Open License Society

www.OpenLicenseSociety.org

Unifying and systematic system development methologies
with trustworthy embedded components

Eric.Verhulst@OpenLicenseSociety.org

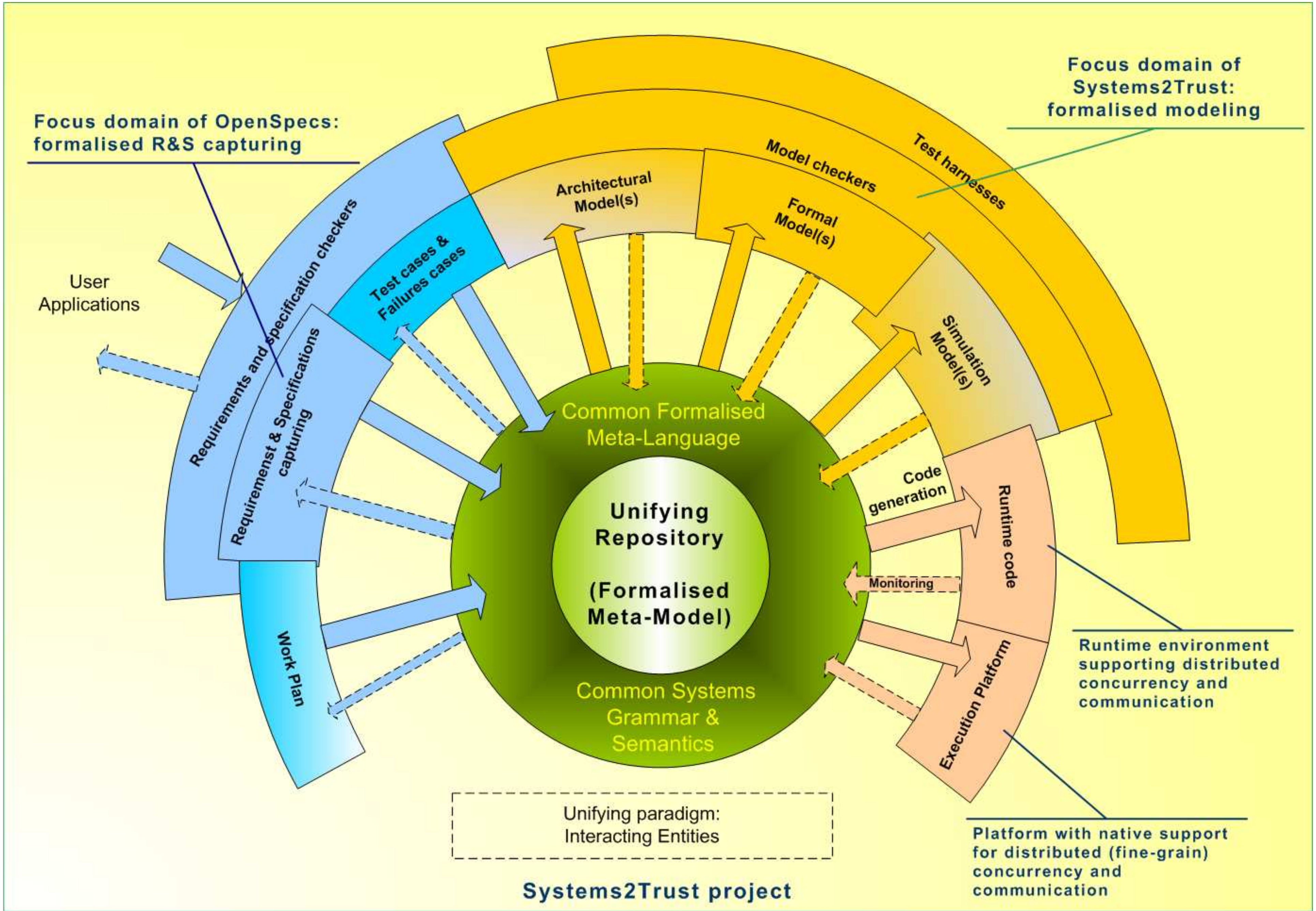
OpenComRTOS: An Ultra-Small Network Centric Embedded RTOS Designed Using Formal Modeling

Eric Verhulst and Gjalt de Jong
Open License Society
Leuven, Belgium
{eric.verhulst,gjalt.dejong}@OpenLicenseSociety.org

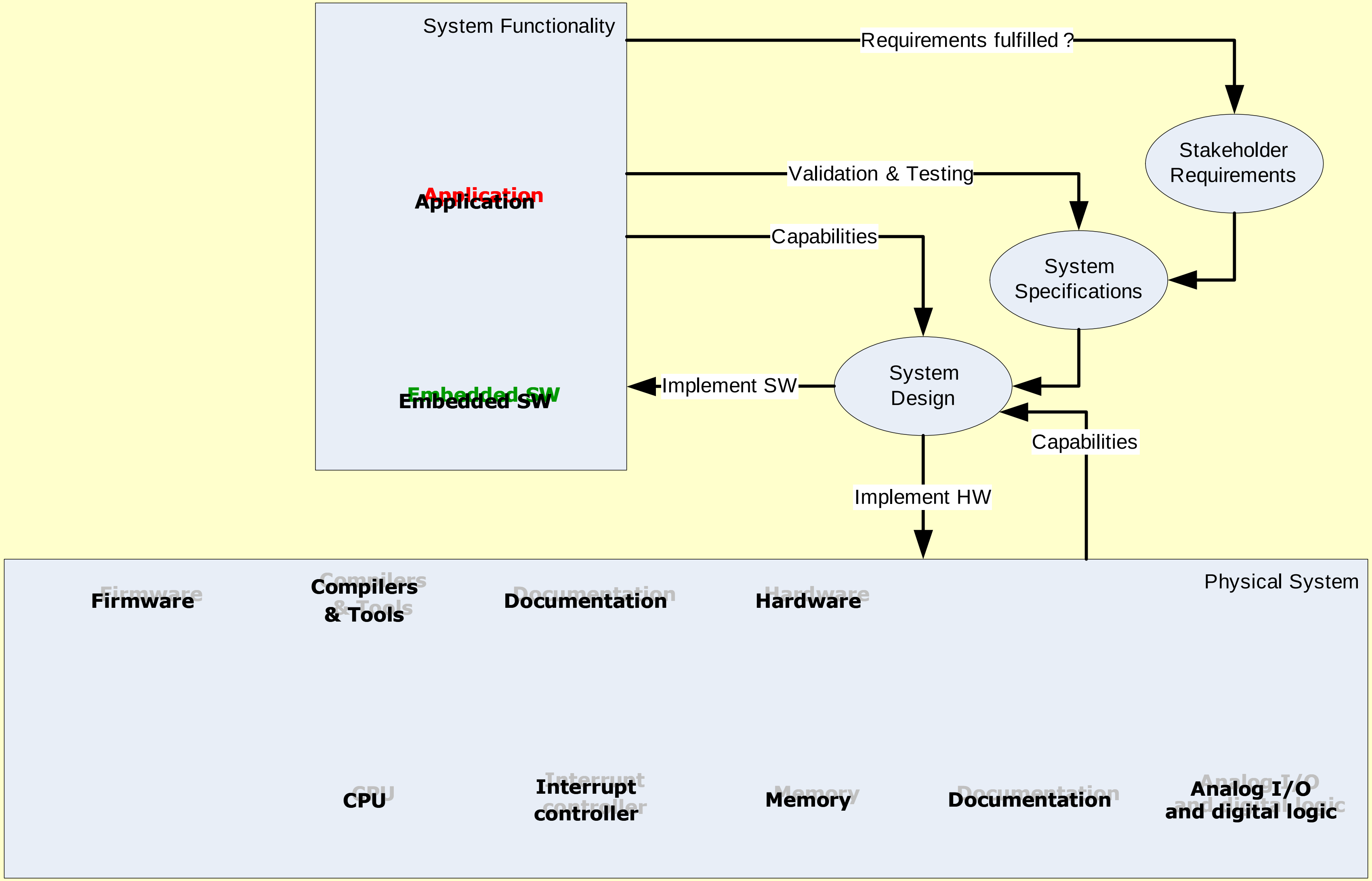
Who is Open License Society?

