Open Research Webinars

Open Collaboration in European Research Projects

Early System Exploration: POOSL

Jos Hegge - ESI (TNO)

Presented by

RESEARCH @ ECLIPSE FOUNDATION

OW2

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

Outline

Purpose

POOSL Language

Tool features

Example from EU project TRANSACT

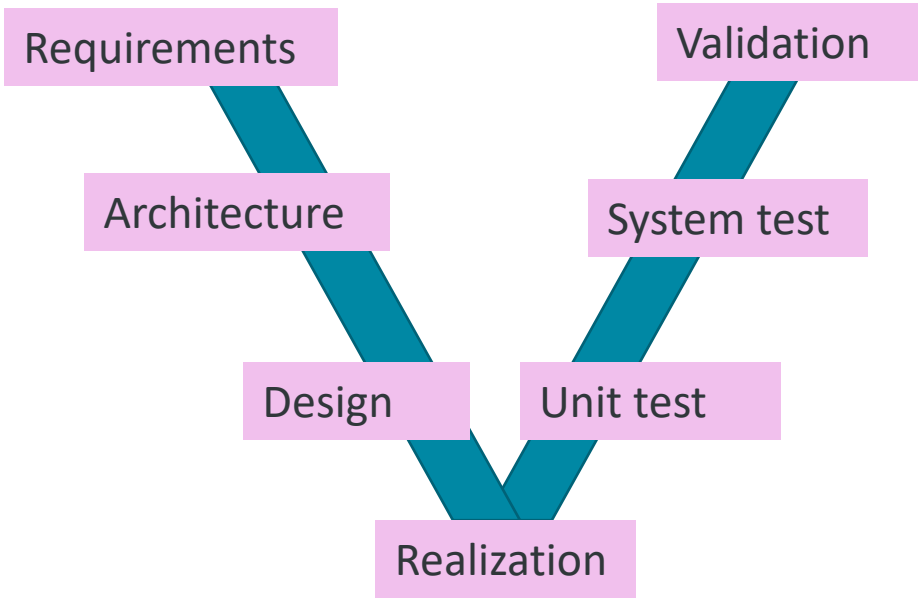
Purpose

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



Unclear or suboptimal decisions may cause problems later on

Goal: Shorten the development time by early system validation

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

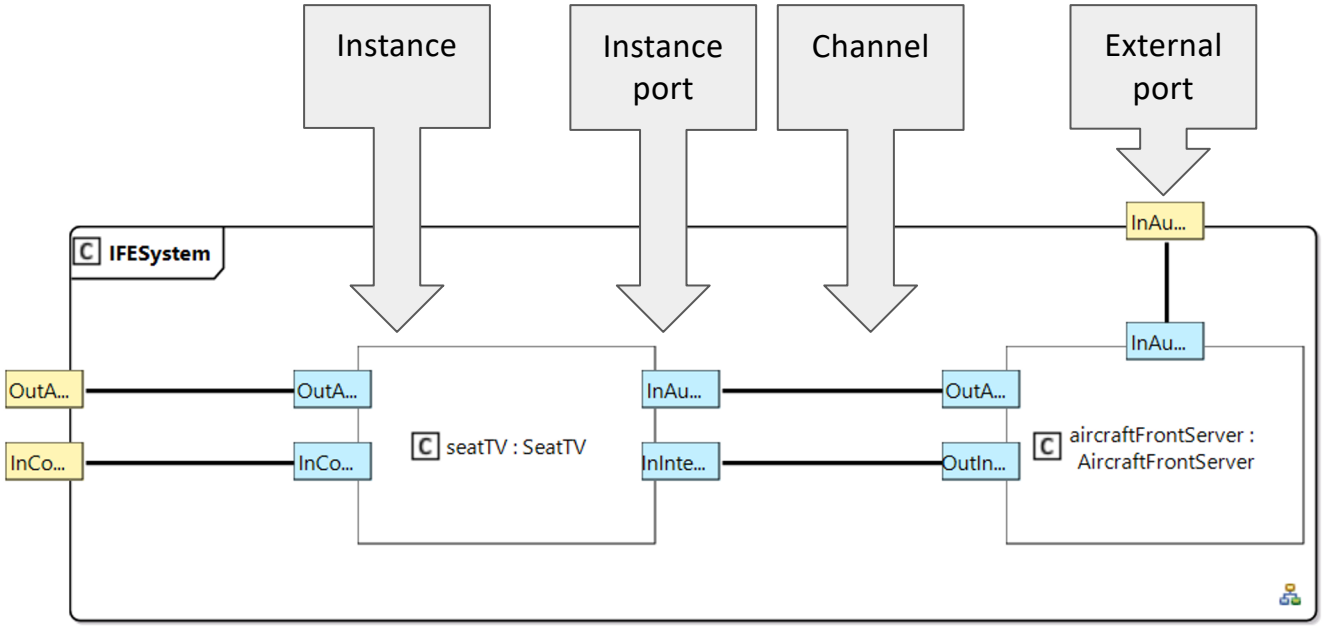
POOSL: Lightweight modeling and simulation

