Specification Languages

Presented by Cecilia Ekelin

Purpose of the language

- To express the specification of the system to be designed
- To enable formal reasoning about the design
- To provide possibilities for tool support on modeling, validation and implementation

Implications on language design

- A high-level approach necessary to cope with system complexity
  - Should be possible to express typical concepts
- The language should be based on formal semantics (Models of Computation)
  - No assumptions about implementation
- Formal syntax required as input to tools
  - Should be intuitive to the user

Concepts of embedded systems

- Concurrency
  - Interleaved vs Parallel
  - Control vs Data oriented

- Hierarchy
  - Behavioral vs Structural

- Communication
  - Message passing vs Shared memory

- Synchronization
  - Synchronous vs Asynchronous

- Implementation
  - Software vs Hardware

- Time ?

Models of computation

- Synchronization (Communication)
  - Single vs Multi-thread
- Concurrency (Functionality)
  - Data vs Control-driven

Representations: language-oriented (graphs), architecture-oriented (FSM)

Languages

VLSI System Design:

- Hardware abstraction levels, timing and data flow computations
- Hardware Description Languages (HDLs)
- E.g., VHDL, HardwareC, SpecCharts, SpecC

Languages (continued)

Protocol specification:

- Formal description to enable verification
- LOTOS
  - Based on process algebra and abstract data types
  - Specification is executable
- SDL
  - Based on extended FSMs
  - Both graphical and textual modeling
- ESTELLE
  - Pascal-like programming language
  - Implementation details necessary

Languages (continued)

Reactive (real-time) system design:

- Need to guarantee (timely) response to events

- ESTEREL
  - Based on events
  - Synchronous time model

- LUSTRE, SIGNAL
  - Based on programmable automaton
  - Simple time aspects in LUSTRE but more advanced in SIGNAL

- Petri net tools
  - Based on Petri nets
  - Not always formally defined

Languages (continued)

Programming languages:

- Often lacking constructs for concurrency and timing
- Extensions break the language standards
- E.g., C, Ada, Java, Fortran

Languages (continued)

Formal methods:

- Offers high abstraction but perhaps not all necessary concepts
- VDM, Z
  - Based on set theory and predicate logic
  - “Lack of tools” (www.ifad.dk)
- B
  - Based on Abstract Machine Notation

Languages (continued)

Structural Analysis:

- Systematic approach for structuring code and data in software systems
- “Divide and conquer”
- E.g., OO, UML

Languages (continued)

Continuous languages:

- High-level modeling based on differential equations
- Used for DSP, mechanical and hydraulic design
- Large expressiveness makes verification and synthesis hard
- E.g., Matlab, Matrixx, Mathematica

Case Study: SDL

Hierarchy

- System
- Block
- Process
- Procedures

Case Study: SDL (continued)

Communication & Concurrency

- No global data
- Asynchronous signals
- Synchronous RPC:s
- Channels interface blocks and processes
- A signal is sent to an explicit process instances

Case Study: SDL (continued)

Time

- `time` and `duration`
- A process may start timers
- Timeouts are received as signals
- Timing can be simulated before implementation

Case Study: SDL (continued)

Implementation

- Data is described using ADT or ASN.1
- Easily converted to other languages
- Reuse possible

Tool support

- Editor
- Simulator
- Proover
- Debugger
- Prototyper

Heterogeneous modeling

- Different phases (specification, design, implementation)
- Different subsystems (protocols, signal processing, control tasks)

Multilanguage design: Select language for each component and perform integrated validation

Multilanguage validation

- Independent approach
  - Individual validation
- Integrated (compositional) approach
  - Translate each language into a general representation on which validation is performed
  - E.g., Polis environment which is based on Codesign FSM
- Coordinated (cosimulation) approach
  - Validate each component separately but within a common framework

Cosimulation models

- Data model
  - User-defined types ?

- Timing model
  - No time (functional validation)
  - Time (granularity)

- Synchronization (communication) model
  - Master-slave (direct connection)
  - Distributed (software “bus”)

- Interfaces
  - In framework and implementation

Example - Automotive application

Three levels: system, system architecture, cycle

- **System:** Electronics (SDL) and Mechanics (Matlab)
  - Determines external specification
- **Architecture:** Hardware (VHDL) and Software (C)
  - Validates partitioning and communication protocols
- **Cycle:** Gates and Binary code
  - Verifies timing behavior

Prototyping

Comments

- “Performance” measures (development, usability, turn-around time, cost)
- Generalization (tools, concepts)