- Privately funded R&D institute
  - Leuven (BE), Berdyansk (UA)
  - Industrial sponsors
  - IWT project funding for OpenComRTOS
- Why: 70 % of all SE projects do not deliver
- Objectives
  - Systematic & Unified Systems Engineering Methodology
  - ‘Interacting Entities’ paradigm at all levels:
    - OpenComRTOS as runtime environment (formally developed)
  - Implies ‘Trustworthy Components’
    - => Open License (source code + all design, test, …. docs)
- Focus:
  - Embedded Systems:
    - Constraints driven development
    - Real-time, distributed, hardware & software, …

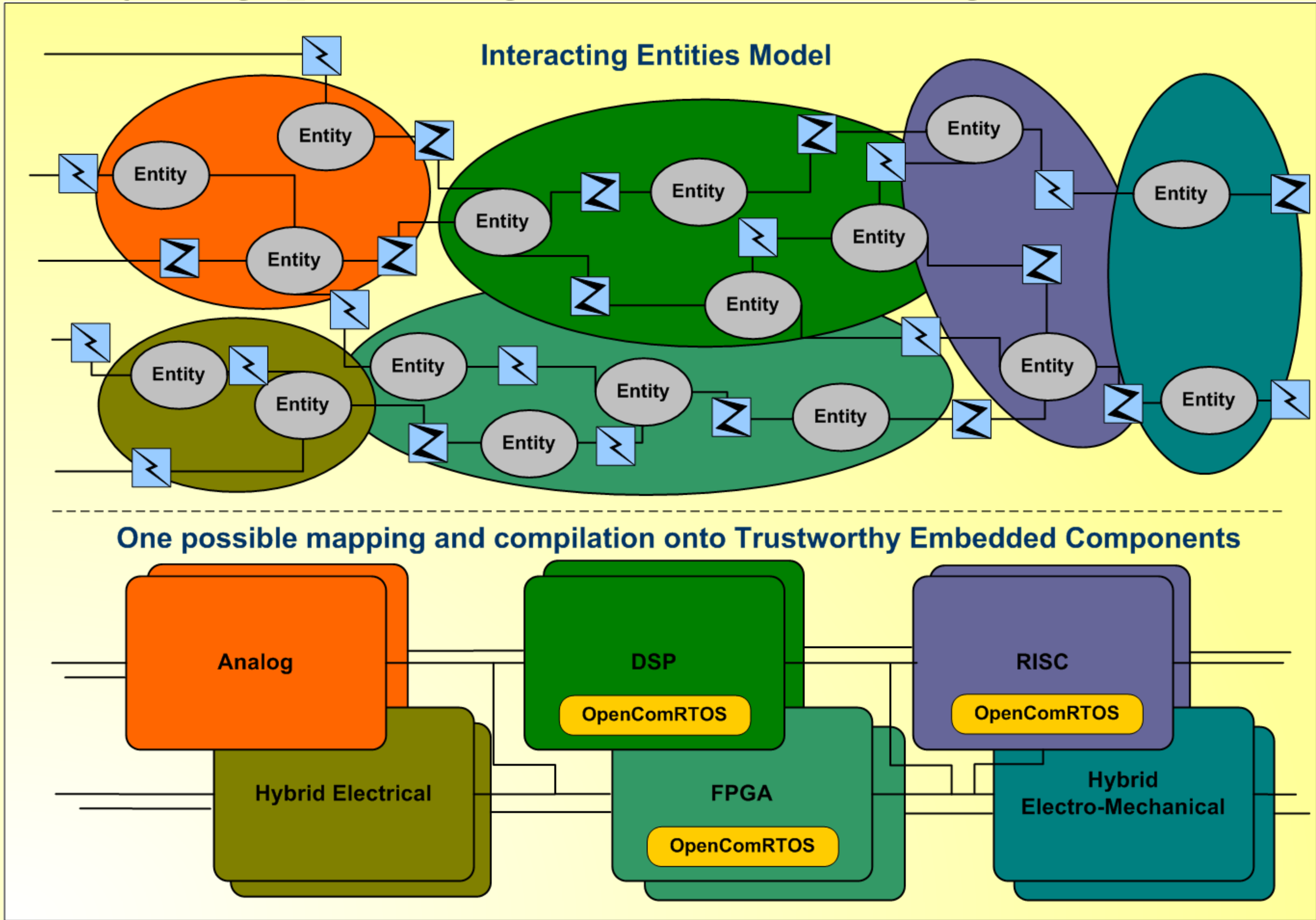
OPEN LICENSE SOCIETY


Focus domain of OpenSpecs:
formalised R&S capturing
Focus domain of
Systems2Trust:
formalised modeling
User
Applications
Requirements and specification checkers
Requirement & Specifications
capturing
Test cases &
Failures cases
Architectural
Model(s)
Model checkers
Test harnesses
Formal
Model(s)
Simulation
Model(s)
Code
generation
Runtime code
Monitoring
Execution Platform
Work Plan
Common Formalised
Meta-Language
Unifying
Repository
(Formalised
Meta-Model)
Common Systems
Grammar &
Semantics
Runtime environment
supporting distributed
concurrency and
communication
Platform with native support
for distributed (fine-grain)
concurrency and
communication
Unifying paradigm:
Interacting Entities
Systems2Trust project

SE Process dependency graph



Unifying paradigm: Interacting Entities



OpenComRTOS project objectives

- Funded R&D project (IWT, Flanders)
  - Lancelot Research: management, commercialisation
  - Open License Society: technology development
  - University Gent (INTEC, Prof. Boute): formal modeling
  - University Berdyansk: tools and formal validation
  - Melexis: co-sponsor and first user (16bit uC)
- GUI tools:
  - graphical modeling/development environment
- Goal:
  - Develop Trustworthy <u>distributed</u> RTOS
    - Follow OLS SE methodology
    - Formal verification & analysis: formal modelling
  - Scalable distributed RTOS
  - Verify benefits and issues of using Formal Modeling

Some requirements

- Targets:
  - Single chip, tightly coupled: multi-core
  - Multi-chip, tightly coupled: parallel processors on board
  - Multi-boards, multi-rack: using backplane interconnects
  - Distributed: using LAN and WAN
  - Host node
- Programming models:
  - "Interacting Entities"
  - "Virtual Single Processor":
    - transparent for topology
    - Supporting heterogenous targets
  - Distributed real-time
  - Safe, secure
  - Small code size, low latency (=high performance)

OpenComRTOS systems grammer

OpenComRTOS **IS_DEFINED_BY**

Configuration (1) // The root node of XML file

Configuration **IS_DEFINED_BY** // Nodes of configuration section

Parameters (1) **AND** // *Attributes* of the configuration section of XML file

SystemTasks (4) **AND** // Kernel, Idle, Rx or Tx

ApplicationTasks (1-N) **AND**

Ports (1-N) **AND**

Nodes (1-N) **AND**

Links (1-N)

