Using UML-based Framework to Integrate Real-Time Object-Oriented Programming Models

Carlos Eduardo Pereira e Leandro Buss Becker

cpereira@delet.ufrgs.br lbecker@inf.ufrgs.br

Electrical Engineering Department Computer Science Institute

Universidade Federal do Rio Grande do Sul - UFRGS
Brazil
Agenda

- Historical Overview
  - OO Requirements Specification and Validation
  - Methodology, AO/C++
  - SIMOO, SIMOO-RT

- Current Work
  - SIMOO/RT-Deployment Diagram
  - Integrated Scheduling, QoS
  - HW-Coprocessors

- Conclusions
Historical Perspective

→ Research work has started on 1990: PhD thesys

→ Context: few OOA methods (Coad/Yourdon, Shlaer/Mellor, OMT, ...), no UMLS

→ Main goal:

how to apply OO to the development of distributed real-time systems (DRTS) with special emphasys for industrial automation systems
Historical Perspective

- Results of a state-of-the-art analysis in 1991:
  - OO is very suitable for modeling DRTS
  - Semantic mapping of automation equipments
  - Reuse/Inheritance/Extensions

Major Drawbacks:
- Timing requirements specification
- Requirements validation and verification
- CASE tool support
- Runtime support for distributed objects
Historical Perspective

⇒ Proposal made in 1992 (presented at OOPSLA - Workshop on RT Analysis and Design Methods):
Notation for describing timing requirements
Based on RTL
Tradeoff: understandable vs. verifiable
Periodical activities, synchronization, end-to-end requirements (between events)
Historical Perspective

- Research work during 92-94:
  - Requirements verification (use of Constraint Propagation algorithms - CobaltBlue)
  - Requirements validation through simulation
  - RT-extension to C++ (AO-C++)
Historical Perspective

AO/C++

Main idea: mapping of OO "logical concurrency" to "physical concurrency" in RT-UNIX (QNX)

Use of keywords (keep it as simpler as possible)
active class Sensor {
private:
    Conveior REF theConveyor;  //references to other objects
    Arm REF theArm;
public:
    void DetectWorkPiece(cycle_t){  //Time-triggered Method
        // initialization code
        begin_cycle
        // cyclic operation
        end_cycle
    }
    void StoreWorkPiece(dead_l){  //Method with deadline
        // exception code
        begin_exception
        // exception code
        end_exception
    }
};

AO/C++ code example
Historical Perspective

AO/C++

Parser translates .ph .pC files into:
Skeleton, stubs, ...

=> similarities with CORBA IDL <=

Pre-processing transformations:

Sensor.(ph;pC)
active class Sensor

Parser

Sensor.h
class Sensor
class SensorProc

Sensor.C
class Sensor

SensorProc.C
class SensorProc

SensorMain.C
Historical Perspective

1996-1997: SIMOO Environment
Object-Oriented modeling tool
Simulation tool
Structure modeling: Classes and Instances diagram

SIMOO Model Editing Tool (MET)
Simulation/animation output:
Historical Perspective

1998-1999: SIMOO-RT

- Timing requirements specification
- Behavior definition with state-machines
- Functional specification with use-cases and MSD
- Consistence checking based in Meta-model
- Automatic Code Generation in AO/C++
Timing specification: special attributes

- Cyclic operations
- Timed operations (deadline)
- Default value for the class
- Exception handling code
Behavior Specification: State-Transition Diagrams
Behavior Specification: C++ Templates
Functional specification: use-cases editor
Meta-Model

Consistency Checking
RT-code Generation: AO/C++ program

AO/C++ Classes and Makefile

SIMOO-RT
  - Structure
  - Behavior
  - Functionalities

Pre-processor
Instrumented C++ Classes

C++ Compiler
RT Application
Current work

• Further extensions to SIMOO-RT:
  • Instrumentation: Gergeleit (ISORC 99 + special issue)
Instrumentation: Validation of the temporal requirements
Current work

• Further extensions to SIMOO-RT
  • Instrumentation: Gergeleit (ISORC 99 + special issue)
  • Use cases + EDFDs: Automation Object Identification (ISORC 00 + AARTC 00 best paper)
MOSYS Design Methodology

- Functional model is mapped into a graph description
- A graph partition is carried out according to different criteria, leading to different possible task clustering
- Clustering criteria:
  - min-cut algorithm (enhance object autonomy)
  - number of objects (suggested by the designer)
  - object weight balance (in the algorithm)
Current work

- Further extensions to SIMOO-RT
  - Instrumentation: Gergeleit (ISORC 99 + special issue)
  - Use cases + EDFDs: Automation Object Identification (ISORC 00 + AARTC 00 best paper)
  - Supervisory systems (WORDS 99)
Proposed Environment:

- Computer-based Real-time System
- Supervisory Model
- Visualization
- Industrial Plant
- OO Models
- Code Generation
- Control
- Simulation
- Operator
- Customer
- Designer

Multi/Mono Processor
Current work

- Further extensions to SIMOO-RT
  - Instrumentation: Gergeleit (ISORC 99 + special issue)
  - Use cases + EDFDs: Automation Object Identification (ISORC 00 + AARTC 00 best paper)
  - Supervisory systems (WORDS 99)
  - Deployment Diagram
Mapping Objects to Processors
Deployment diagram
Current work

• Further extensions to SIMOO-RT
  • Instrumentation: Gergeleit (ISORC 99 + special issue)
  • Use cases + EDFDs: Automation Object Identification (ISORC 00 + AARTC 00 best paper)
  • Supervisory systems (WORDS 99)
  • Deployment Diagram
• Code generator:
  • Linux/TCP, mLinux/CAN
  • Future work: TAO (RT-CORBA), RT-JAVA
Current work

• Further extensions to SIMOO-RT
  • Instrumentation: Gergeleit (ISORC 99 + special issue)
  • Use cases + EDFDs: Automation Object Identification (ISORC 00 + AARTC 00 best paper)
  • Supervisory systems (WORDS 99)
  • Deployment Diagram
  • Code generator:
    • Linux/TCP, mLinux/CAN
    • Future work: TAO (RT-CORBA), RT-JAVA
• Mapping timing requirements to QoS specifications
Current work

- Further extensions to SIMOO-RT
  - Instrumentation: Gergeleit (ISORC 99 + special issue)
  - Use cases + EDFDs: Automation Object Identification (ISORC 00 + AARTC 00 best paper)
  - Supervisory systems (WORDS 99)
  - Deployment Diagram
  - Code generator:
    - Linux/TCP, mLinux/CAN
    - Future work: TAO (RT-CORBA), RT-JAVA
  - Mapping timing requirements to QoS specifications
  - From QoS specs to Integrated Scheduling of Processes and Messages => Mode-change, adaptive schedule (based on sound RT scheduling theory)
Current work

• HW-Support
  • Low-cost board (microcontroller-based) for running distributed objects (RT-mLinux, CAN)
  • Scheduler co-processor (handling of event-based and time-based activation of tasks, scheduling, measurement of runtime execution times)
Final Remarks

• Drawbacks identified in 90s were only partly solved and remain a challenge
• RT-DOC has increased interest of research community (this workshop is a good example)
  • Good: more people and more funding
  • Bad: proliferation of non technical papers