- **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




```
cluster class IFESystem
ports
    InAudio
    OutAudio
    InControl
instances
    aircraftFrontServer : AircraftFrontServer
    seatTV : SeatTV
channels
    { InAudio, aircraftFrontServer.InAudio }
    { aircraftFrontServer.OutAudio, seatTV.InAudio }
    { aircraftFrontServer.OutInterruptionCommand, seatTV.InInterruptionCommand }
    { OutAudio, seatTV.OutAudio }
    { InControl, seatTV.InControl }
```

Communicating processes

Discrete-time process behavior

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

```
process class AudioVideoPlayer
ports
    InAudio
    OutAudio
    InControl
    InInterruptionCommand
messages
    InAudio?BeginAnnouncement(Integer)
    InAudio?EndAnnouncement(Integer)
    OutAudio!BeginAnnouncement(Integer)
    OutAudio!EndAnnouncement(Integer)
    OutAudio!BeginVideo(Integer)
    OutAudio!EndVideo(Integer)
    InControl?StartVideo(Integer, Integer)
    InInterruptionCommand?Interrupt()
    InInterruptionCommand?Resume()
variables
    lastTraceStartTime : Real
init
    Initialise()()
methods
```

```
Initialise()()
    interrupt
        PlayVideos()()
    with (
        InInterruptionCommand?Interrupt();
        HandleInterruption()();
        lastTraceStartTime := currentTime
    )

PlayVideos()() | videoID : Integer, length : Integer |
    InControl?StartVideo(videoID, length);
    delay 1; // internal processing time
    lastTraceStartTime := currentTime;
    OutAudio!BeginVideo(videoID);
    delay length; // playtime of video
    OutAudio!EndVideo(videoID);
    PlayVideos()()

HandleInterruption()() | id : Integer |
    sel
        InAudio?BeginAnnouncement(id);
        delay 1; // internal processing time
        OutAudio!BeginAnnouncement(id);
        InAudio?EndAnnouncement(id);
        delay 1; // internal processing time
        OutAudio!EndAnnouncement(id);
        HandleInterruption()()
    or
        InInterruptionCommand?Resume()
    les
```

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

```
data class LatencyObserver extends Observer
variables
    count : Integer
    sum : Real
    numberOfIterations : Integer
    prefix : String
    console : Console
methods
    @Init
    init(iterations : Integer, str : String) : Observer
        count := 0;
        sum := 0.0;
        numberOfIterations := iterations;
        prefix := str;
        console := new(Console);
        return self register

    result() : String
        return (sum / count asReal) printString

    store(value : Real) : Observer
        count := count + 1;
        sum := sum + value;
        if count == numberOfIterations then
            self complete
        fi;
        console writeLine(prefix + self result);
        return self
```