Configuration **HAS_ATTRIBUTES** // Parameters

DataSize (1) **AND** // Packet data size (in bytes)

NodeIdSize (1) // Length of Node identifier (in bits)

SystemTask **CAN_BE** // Type of system task

KernelTask **OR**

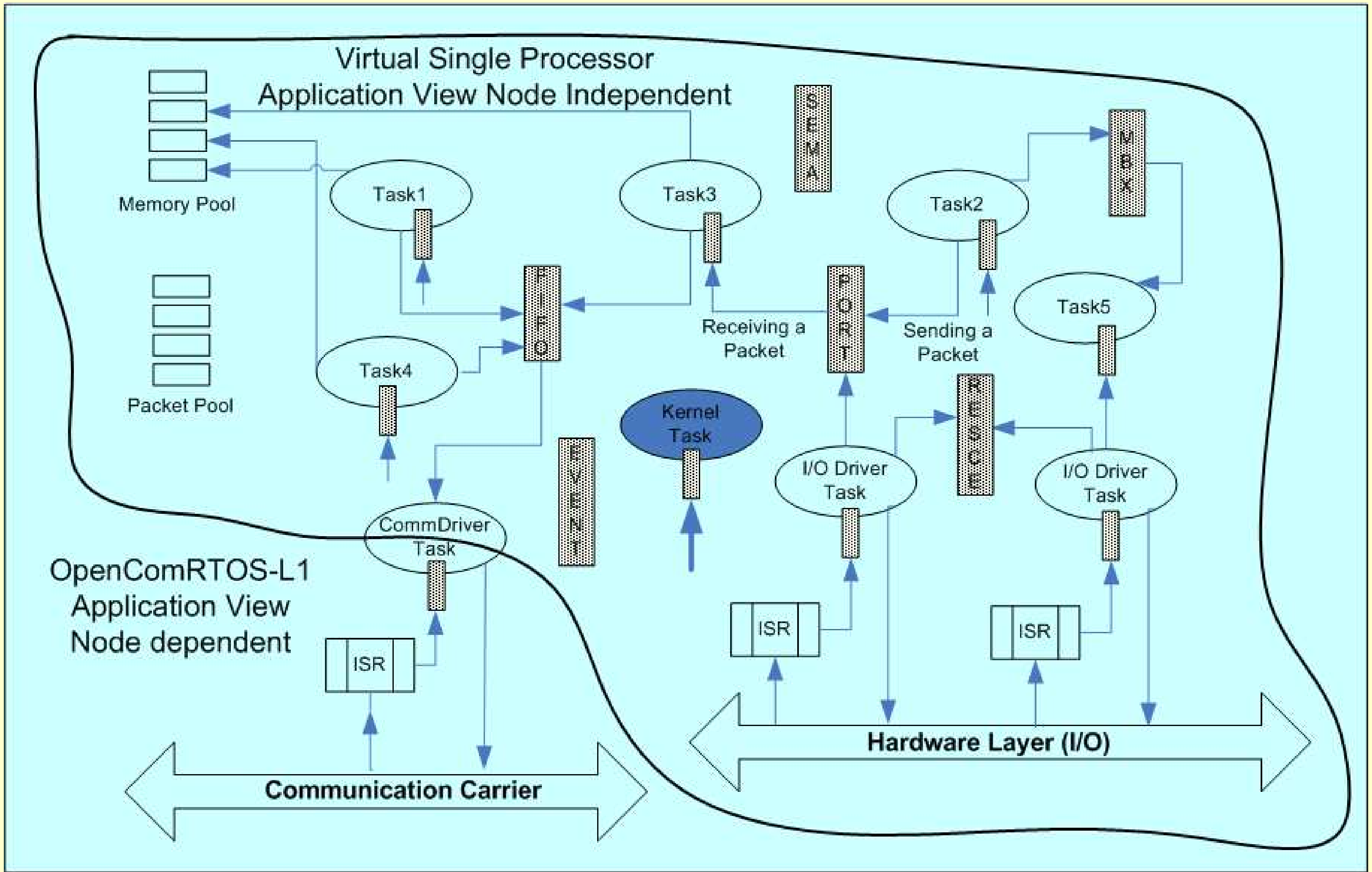
IdleTask **OR**

RxTask **OR**

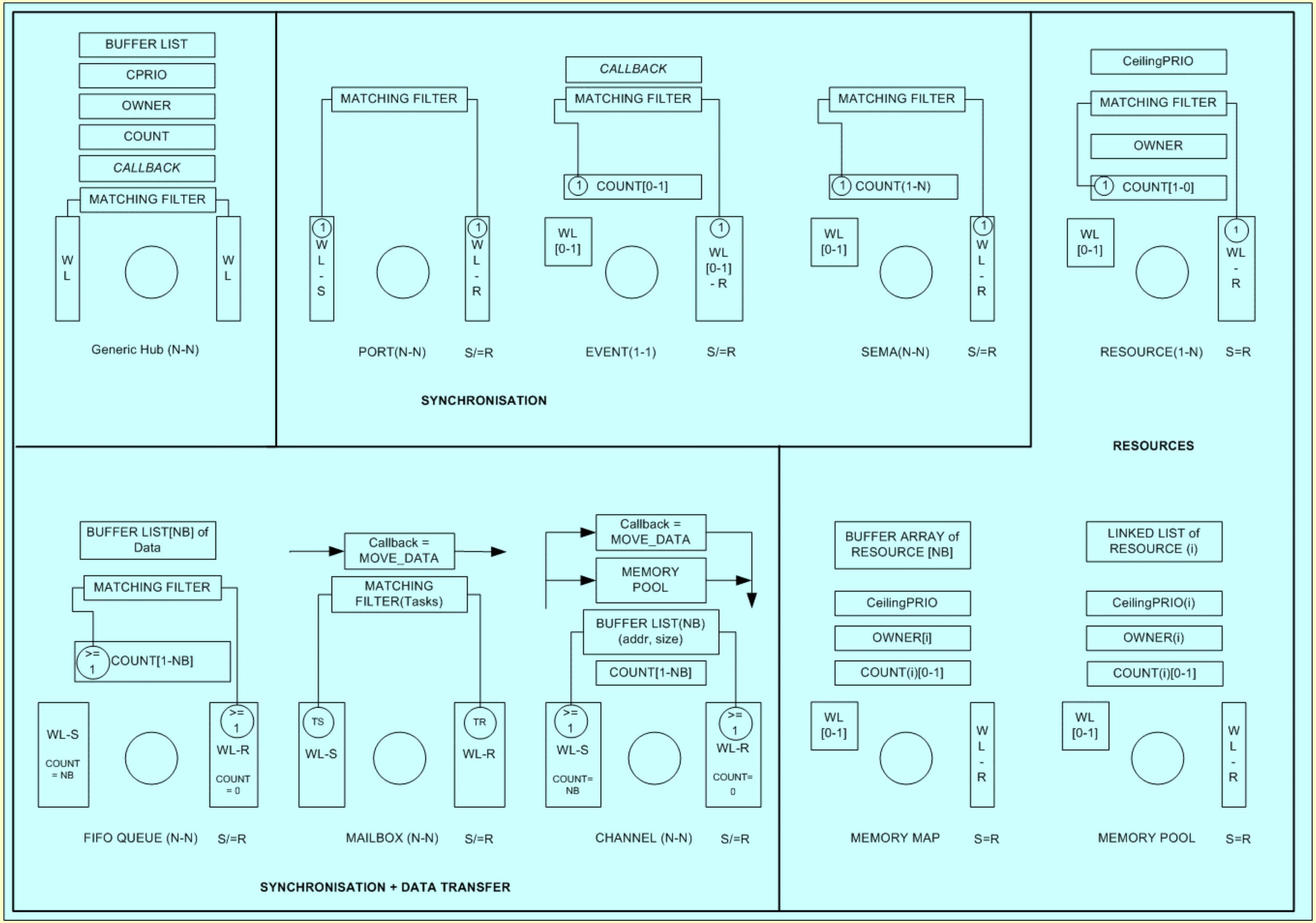
TxTask

Etc.

