POOSL for early system validation

Parallel Object Oriented Specification Language (POOSL)
- Light-weight modeling and simulation for early system validation
- Successful applications in high-tech companies

Originates from two Dutch research organizations:
- Eindhoven University of Technology, Department of Electrical Engineering
- ESI (TNO): research center with strong partnerships with high-tech companies

Eclipse Open Source Project, industrialized by Obeo
- Eclipse Modeling Project for model-based development technologies
- [projects.eclipse.org/projects/modeling.poosl](https://projects.eclipse.org/projects/modeling.poosl)
Outline

Purpose
POOSL Language
Tool features
Example from EU project TRANSACT
Purpose

Early system (architecture) validation using light-weight modeling and simulation
Early phases of system development

Requirements are often imprecise and incomplete

Many architectural decisions are taken

Goal: Shorten the development time by early system validation

Requirements
Architecture
Design
Realization
Unit test
System test
Validation

Unclear or suboptimal decisions may cause problems later on
Early system validation

Properties that need early validation:

- Functional requirements
- Architectural decomposition
- Performance indicators

Typical approaches in industry:

- Plain drawing tools
- Elementary calculations
- Physical or virtual prototypes

POOSL: Lightweight modeling and simulation

- Executable models that abstract from as many details as possible
- Provide fast insights into requirements and early design decisions
- Reduce the risk of expensive iterations during design, integration and testing
The POOSL Language

Hierarchical decomposition
Discrete-time process behaviour
Object-oriented data structures
Hierarchical decomposition

Communicating processes

cluster class IFESystem
  ports
  InAudio
  OutAudio
  InControl
  instances
    aircraftFrontServer : AircraftFrontServer
    seatTV : SeatTV
  channels
    { InAudio, aircraftFrontServer.InAudio }
    { aircraftFrontServer.OutAudio, seatTV.InAudio }
    { aircraftFrontServer.Out INTERRUPTION Command, seatTV.In INTERRUPTION Command }
    { OutAudio, seatTV.OutAudio }
    { InControl, seatTV.InControl }
Discrete-time process behavior

Send / receive a synchronous message
Parallelism
Non-deterministic choice
Time delay
Abort
Interrupt

```
process class AudioVideoPlayer

ports
 InAudio
 OutAudio
 InControl
 In INTERRUPTION Command

messages
 InAudio? Begin Announcement (Integer)
 InAudio? End Announcement (Integer)
 OutAudio? Begin Announcement (Integer)
 OutAudio? End Announcement (Integer)
 OutAudio? Begin Video (Integer)
 OutAudio? End Video (Integer)
 InControl? Start Video (Integer, Integer)
 In INTERRUPTION Command? Interrupt()
 In INTERRUPTION Command? Resume()

variables
 last Trace Start Time : Real

init
 Initialise()

methods
 Initialise()
 interrupt
 Play Videos() {
   In INTERRUPTION Command? Interrupt();
   Handle INTERRUPTION();
   last Trace StartTime := currentTime
 }

Play Videos() | video ID : Integer, length : Integer |
 InControl? Start Video (video ID, length);
 delay 1; // internal processing time
 last Trace StartTime := currentTime;
 OutAudio? Begin Video (video ID);
 delay length; // playtime of video
 OutAudio? End Video (video ID);
 Play Videos();

Handle INTERRUPTION() | id : Integer |
 sel
 InAudio? Begin Announcement (id);
 delay 1; // internal processing time
 OutAudio? Begin Announcement (id);
 InAudio? End Announcement (id);
 delay 1; // internal processing time
 OutAudio? End Announcement (id);
 Handle INTERRUPTION();
 or
 In INTERRUPTION Command? Resume()
```
Object-oriented data structures

Data class:
- **Super class:** Single class inheritance
- **Variables:** Protected class variables
- **Methods:** Atomic, deterministic operations

Built-in API and libraries:
- **Boolean**
- **Float, Integer, Real**
- **Char, String, JSON**
- **Console, FileIn, FileOut, Socket**
- **RandomGenerator**
- **Data collection structures**
POOSL Modeling Tool features
Modeling & simulation

Modeling:

- **Textual editing**
  - Navigation, content assist, file importing, etc.
- **Graphical editing**
  - Navigation, multiple views (class, composite structure), etc.
- **Model validation**
  - Type checking, quick fixes, warnings for likely problems, etc.

Simulation:

- **Interactive debugging**
  - Breakpoints, threads, stack frames, variables, sequence diagram, etc.
- **High-performance execution**
  - Launch from both IDE and command line, etc.
producer_consumer.poool [Simulated time: 4098]

system

ProducerConsumer

Bus

messages
/* Incoming message */
In?Message(Packet), /* Outgoing message */
Out!Message(Packet)

variables
   p : Packet
init
   Transfer();

methods
   /* Transfer messages */
   Transfer();
       In?Message(p);
       Out!Message(p);

   ///* Process class that consumes messages. */
process class Consumer
ports
   /* This is the only port; it is used for incoming messages */
   In
messages
/* Incoming message */
In?Message(Packet)
variables
   ID : Integer
init
   ReceivePacket();

methods
   /* Receive messages */
   ReceivePacket();(int p : Packet | In?Message(p) (ID := p getIdentity));
   ReceivePacket();

// Sequence Diagram

System
ProducerConsumer
Consumer
Bus

Message(Packet[2042])
Message(Packet[2043])
Message(Packet[2044])
Message(Packet[2045])
Message(Packet[2046])
POOSL example from EU project TRANSACT

transact-ecsel.eu
POOSL in TRANSACT: Use case cloud-assisted image guided surgery

- Advanced image processing in the cloud, while patient is on the table
- Response time should not keep surgeon waiting
How to meet SLA for Response Time?

*Impact of dynamic demands on auto-scaling needs for cloud resources*

So many hospitals?
So many operations?
So many cloud requests?

Stochastic, time-varying demand

How many cloud resources?
How to scale to meet timing?
How? Model the system and cloud workflow with POOSL!

POOSL model of Hospital(s)

POOSL model of Cloud
How? Simulate the workflow with POOSL!

14 hospitals, 5 rooms per hospital, 5 patients per room (3 in the morning, 2 in the afternoon)
How? Analyse the POOSL simulation – Response time distribution

1 cloud resource instance

1-3 cloud resource instances (dynamic scaling)
Try POOSL!

POOSL website: www.poosl.org/

Source code: github.com/eclipse/poosl

More about TRANSACT: transact-ecsel.eu
Thanks for your kind attention

Acknowledgement

The TRANSACT project (transact-ecsel.eu) has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007260. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Netherlands, Finland, Germany, Poland, Austria, Spain, Belgium, Denmark, Norway.