POOSL Modeling Tool features

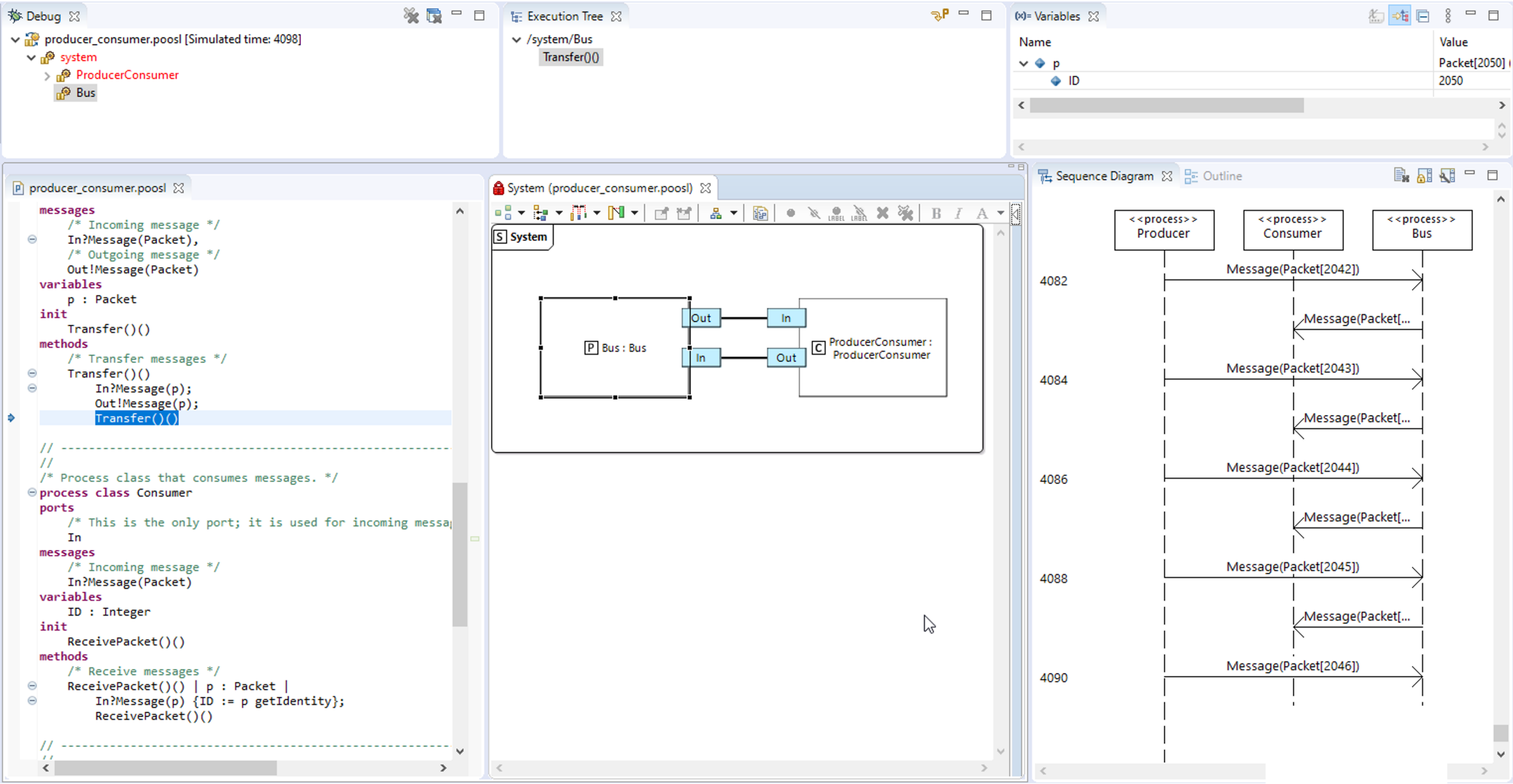
Modeling & simulation

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.


