an implementation relation is known. As a consequence, we get a test case generation method that can be instantiated for a variety of different languages, including OCL, Z, B, RSL, CSP and Lotos. Dependent on the semantic framework, constraint solvers, model checkers, and theorem provers will be applied for implementing such a fault-based test case generator. Two such test case generators are currently under development. One for OCL using constraint solving, another one for labelled transition systems applying the CADP tools and the TGV test case generator.

Test case generation is also interesting from a different perspective. Test cases might be used to represent properties of an artifact. Thus, they can serve to hide theory from unskilled tool users.

Tool Integration Aspects in the Model-Driven Architecture

Gabor Karsai

Vanderbilt University

Proponents of the MDA vision seem to agree that it will become reality only if we have the proper tools to practice it. Using models in software development poses interesting challenges for the tool developers: tools are needed (1) for modeling on varying levels of abstraction, (2) for transforming models between modeling paradigms (and code), and (3) for analyzing and verifying properties of models (to ensure we build the right system correctly). In addition to the need for usable tools (that avoid becoming "shelfware" - a' la CASE), tools must talk to each other and work together as a seamlessly integrated ensemble. This talk will outline the various aspects of the model-driven development process, the specific tool categories needed, and the integration problems arising in tool suites.

Simulink integration patterns and lessons learned from integrating Giotto/TDL and Simulink

Wolfgang Pree

University of Salzburg

Mathworks' Matlab and Simulink tools have become a defacto modeling standard for control applications. Giotto with its sound formal semantics provides language constructs that allow the development of deterministic control applications. Their timing and communication behavior can be specified independent of their implementation. The Timing Description Language (TDL) enhances Giotto towards a component architecture for real-time control applications.

We present two integration patterns that result from an effort to seamlessly integrate TDL and Simulink. The integration of TDL and Simulink has several benefits. The developer uses Simulink to model the control laws and TDL to model the timing and communication aspects of a control system. The overall, deterministic system can be simulated in Simulink.

We try to generalize from our integration experience and describe the following two patterns.

1. Editor integration: Which extension mechanisms does Simulink offer? What are their advantages and their limits?
2. Extensions and restrictions of tool semantics: On the one hand Simulink needs to be restricted regarding the specification of timing behavior and of state transitions so that it adheres to Giotto/TDL semantics. On the other hand, Simulink offers only functions (called subsystems) as means for structuring models. TDL also provides the module construct for component-based development. Thus, the pattern describes the available options for modifying Simulinks semantics.

Basing a Modeling Environment on a General Purpose Theorem Prover

Myla Archer

Naval Research Lab

A general purpose theorem prover can be thought of as an extremely flexible modeling environment in which one can define and analyze almost any kind of model. A disadvantage to the full flexibility of a general purpose theorem prover is the lack of any guidance on how to construct a model and how then to apply the theorem prover to analyzing the model. In the general environment supplied by the prover, much time can be consumed in deciding how to specify a model and in interacting with and understanding feedback from the prover. However, specification templates, together with proof strategies whose design follows certain principles, can be used in many general purpose provers to create specialized modeling environments that address these difficulties. A specialized modeling environment created in this way can be further extended and/or further specialized in a straightforward way by drawing on the underlying theorem prover for additional capabilities. Moreover, a specialized modeling environment derived from an general purpose theorem prover provides a means of integrating powerful theorem proving capabilities into existing software development environments by way of appropriate translation schemes. This talk will use TAME (Timed Automata Modeling Environment) to illustrate the creation, extension, and specialization of a modeling environment based on PVS, and its integration into the SCR and TIOA development environments.

Toward a more Formal Basis for Middleware

Chris Gill

Washington University at St Louis

Model integrated computing and model-driven architectures are driving a resurgence of interest in system software that is designed and implemented on a formal basis. However, a significant tension exists between (1) supporting common off-the-shelf system software abstractions that developers of modern systems have come to expect, and (2) ensuring high fidelity between the models and the system software that is used. To resolve this tension, a new generation of system software is being designed and developed based on timed automata or other well formed abstractions from its inception. This approach offers significant advantages in fidelity of the models, while providing system developers the abstractions they have come to expect.

Broad progress has been made on the language and virtual machine fronts in projects such as Time-Triggered Architecture implementations, Giotto and the E-Machine. There have also been a select handful of middleware-focused efforts such as Time-Triggered Message-Triggered Objects, the Event Corellation Language and its implementation, and the Cadena and Bogor projects. However, broad-based formal approaches to lower-level distributed real-time and embedded systems middleware such as object request brokers (ORBs) are also needed.

This presentation outlines such an approach to ORB design and implementation, which leverages existing behavioral and structural descriptions of pattern-oriented middleware, but refines those descriptions with formal models of structure and behavior. This presentation also describes how existing middleware implementations can then be evaluated and possibly refactored to provide higher fidelity between ORB models and implementations.