L1 application view



All RTOS Entities: variation on a theme



Generic hub

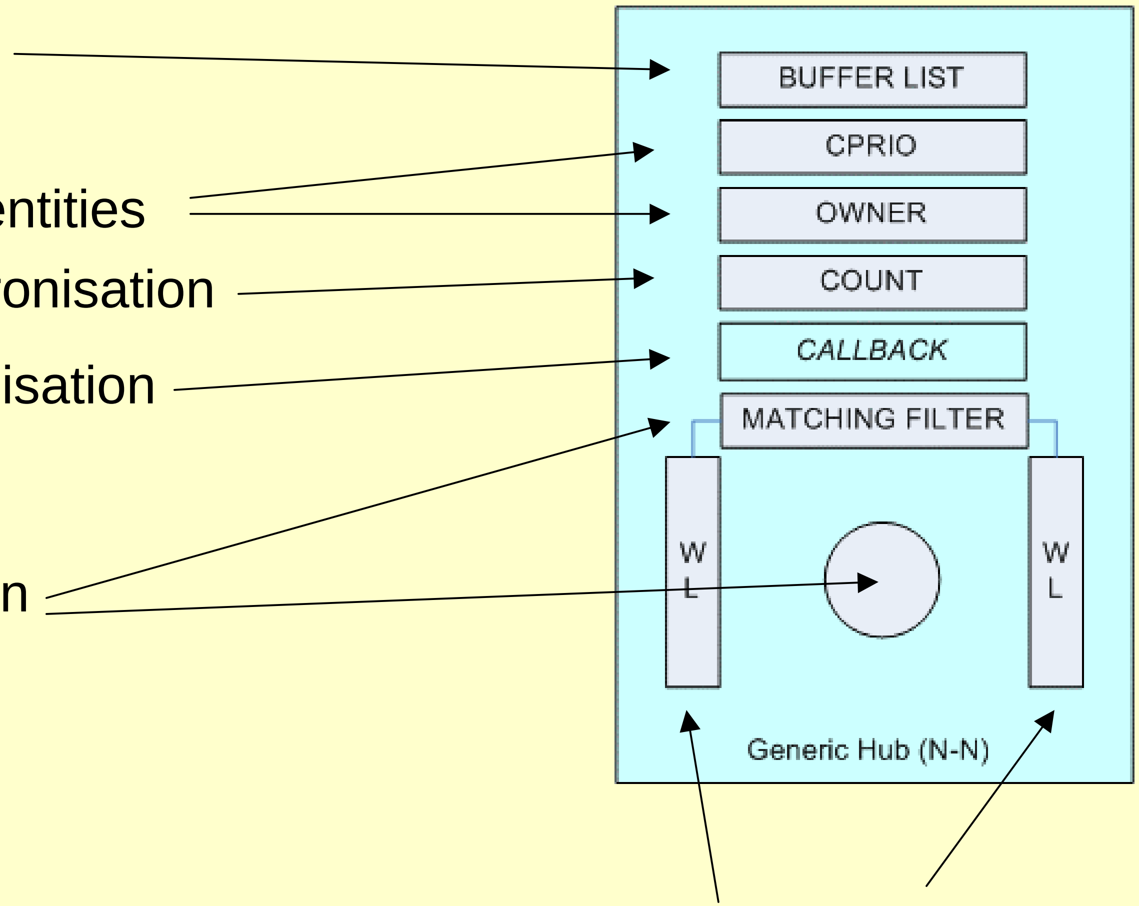
When buffering

With resource entities

Pending synchronisation

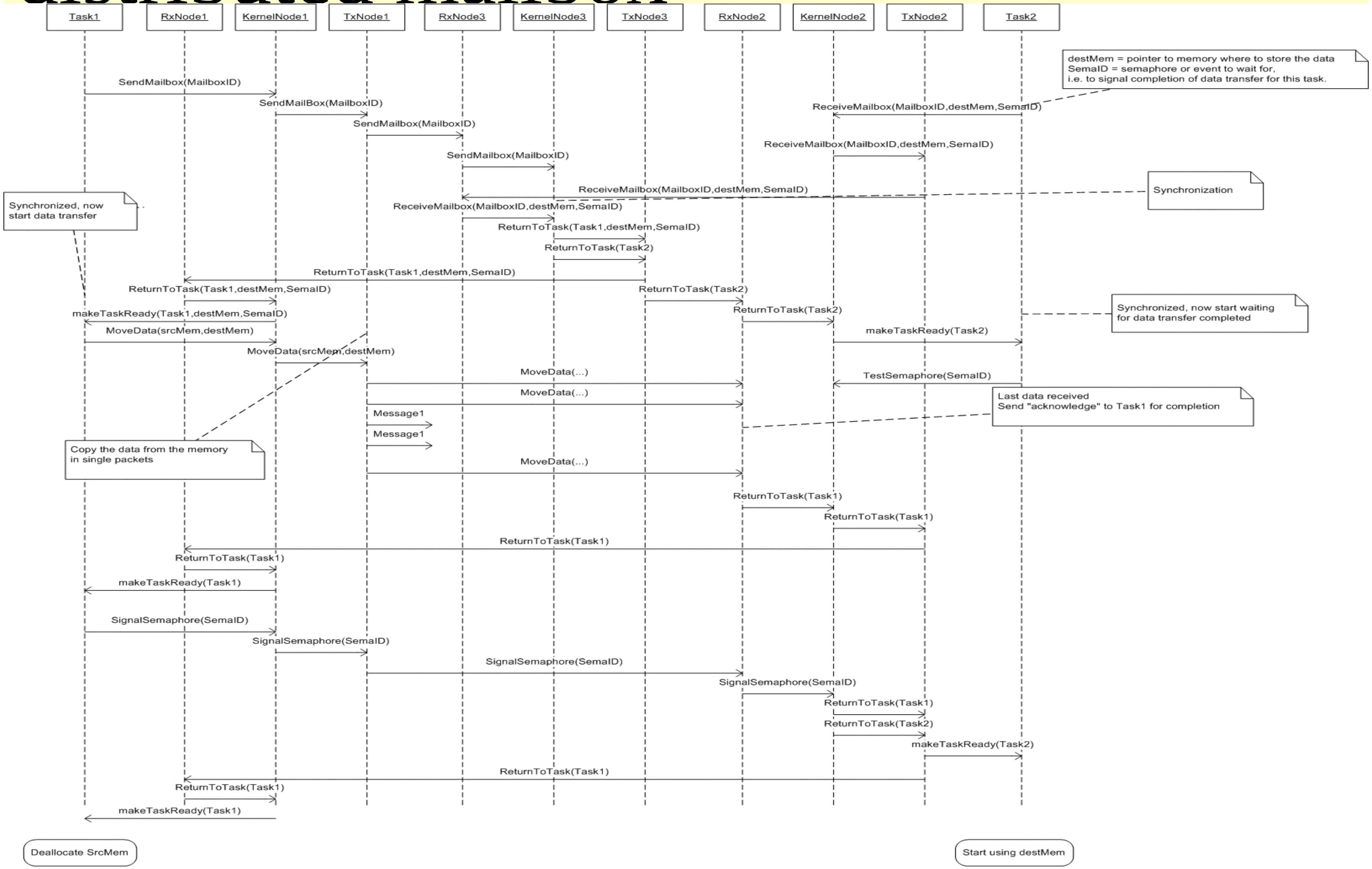
Upon Synchronisation

Synchronisation



Waiting lists “senders/receivers”