Debug
producer_consumer.poosl [Simulated time: 4098]
system
ProducerConsumer
Bus
Execution Tree
/system/Bus
Transfer()()
Variables
Name
Value
p
Packet[2050]
ID
2050
producer_consumer.poosl
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);
Transfer()()
// ----------
//
/* Process class that consumes messages. */
process class Consumer
ports
/* This is the only port; it is used for incoming messa
In
messages
/* Incoming message */
In?Message(Packet)
variables
ID : Integer
init
ReceivePacket()()
methods
/* Receive messages */
ReceivePacket()() | p : Packet |
In?Message(p) {ID := p getIdentity};
ReceivePacket()()
// ----------
System (producer_consumer.poosl)
System
Out
In
Bus : Bus
In
Out
ProducerConsumer : ProducerConsumer
Sequence Diagram
Outline
<<process>> Producer
<<process>> Consumer
<<process>> Bus
4082
Message(Packet[2042])
Message(Packet[...
4084
Message(Packet[2043])
Message(Packet[...
4086
Message(Packet[2044])
Message(Packet[...
4088
Message(Packet[2045])
Message(Packet[...
4090
Message(Packet[2046])

POOSL example from EU project TRANSACT

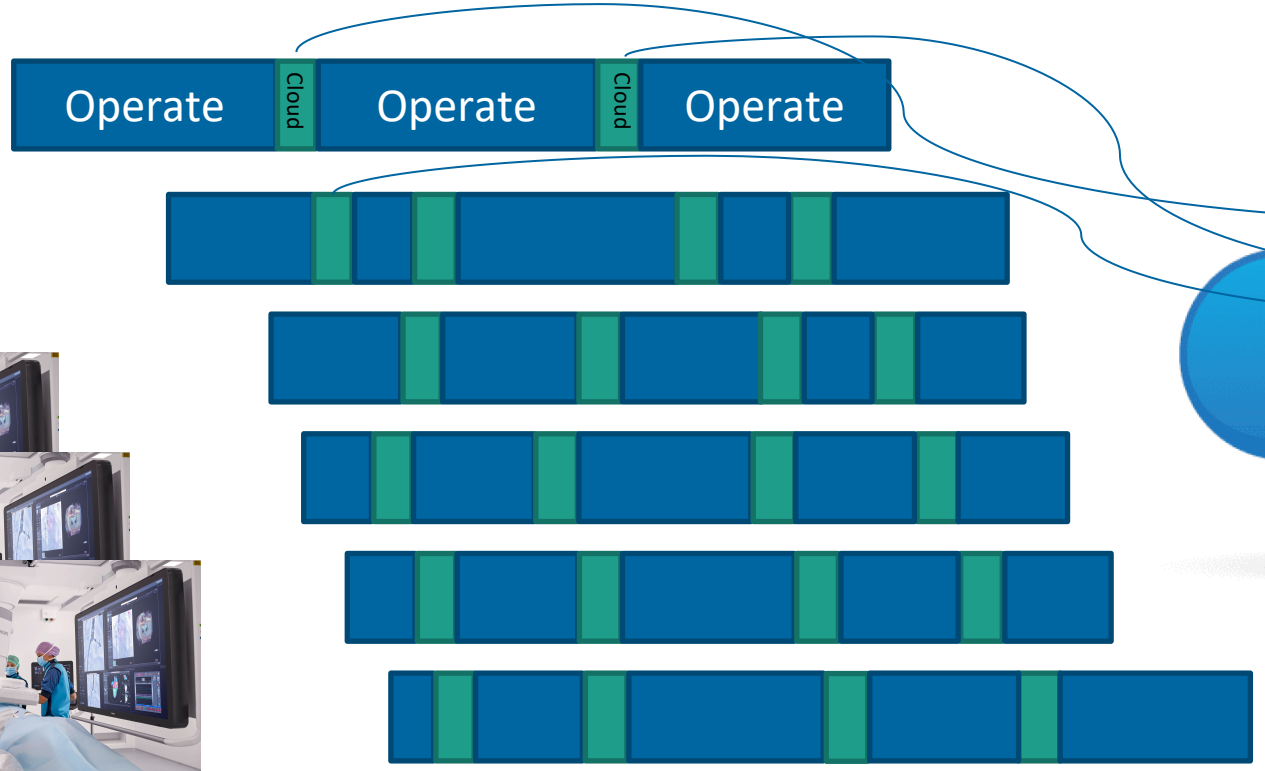
transact-ecsel.eu

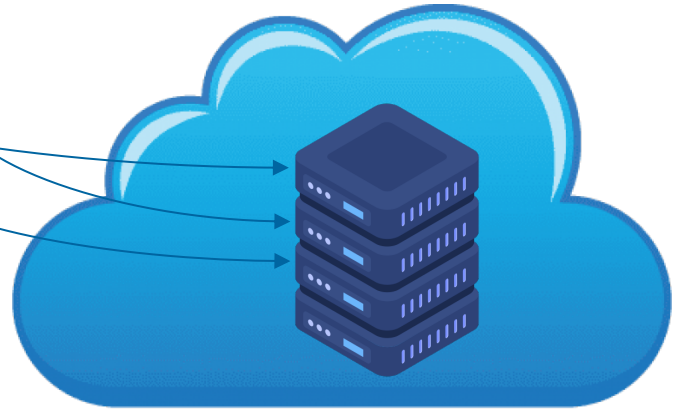
POOSL in TRANSACT: Use case cloud-assisted image guided surgery

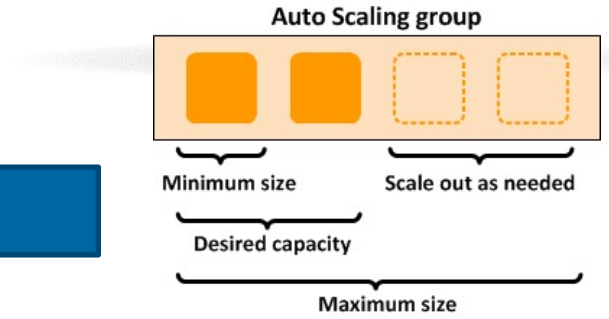
- Advanced image processing in the cloud, while patient is on the table
- Response time should not keep surgeon waiting

How to meet SLA for Response Time?

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







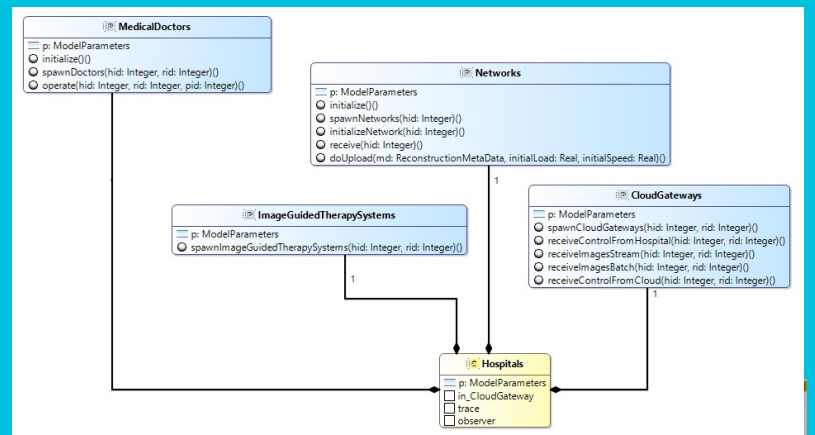
So many hospitals?
So many operations?
So many cloud requests?

Stochastic, time-varying demand

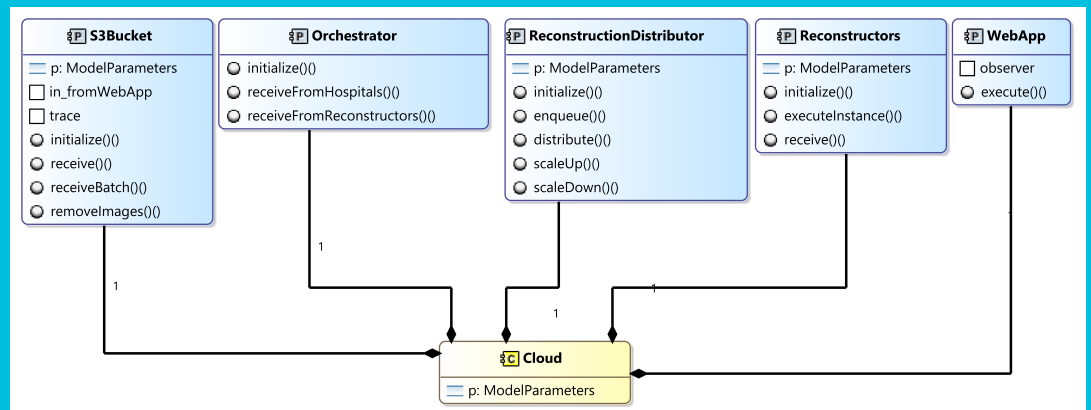
How many cloud resources?
How to scale to meet timing?

