Lehrstuhl für Softwaretechnik und Softwareproduktionsumgebung, Institut für Informatik, Albert-Ludwigs-Universität Freiburg

David Basin, Luca Viganò, Burkhart Wolff

Overview

The software engineering group was founded in 1997 within the newly created Faculty for Applied Sciences at the University of Freiburg. The faculty currently consist of 12 professors in Computer Science and an additional 10 in Microsystem Technology; when the hiring process is completed, there will be a total of 33 professors, split between these areas.

The software engineering group is currently staffed by 10 researchers, under the direction of Prof. Dr. David Basin. The main competence area is methods and tools for the specification, verification, and testing of computer software and hardware. Particular emphasis is on formal methods and their applications. Below we describe our teaching activities and research in this area.

Teaching

The software engineering group is responsible for the core curriculum course in software engineering (offered every summer semester), as well as courses and seminars on advanced topics in software engineering and related fields.

Our core software engineering course focuses on methods for developing large, complex systems. This includes topics such as models of the software development process, requirements analysis, model building, formal specification, design, and testing. Also covered are support tools (e.g., version management and different kinds of CASE tools) as well as language paradigms that support structured development in the large (covering subjects ranging from modularization, object orientation, component based system design to formal system architectures). The course is projected-centered: the students work on a software project together in groups and work through all the phases of development.

In addition, the software engineering group offers a number of advanced courses in software engineering and related topics. Recently offered courses include:

Model-based system specification: specification of system requirements using model-based specification languages (like Z and B), and methods and tool support for refining formal specifications into software.

Process oriented modeling: logical formalisms (e.g. state-charts and process calculi) and tool support (e.g., Statemate and model checkers like SPIN and FDR) for modeling and validating distributed systems and protocols, as well as their requirements.

Program verification: machine supported formal reasoning about programs using higher-order logic and specification languages embedded in higher-order logic (e.g., Z and CSP).

Computer Security: theory and practice of computer security. A wide range of topics are covered including cryptography, public-key infrastructures, access control, security models, security protocols, and security in open networks.

Testing: approaches to testing hardware and software systems, e.g., the systematic generation of tests from specifications or code.

Several of our courses have been offered as part of the VIROR project — the Virtual University Oberrhein — which involves distributed teaching/learning where multi-medial seminars and courses are multi-broadcast between different universities.

Research

The software engineering group's main area of expertise is on formal methods, which are methods based on mathematical modeling and formal logic, and their applications. One of the myths of formal methods is that they are too difficult and resource intensive for practical application. Our research addresses different ways that formal methods can be used with different associated costs. These methods range from "light-weight" approaches to static program analysis to full-scale formal refinement or verification. Depending on the application and the degree to which it is safety, or mission-critical, different methods can be applied with different resource requirements. Below we survey some of our current projects in this area.

Projects

Formal Methods for Secure Mobile Code Over the past three years we have carried out two projects with *Deutsche Telekom* on applying formal methods to establishing security properties of mobile code. The problem we investigated concerns establishing that mobile code accesses host memory in only limited, safe, ways (e.g., memory is accessed only within a pre-defined address space). We have implemented a solution based on abstraction and model checking: a Java byte code program is abstracted into a finite state transition system for which we automatically generate temporal logic formulae specifying constraints on memory access. By model checking we can automatically test whether the constraints are satisfied. Our solution offers a number of advantages over alternative approaches to byte code verification, including extensibility and ease of integration with other formal methods.

Security Architectures We have recently begun a DFG project on formal methods for modeling and validating security requirements of open distributed systems. Our goal is to develop methods and tool support to establish security requirements during different system design stages. In particular, *high-level designs* will be formally specified as architectures that describe abstract views of processes and their communication. In subsequent *low-level designs* one commits to particular concrete component middleware technologies to support distribution. The technical problem we address is how one formalizes and establishes security requirements of high-level designs that can later be refined to low-level (implementation oriented) designs. Part of the challenge here is that different middleware technologies come with their own security models (e.g., the CORBA security model, given by the OMG), which must be part of the formalization and play a role in the proof of refinement.

CASE Supported Formal Methods for Distributed Object Systems UML is a popular semi-formal language used in modeling distributed object systems. With the company Interactive Objects GmbH, we have started a project investigating how formal specification languages (like OCL, Z, or VDM) can be incorporated in UML-based case-tools. In particular, we explore formal extensions of UML that allow pre- and post-conditions to be specified as well as invariants of functions calls. The goal is to support "light-weight" formal methods during design, for example, using the formalized object constraints as a basis for automatic generation of program invariants and test-cases.

Tool Support for Combined Specification Formalism The last two projects described above involve reasoning using combinations of specification formalisms. In two additional projects, we are working on foundations for supporting this kind of combined reasoning.

First, we are part of a European Union (IST) Working group on *Computer-Assisted Reasoning Based on Type Theory*. The aim of the working group is to develop formal reasoning tools based on type theories and typed languages and to explore their applications in domains such as program verification and synthesis as well as formal machine-supported mathematics.

Second, with the Computer Systems Laboratory at SRI (California, USA), we are investigating tool support for model checking systems specified in heterogeneous languages. This work involves, in particular, the use of rewriting logic (a rich, executable, specification language) as a unifying specification language for combining formalisms, the use of abstraction techniques to reduce systems specified in rewriting logic to finite state systems, and the integration of model checking techniques within the Maude system (an optimized implementation of rewriting logic).

Tools

To test the effectiveness of formal methods and to apply them to realistic industrial scale problems, part of the research in our group is on tool development.

Below we list some of the systems we have developed for this purpose.

Byte Code Verification by Model Checking: A system (described above) developed together with Deutsche Telekom for model checking security properties of the Java Card subset of Java Byte Code.

BMC: A counter-model generation system for finding errors in systems specified in a powerful monadic second-order logic on strings (M2L-Str). The BMC system reduces specifications of systems and their properties to propositional satisfiability problems.

BLIF2MONA: A verification system for BLIF (the Berkeley Logic Interchange Format), a hardware description language designed for the hierarchical description of sequential circuits. Circuit descriptions are translated into the decidable monadic second-order logic on strings, for which we provide tool support (the MONA and BMC systems) for circuit simulation, testing, and verification.

SML_TK, IsaWin, and TAS: A suite of tools for building graphical interfaces for formal methods. SML_TK provides a TCL/TK interface to the programming language SML. IsaWin provides a graphical interface for theorem proving with the Isabelle system. And TAS provides, on top of these, a system for transformational program development where programs can be interactively transformed by graphically supported transformations, with corresponding justifications carried out behind-the-scenes in Isabelle. We have also developed logical embeddings for the specification languages Z and CSP in higher-order logic, which can be used inside TAS and IsaWin.

Further Information

Further information concerning teaching and research activities may be found on the group's WWW-page `www.informatik.uni-freiburg.de/~softech` or by writing to:

Contact Address:
Prof. Dr. David Basin
Institut für Informatik
Albert-Ludwigs-Universität Freiburg
Universitätsgelände Flugplatz
D-79110 Freiburg i. Br.