Experience with Validation by Simulation, Automated Code Generation and Integration

'DASIA 97'
- Data Systems in Aerospace -
Sevilla, Spain
May 26 - 29, 1997

Rainer Gerlich
BSSE System and Software Engineering
Auf dem Ruhbühl 181
D-88090 Immenstaad
Phone: +49/7545/91.12.58
Mobile: +49/171/80.20.659
Fax: +49/7545/91.12.40
e-mail: gerlich@t-online.de
Computer-Integrated Validation

- through the platforms: validation support
  - simulation on host
  - execution on host
  - execution on target

- through the life cycle: coherent transitions
  - specification - design
  - design - coding
  - coding - module testing
  - module testing - integration
  - integration - acceptance

- through the application types
  - embedded (real-time) systems
  - MMI
  - databases
  - algorithms
The 2-Dimensional Life Cycle

**Life Cycle Phases**

- Specification
- Design
- Final System

**Platform**

- Simulation
- Host System
- Hardware-Software Integration

**Target System**

- Target Code

---

(C) Copyright Rainer Gerlich, BSSE 1997
Experience

- **validation support and coherent transitions**
  - very efficient after re-organisation and automation
  - from step to step (verification and validation)
  - from platform to platform
  - on-line walk-through during second half of this presentation

- **application types**
  - careful consideration needed
  - methods and tools do not cover every application type
  - strong in one area, weak in another area
  - when trying to apply a method / tool to the whole scope of the application: one may lose everything → no advantage at all
Example: SDL

- very strong and very efficient for behaviour and distributed systems
  - verification and validation
  - automation of verification and code generation

- problems with verification when large data structures are processed
  but only in case of verification by exhaustive simulation,
  no constraints / problems in other cases
  - behavioural verification does not terminate:
    values of data are added to system space: explosion of system state space
  - advantage of automated verification of behaviour is lost
  - remark: filtering does not help here

- considered solution: EaSySim II + future evolution
  - partition the problem (system) into application types
  - define interfaces between the partitions
  - apply the best method / tool to each application type
Executed Steps (1)

- **export data processing to C**
  - use the SDL-C interface
  - define data types in SDL, take them in C
  - do not allocate memory for (large) data in C
  - do only declare such data in SDL which impact behaviour

- **compress range of data which are used in SDL**
  - consider out-of-range condition for x (real, integer): \([X_L,X_U]\)
  - if we only need the decision: x is out-of-limit: true/false
    - introduce an operator:
      - \(x\text{OutOfLimit}: \rightarrow\) Boolean "extern C"
    - instead of implementing (pseudo-code, not SDL code)
      - \(x\text{ Real};\)
      - \(\text{if } (x>X_U \text{ OR } x<X_L) \text{ then } ....\)
Executed Steps (2)

- **export to C**
  - verification means are lost
  - disadvantage

- **consequence:**
  - apply appropriate verification method (formal method) to the exported part
  - "export via C interface" instead of "export to C"
  - do implementation in the best environment for each part of the system
  - import results via C interface: generated code
  - use generated code in SDL part for simulation and code generation phase

- **optimisation of partitioning approach:**
  - use the interfaces to provide test drivers for the exported parts
  - gives a better coverage of simulation for behavioural verification
Complementary Verification and Validation

behavioural, performance, operational V&V

code generation

EaSySim II SDL ObjectGEODE

interface transformation

C Interface

code integration

Ada

B / RAISE

SAO+

Data Bases

Tcl/Tk

Others

functional verification

code generation
Efficient Walk through the Platforms

- **on-line demo: walk from simulation to target**
  - example: data management system
    processor, bus, sensor, actuator

- **no need to wait for the very end of life cycle to move to target**
  - due to automated code generation capabilities of SDL / ObjectGEODE for different platforms
  - however: appropriate organisation needed
Application: On-Board Data Management System

- RSIM: Resource, Scenario & Interface Mgt.
- Operational and Test Commands
- Telemetry
- DMS Processor
- Sensor Data & Actuator_cmds
- Bus
- Sensor Data & Actuator_cmds
- Devices
- Sensor
- Actuator
The Demo Steps

- simulator (on Sparc)
  - installation
  - simulation in batch
  - random simulation
  - generation of MSC
  - exhaustive simulation

- execution on host (Sparc UNIX)
  - installation
  - code generation
  - execution and tracing

- execution on target (PC bare machine, VxWorks, WindView)
  - installation
  - code generation
  - execution, tracing and recording of task status
From Verification & Validation to Code Execution

- System representation: SDL model
- Verification & Validation: ObjectGEODE simulator
- Message Sequence Charts (MSC)
- Code Generation: UNIX / Linux platform, ObjectGEODE code generator
- Traces and MSC's
- Code Generation: target (80x86), PC / bare machine, ObjectGEODE code generator
- Traces
- WindView View Graphs

Installation, Execution & Validation

15 minutes
Performance Evaluation

- **resource consumption / Message Sequence Chart**
  - all devices consume independent resources
  - time stamps
    - start time of data acquisition
    - actual time when leaving a device
  - modelling bug is still included, is visualised by time stamps

- **recording on target system**
  - task activities are visible
  - bug detected for cyclic processing
    when measuring the time intervals between the cycles
Message Sequence Chart
Recording of Target System Execution

DMS processor
sensor
actuator
bus
Data Flow

```
11   RSIM
41   DMS processor Sensor Data Request | Sensor Data Reception | Act. Cmd. Sending
    4120 | 4123 | 4125
51   Sensor  Sensor Data Preparation | 5121 - 5122 |
61   Actuator
75   Bus  Bus —— Bus  Bus —— Bus  Bus —— Bus
```

Telemetry Data Processing 1128
Conclusions

- partitioning of system development and validation needed
  - best method and tool for a certain application type: separated verification
  - integration of system partitions in SDL
  - complete or advanced system validation in SDL
  - through all platforms

- partitioning ensures success of validation of a broader class of applications

- efficient walk through the platforms
  - after optimisation of organisation

- for detailed questions:
  - time for detailed walk-through