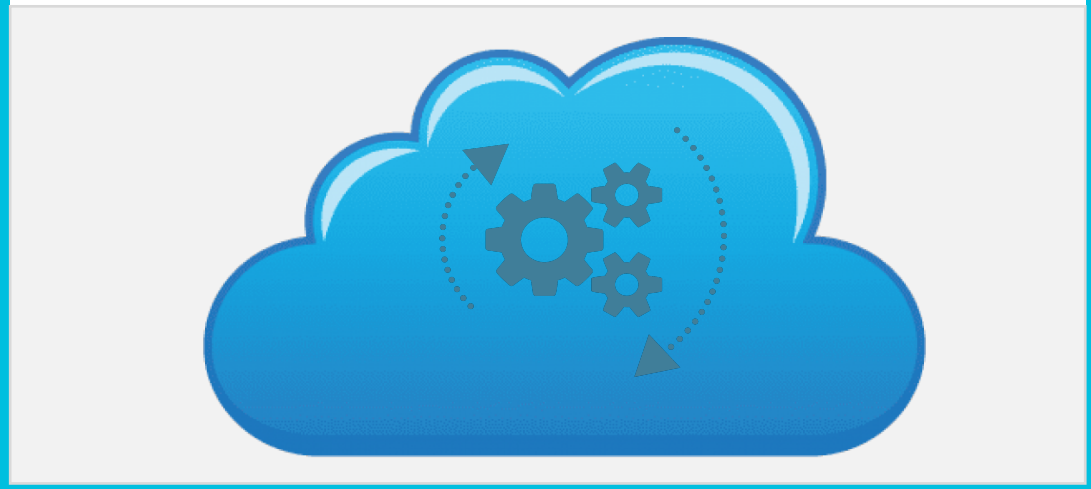
How? Model the system and cloud workflow with POOSL!

POOSL model of Hospital (s)