Example of Interaction diagram: distributed mailbox



Clean architecture gives small code

OpenComRTOS L1 code size figures (MLX16)				
	MP FULL		**SP SMALL**	
	L0	**L1**	**L0**	**L1**
L0 Port	162		132	
L1 Hub shared		574		400
L1 Port		4		4
L1 Event		68		70
L1 Semaphore		54		54
L1 Resource		104		104
L1 FIFO		232		232
L1 Resource List		184		184
Total L1 services		1220		1048
Grand Total	**3150**	**4532**	**996**	**2104**

<u>Smallest application</u>: 1048 bytes program code and 198 bytes RAM (data)
(SP, 2 tasks with 2 Ports sending/receiving Packets in a loop, ANSI-C)
Number of instructions : 605 instructions for one loop (= 2 x context switches, 2 x L0_SendPacket_W, 2 x L0_ReceivePacket_W)

Results (ctd)

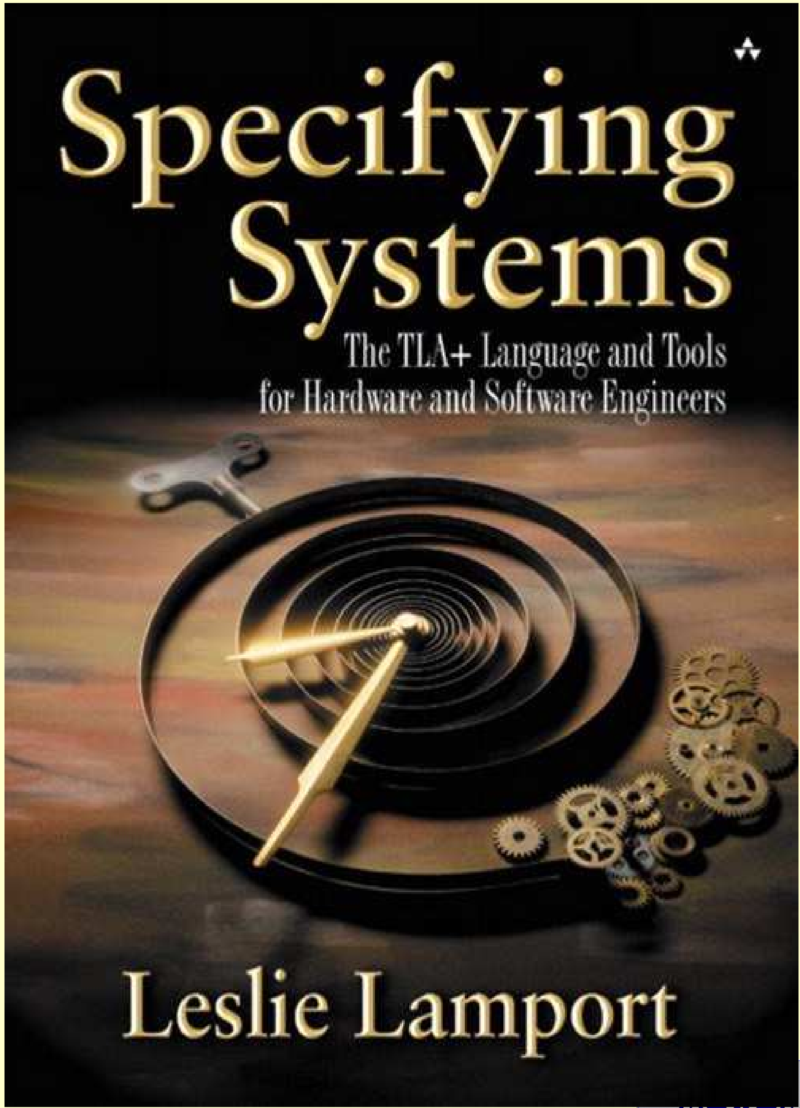
- Break-through results in well-known domain
  - 100’s of RTOS with such support
  - 15 years of experience, 3 generations of RTOS design
  - Typically CPU dependent, use of assembler and async operation

- Small, scalable, distributed and maintainable code
  - SP(L0): < 1000 machine instructions
  - MP(L1): < 2000 - 5000 machine instructions
  - Needs a few 100 bytes of data RAM
  - Fully in ANSI-C, MISRA-C compliant
  - Runs on MelexCM (16 bit) and Windows
  - Scheduling algorithm can be improved to reduce worst-case rescheduling latency and blocking time
  - All RTOS Entities are variations of a generic « hub » object
    - => less but faster code: 5 KBytes vs. 50 KBytes before

Formal TLA models of OpenComRTOS entities

TLA

- **TLA** (the **Temporal Logic of Actions**) is a logic for specifying and reasoning about concurrent and reactive systems.

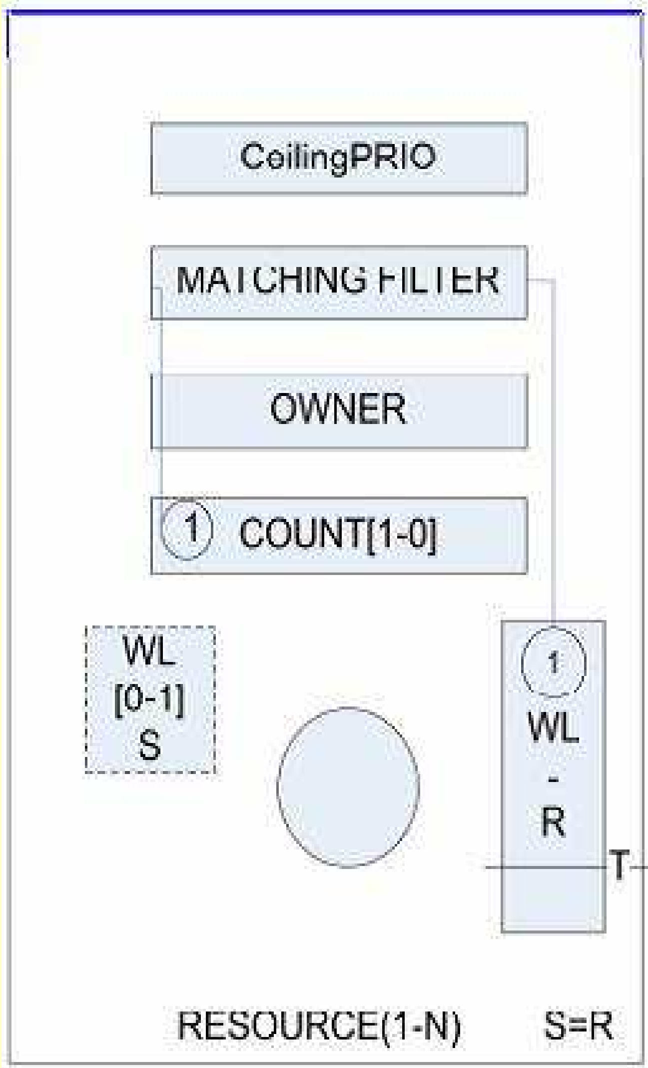


OPEN LICENSE SOCIETY

TLA modelling results

We modeled entities of OpenComRTOS:

