Summer School Marktoberdorf (1970-2010)
Software and Systems Safety: Specification and Verification

Muhammad Taimoor Khan

Doktoratskolleg Computational Mathematics
Johannes Kepler University
Linz, Austria

October 20, 2010

Outline

- Introduction
- Lectures/Talks
- Issues of Adaptable Software for Open-World Requirements by *Carlo Ghezzi*

Introduction

- History
  - Marktoberdorf (100km south of Munich)
  - Software Engineering Conference in Germany (1968)
  - Tony Hoare and E.W. Dijkistra
- Introduction
  - For two weeks (August 3-15, 2010)
  - Academic Activities
    - Lectures
    - Tutorials
    - Discussions
  - Entertainment
    - Visit to the Alps
    - Visit to the Brewery
    - A concert
    - A barbecue night

Model-Driven Development of Reliable Services by *Manfred Broy*

- Discrete Systems
  - Interface
  - Logical specification
- Architectures
  - Composition
  - Compositional reasoning
- Contracts
  - Assumption/Promise
  - Logical interpretation
  - Safety and Liveness
- Architectures
  - Design by assumption/promise
  - Generalizations

Unifying Models of Data Flow by *Tony Hoare*

- Unifying
  - Memory
    - shared/private
    - weakly/strongly consistent
  - Communication
    - synchronised/buffered
    - reliable/unreliable
  - Allocation
    - dynamic/nested
    - disposed/collected
  - Concurrency
    - threads/processes
    - coarse/fine-grained
- Dynamic behavior of a resource
- Sequential trace as a Graph
- Relations, relation operators, relation properties
- Relational calculus as laballed graph

Model Checking by *Doron Pelad*

- Modeling of software and hardware systems
- Software specification using temporal logic and Buchi Automata
- Translation between logic and automata
- Model Checking Algorithms
- How to make it work in practice: abstraction/reduction/BDDs

Issues of Adaptable Software for Open-World Requirements by *Carlo Ghezzi*

- Introduction to Software Evolution and new challenges
- Software architectures and languages for adaptation and evolution
- Formal methods and software adaptation and evolution

Software adaptation and evolution

- Pervasive Computing as future computing
  - Context-aware applications/systems
  - Software evolution needs to be supported
  - Service oriented architectures as a solution
- Design for change (Parnas)
  - interface (stable)
  - body (volatile/modifiable)
- Components developed by independent organizations
  - No control over components evolution
  - Middleware provides binding mechanisms
- *Adaptation* is the ability of software to detect changes and react to them in a self-managed manner
- *Evolution* requires the designer in the loop
- Challenges
  - Can we support continuous adaptation and evolution without compromising dependability?
  - To identify the invariant properties that should be preserved by changes and ensure that they hold

Adaptation and software architectures

- Logically global coordination space acts as a mediator for composition
- Components remains decoupled (no explicit name binding)
  - publish-subscribe model
  - tuple-space model
- Publish-subscribe model
  - Event broadcasting to all registered components
  - No explicit naming of target component
  - Different kinds of guarantees possible
  - Easy integration strategies
  - Asynchronous communication
  - Problems with ordering of events
  - Understanding such a system and reasoning about its correctness maybe hard

SAVVY

- Service Analysis, Verification and Validation methodologY for Web Services (SAVVY)
  - Assumption-promise based approach
    - A service integrator *assumes* that the external services used in the composition satisfy their stated specification
    - Under this assumption, the system is designed to *promise* a certain service to its clients
  - Since external services may deviate their specification
    - A monitor does run-time verification
    - Suitable reactions may be activated
  - Supports verified composition of services
  - Compositions are guaranteed to satisfy certain global correctness properties
  - External services as abstract services with assumed behavior specification

Assertion Language for BPEL pRocess inTeractions (ALBERT)

- ALBERT
  - A linear temporal logic language
  - Variables correspond to BPEL variables
  - State a triple (V, I, t), where
    V is a set of <var, val> pairs
    I is a location in the workflow: set of labels
    t is the time at which the state is generated
  - can express assumptions and promises
  - can be used for design-time (verification)
  - can be used for run-time (monitoring+run-time verification)
  - It predicates on variables
  - Classical boolean operators and quantifications
  - Future Temporal Operators
    - Becomes, Until, Within
  - Functions
    - elapsed, past, count, ...

Requirements Models for System Safety and Security by *Connie Heitmeyer*

- Modeling and formal specification of requirements
- Consistency and completeness checking of requirements
- Simulation of requirements to check their validity
- Generating invariants from requirements specifications
- Formal verification of requirements
- Testing and automatic code generation based on an operational requirements model
- Modeling and analyzing systems for critical properties (e.g. security and fault-tolerance)

Formal Methods and Argument-based Safety Cases by *John Rushby*

- Purposes of Formal Methods
  - Verification
  - Consistency and completeness checking
  - Exploration, synthesis, test generation
- Hazard and safety analysis (serious fault prevention)
- Abstraction and automation required
- Argument-based safety analysis
- Tool support (BMC)

Abstraction for System Verification by *Susanne Graf*

- Appropriate abstraction is the key for successful verification of programs/systems
- General verification is of high complexity task (state explosion)
- General framework for abstraction
- Using abstractions to (meaningfully) reason about large composed systems
- General contract framework to prove stronger properties
- Proving properties with top-down design constraints and bottom-up abstractions

Model-based Testing by *Ed Brinksma*

- Model-based testing (terminology and concepts)
- Derivation of functional tests from models in the form of input/output transition systems
- Theory and tools can be extended to deal with real-time behaviour in specifications, implementations and tests
- Test selection and coverage

From Concurrency Models to Numbers: Performance, Dependability, Energy by *Holger Hermanns*

- Compositional construction of probabilistic models
- Modelling principles for concurrent systems based on labelled transition systems (LTS)
- Algorithmic aspects of model checking for probabilistic extensions of CTL
- Extensions of the principal models with *cost* and *reward*
- Tool support for probabilistic model checking
- Selection of applications

Formal Verification by *John Harrison*

- Theorem Proving for Verification
- Propositional logic
- FOL and arithmetic theories
- Combining and certifying decision procedures
- Interactive theorem proving

Model-based Verification and Analysis for Real-Time Systems by *Kim Larsen*

- Introduction to Timed Automata
- Decidablity and symbolic verification
- Priced Timed Automata
- Timed Games and Interfaces
- Tool suppport (UPPAAL)