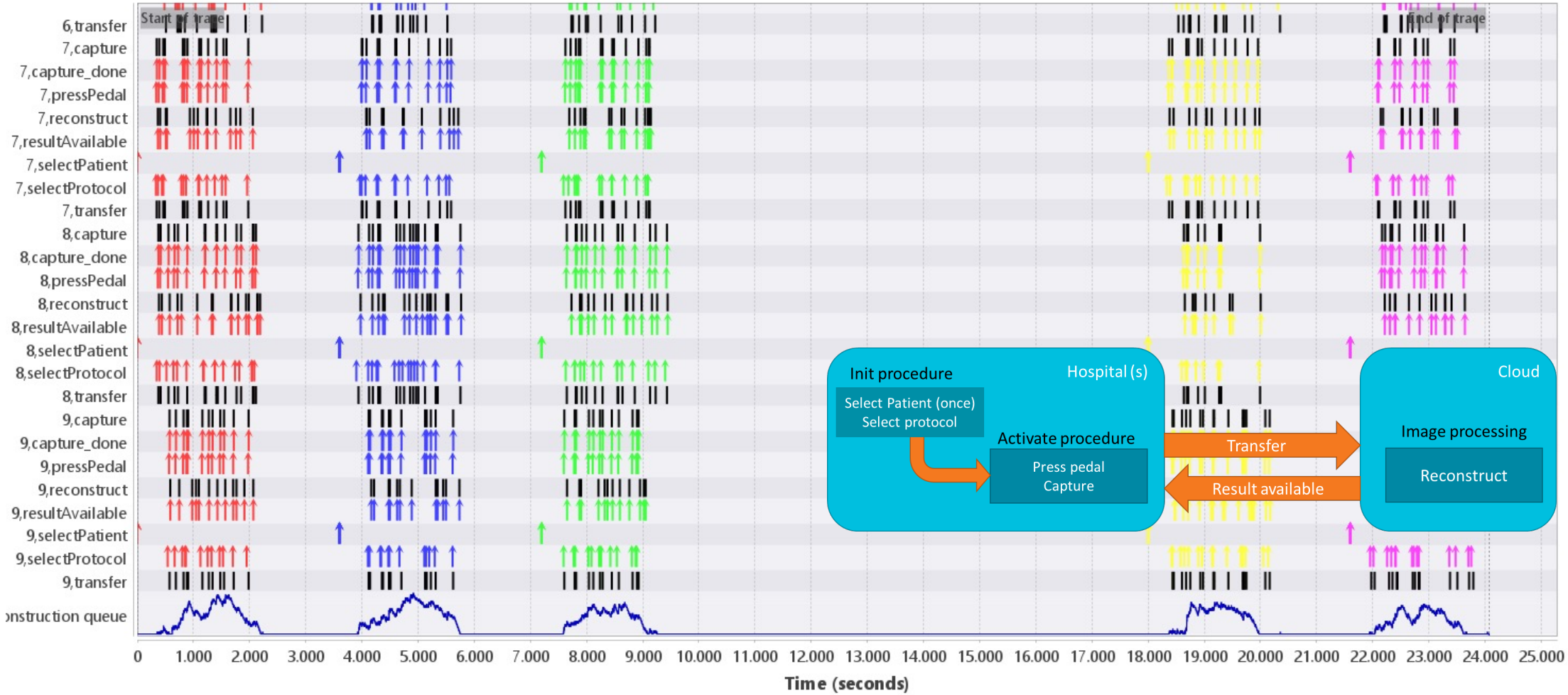
POOSL model of Cloud



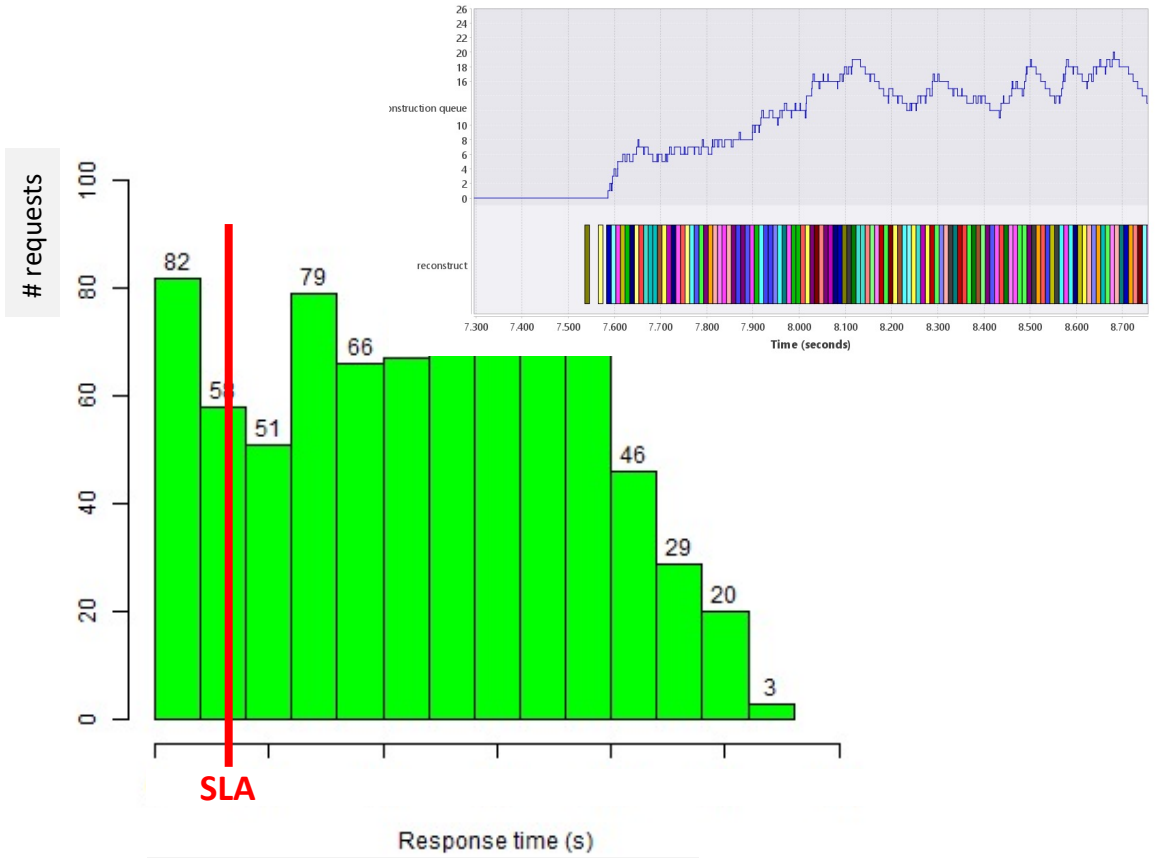


How? Simulate the workflow with POOSL!

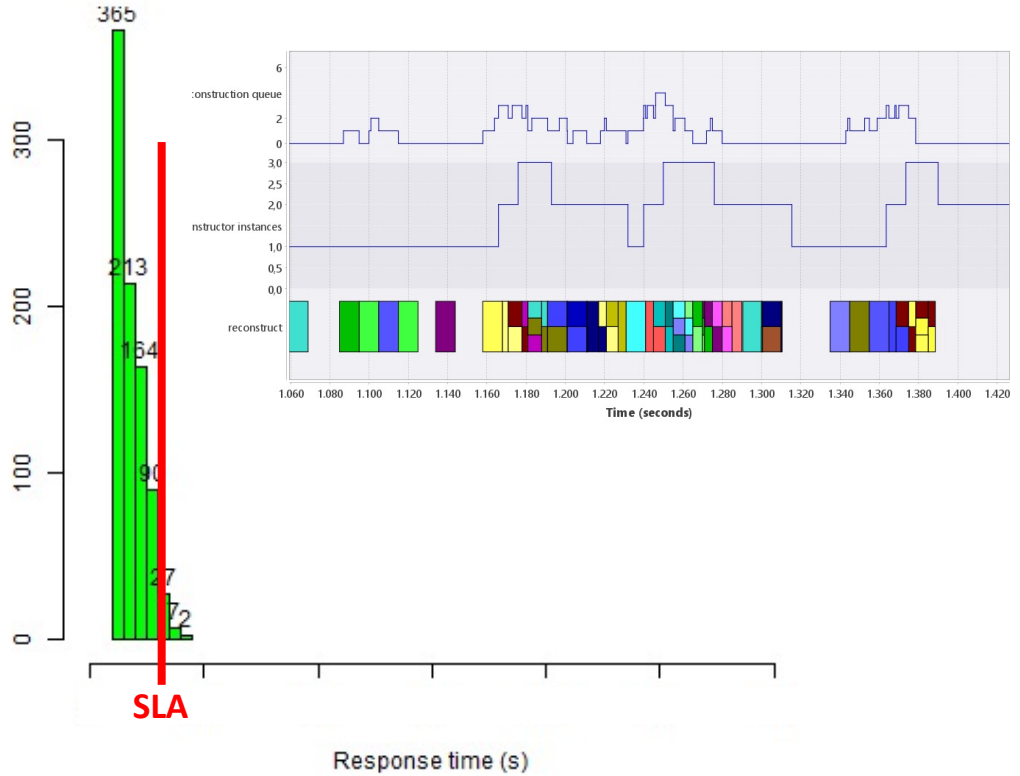
14 hospitals, 5 rooms per hospital, 5 patients per room (3 in the morning, 2 in the afternoon)



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

ECLIPSE FOUNDATION

OBEO

ESI An initiative of industry, academia and TNO

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

TRANSACT

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.