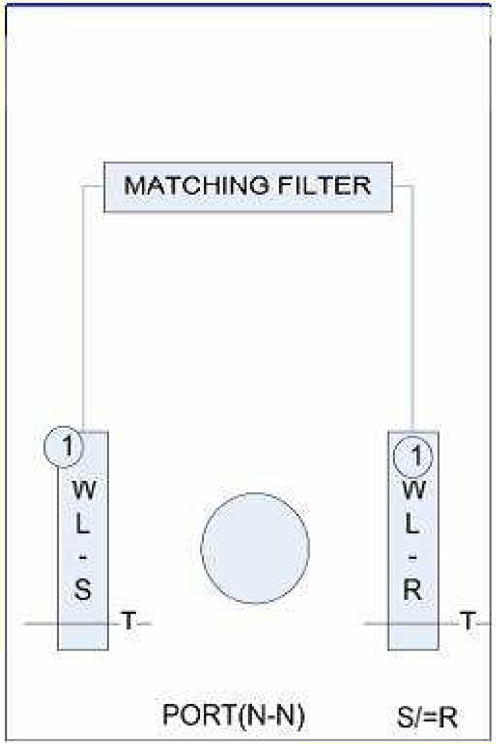
- Port
- Event
- Semaphore
- Resource
- Packet Pool
- Memory Pool
- FIFO
- Mailbox



Port

Verified Properties:

- There are never more Tasks on the ready list than there are Tasks in the system
- There are never more Tasks in the Port's waiting list then there are Tasks in the system
- All Tasks waiting on an Port, either waiting to send a Packet, either waiting to receive a Packet are of the same type in each waiting list

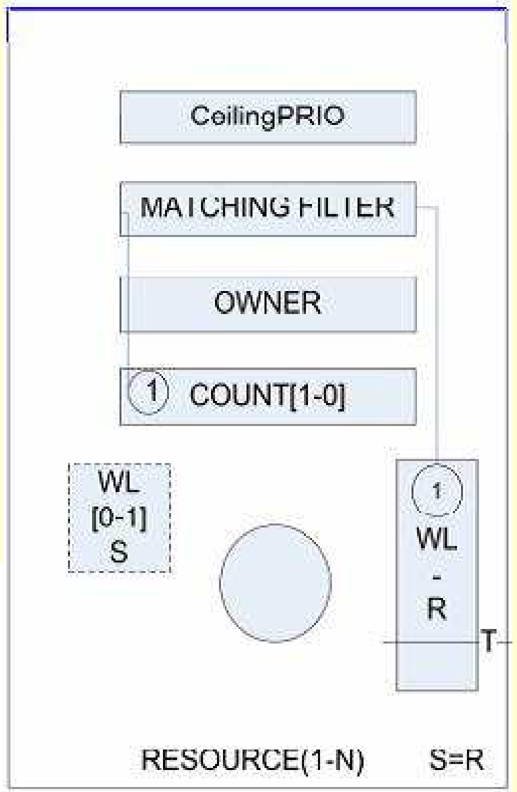


$$
\begin{aligned}
&TypeInvariant \triangleq \\
&\wedge\ Cardinality(ReadyList) \leq Cardinality(TaskId) \\
&\wedge\ \forall\, p \in PortId : \\
&Len(PortWL[p]) \leq Cardinality(TaskId) \\
&\wedge\ \forall\, p \in PortId : \\
&\forall\, i, j \in 1 \,..\, Len(PortWL[p]) : \\
&PreallocatedPacket[PortWL[p][i]].type = PreallocatedPacket[PortWL[p][j]].type
\end{aligned}
$$

Resource

Verified Properties:

- No Task waiting for an Resource can be ready
- Any Task that is ready cannot be in a waiting condition
- When the resource is free, then no Task will be waiting for it



$$
\begin{aligned}
&\land \forall p \in ResourceId : \\
&\forall i \in 1 \,..\, Len(ResourceWL[p]) : \\
&ResourceWL[p][i] \notin ReadyList \\
&\land \forall t \in TaskId : \\
&(t \in ReadyList) \Rightarrow (\forall i \in ResourceId : \\
&\forall j \in 1 \,..\, Len(ResourceWL[i]) : \\
&PreallocatedPacket[ResourceWL[i][j]].RequestingTaskID \neq t) \lor \\
&(\forall i \in 1 \,..\, Len(KernelPortWL) : \\
&PreallocatedPacket[KernelPortWL[i]].RequestingTaskID \neq t) \\
&\land \forall p \in ResourceId : \\
&\land isResourceAvailable(p) \Rightarrow List\_isEmpty(ResourceWL[p])
\end{aligned}
$$

Packet Pool

Checking Properties:

- Packets from the Packet Pool can only be allocated once
- If a Packet has been allocated from the Pool, then it must be in a Packet List of a Task and vice versa, if a Packet is not in any Task Packet List then it must be in the Packet Pool.
- If we have any packet in the Packet Pool, then no Task will be waiting for a Packet

$$
\begin{aligned}
&\wedge \forall\, i, j \in TaskId : \\
&(i \neq j) \Rightarrow (task[i].packetlist \cap task[j].packetlist = \{\}) \\
&\wedge \forall\, t \in TaskId : \forall\, dp \in DynamicalPacketId : \\
&\vee (dp \in task[t].packetlist) \Rightarrow (dp \notin PacketPoolPacketList) \\
&\vee (dp \in PacketPoolPacketList) \Rightarrow (dp \notin task[t].packetlist) \\
&\wedge isPacketAvailable \Rightarrow Len(PacketPoolWL) = 0
\end{aligned}
$$

RTOS Metamodel

- Based on Interacting Entities Paradigm
- Application can be constructed from various entities (*kernel entities*) and interactions between them (*kernel services*).
- The Metamodel allows extensions to different sets of kernel entities and services of other RTOSes.
- Expression of the Metamodel in XML format

OpenComRTOS Entities’ Metamodel = RTOS hub instances

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Metamodel>
    <Entity Name="Task" SvgPath="Task.svg">
        <Attribute Name="Name" Type="String"/>
        <Attribute Name="Priority" Type="Integer" MinValue="1" MaxValue="255"/>
        <Attribute Name="Arguments" Type="String" DefaultValue="NULL"/>
        <Attribute Name="Status" Type="Enum" Values="L0_INACTIVE:L0_STARTED" DefaultValue="L0_STARTED"/>
        <Attribute Name="Node" Type="Node"/>
        <Attribute Name="StackSize" Type="Integer" DefaultValue="170"/>
        <Function  Name="EntryPoint"/>
    </Entity>
    <Entity Name="Port" SvgPath="Port.svg">
        <Attribute Name="Name" Type="String"/>
        <Attribute Name="Node" Type="Node"/>
    </Entity>
    <Entity Name="Event" SvgPath="Event.svg">
        <Attribute Name="Name" Type="String"/>
        <Attribute Name="Node" Type="Node"/>
    </Entity>
    <Entity Name= "Semaphore" SvgPath="Semaphore.svg">
        <Attribute Name="Name" Type="String"/>
        <Attribute Name="Node" Type="Node"/>
    </Entity>
```

```xml
    <Entity Name="Hub" SvgPath="Hub.svg">
        <Attribute Name="Name" Type="String"/>
        <Attribute Name="Node" Type="Node"/>
        <Attribute Name="Type" Type="Enum" Values="L1_PORT:L1_EVENT:L1_SEMAPHORE:L1_RESOURCE:L1_FIFO:L1_P
        <Function Name="UpdateFunction"/>
        <Function Name="SyncFunction"/>
    </Entity>
```

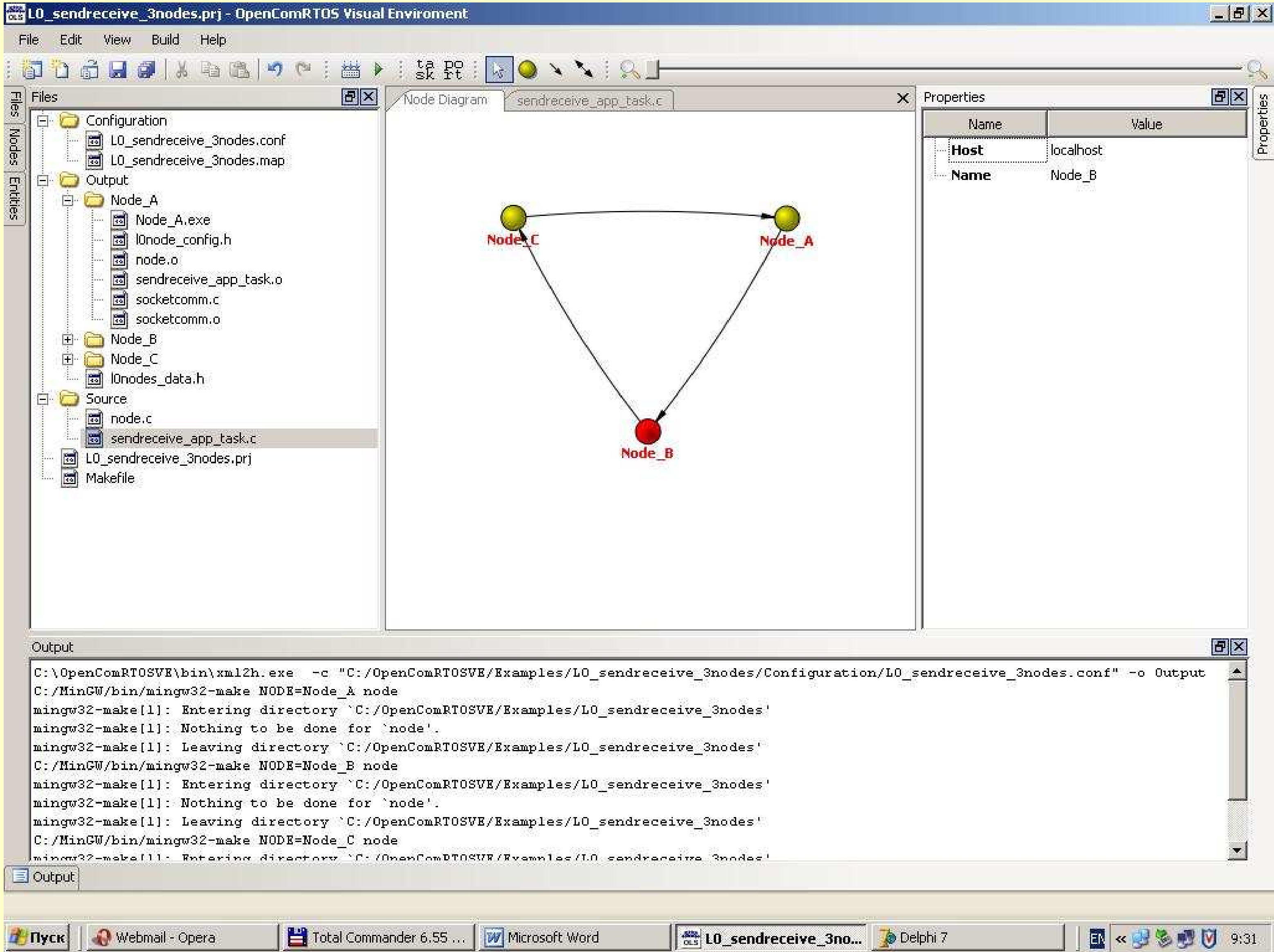
OpenComRTOS Interactions' Metamodel = kernel services

```
<Interaction Name="L1_StartTask_W" Subject="Task" Object="Task" />
<Interaction Name="L1_StopTask_W" Subject="Task" Object="Task" />
<Interaction Name="L1_SuspendTask_W" Subject="Task" Object="Task" />
<Interaction Name="L1_ResumeTask_W" Subject="Task" Object="Task" />
<Interaction Name="L1_SendPacket_W" Subject="Task" Object="Port" />
<Interaction Name="L1_ReceivePacket_W" Subject="Task" Object="Port" />
<Interaction Name="L1_SendPacket_NW" Subject="Task" Object="Port" />
<Interaction Name="L1_ReceivePacket_NW" Subject="Task" Object="Port" />
<Interaction Name="L1_SendPacket_WT" Subject="Task" Object="Port" />
<Interaction Name="L1_ReceivePacket_WT" Subject="Task" Object="Port" />
<Interaction Name="L1_SendPacket_A" Subject="Task" Object="Port" />
<Interaction Name="L1_ReceivePacket_A" Subject="Task" Object="Port" />
<Interaction Name="L1_AllocatePacket" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_DeallocatePacket_W" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_AllocatePacket_W" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_WaitForPacket" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_WaitForPacket_W" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_AllocatePacket_NW" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_WaitForPacket_NW" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_AllocatePacket_WT" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_WaitForPacket_WT" Subject="Task" Object="PacketPool" />
<Interaction Name="L1_SendPacket_W" Subject="Task" Object="Port" />
<Interaction Name="L1_ReceivePacket_W" Subject="Task" Object="Port" />
<Interaction Name="L1_RaiseEvent_W" Subject="Task" Object="Event" />
<Interaction Name="L1_TestEvent_W" Subject="Task" Object="Event" />
<Interaction Name="L1_SignalSemaphore_W" Subject="Task" Object="Semaphore" />
<Interaction Name="L1_TestSemaphore_W" Subject="Task" Object="Semaphore" />
<Interaction Name="L1_LockResource_W" Subject="Task" Object="Resource" />
<Interaction Name="L1_UnlockResource_W" Subject="Task" Object="Resource" />
<Interaction Name="L1_EnqueueFifo_W" Subject="Task" Object="Fifo" />
<Interaction Name="L1_DequeueFifo_W" Subject="Task" Object="Fifo" />
<Interaction Name="L1_SendPacket_NW" Subject="Task" Object="Port" />
```

Stages of application development in OpenComRTOSVE

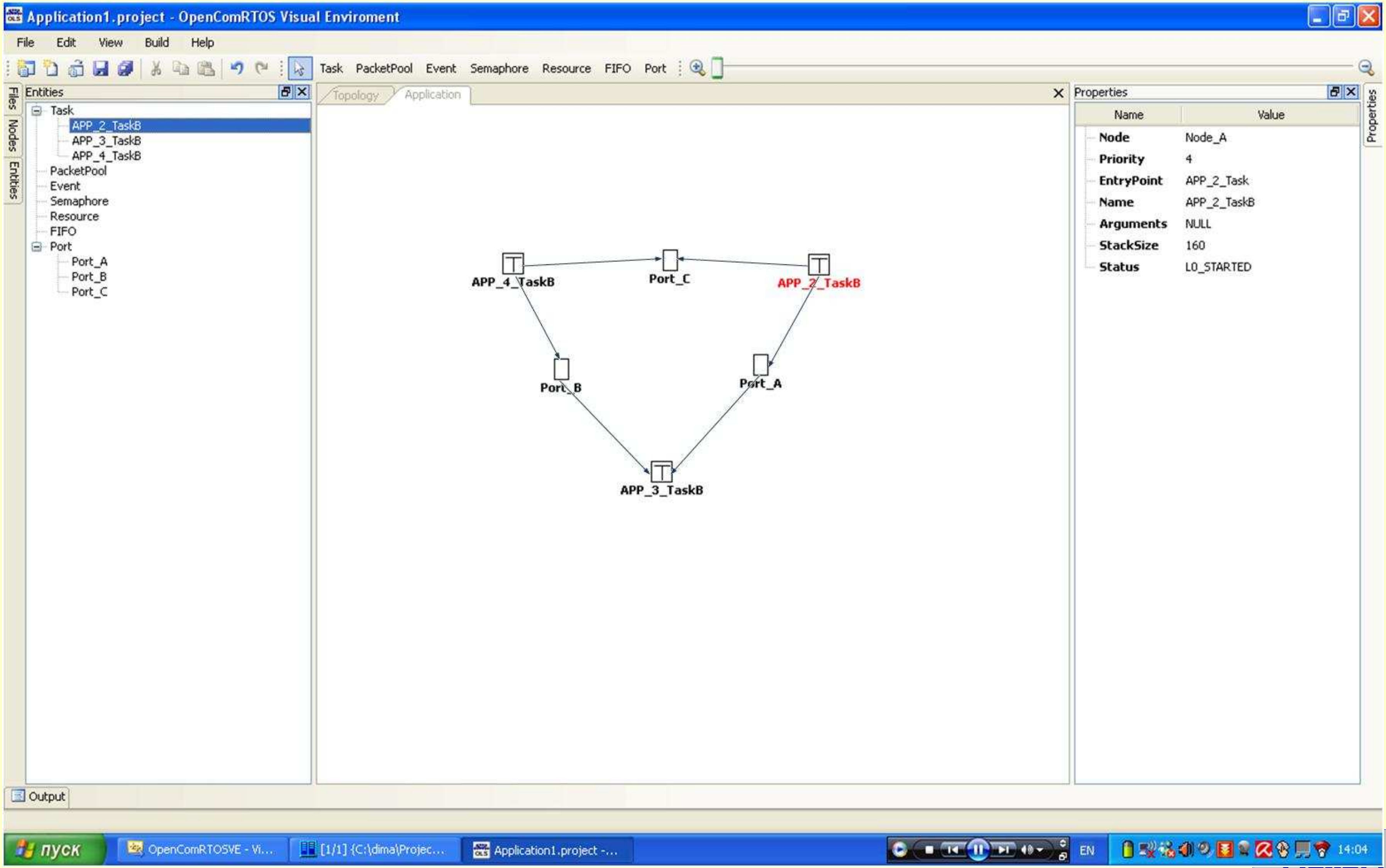
1. Nodes (processors) topology definition
2. RTOS application structure definition
3. Tasks coding
4. Compiling and running
5. Tracing

Nodes topology definition

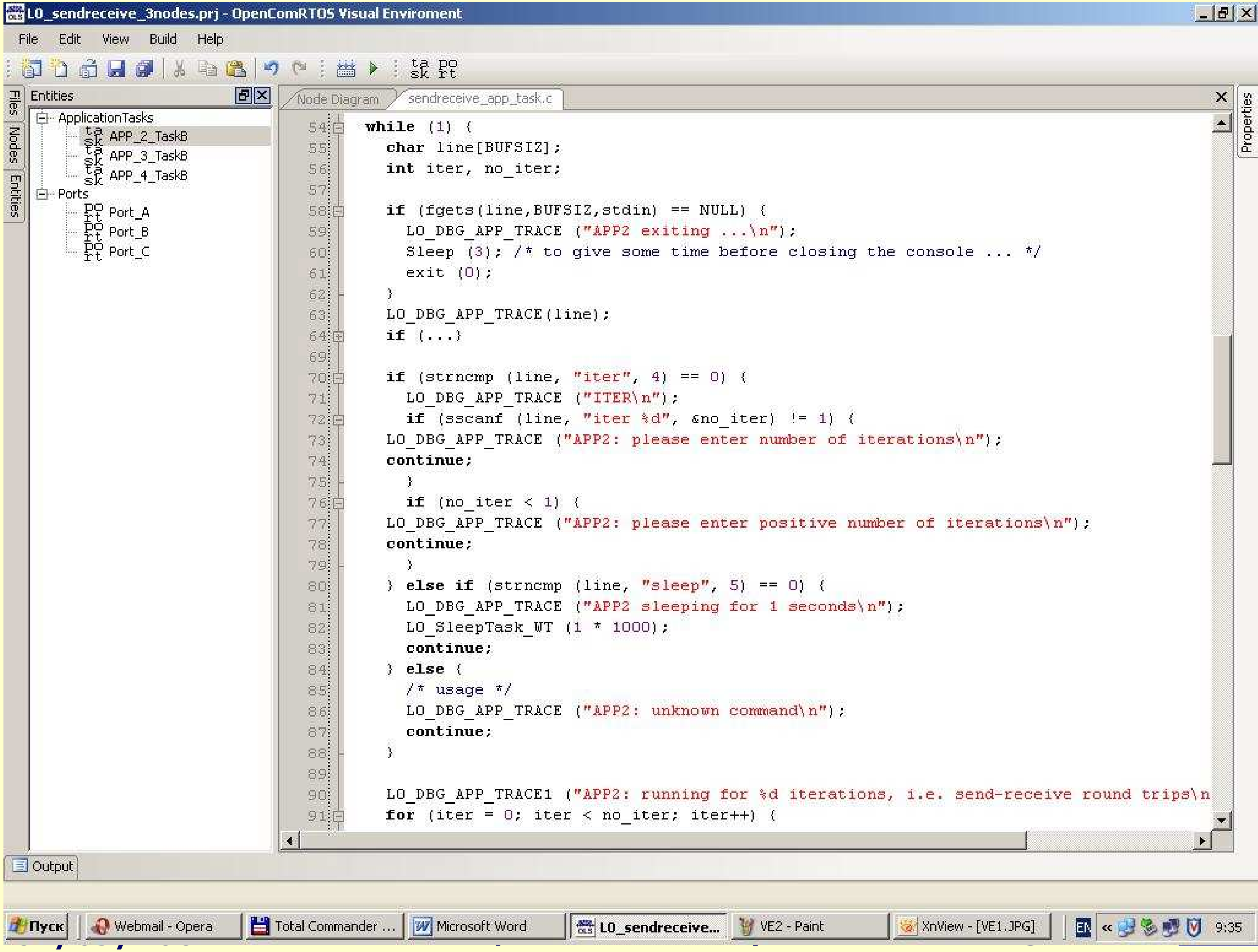


OPEN LICENSE SOCIETY

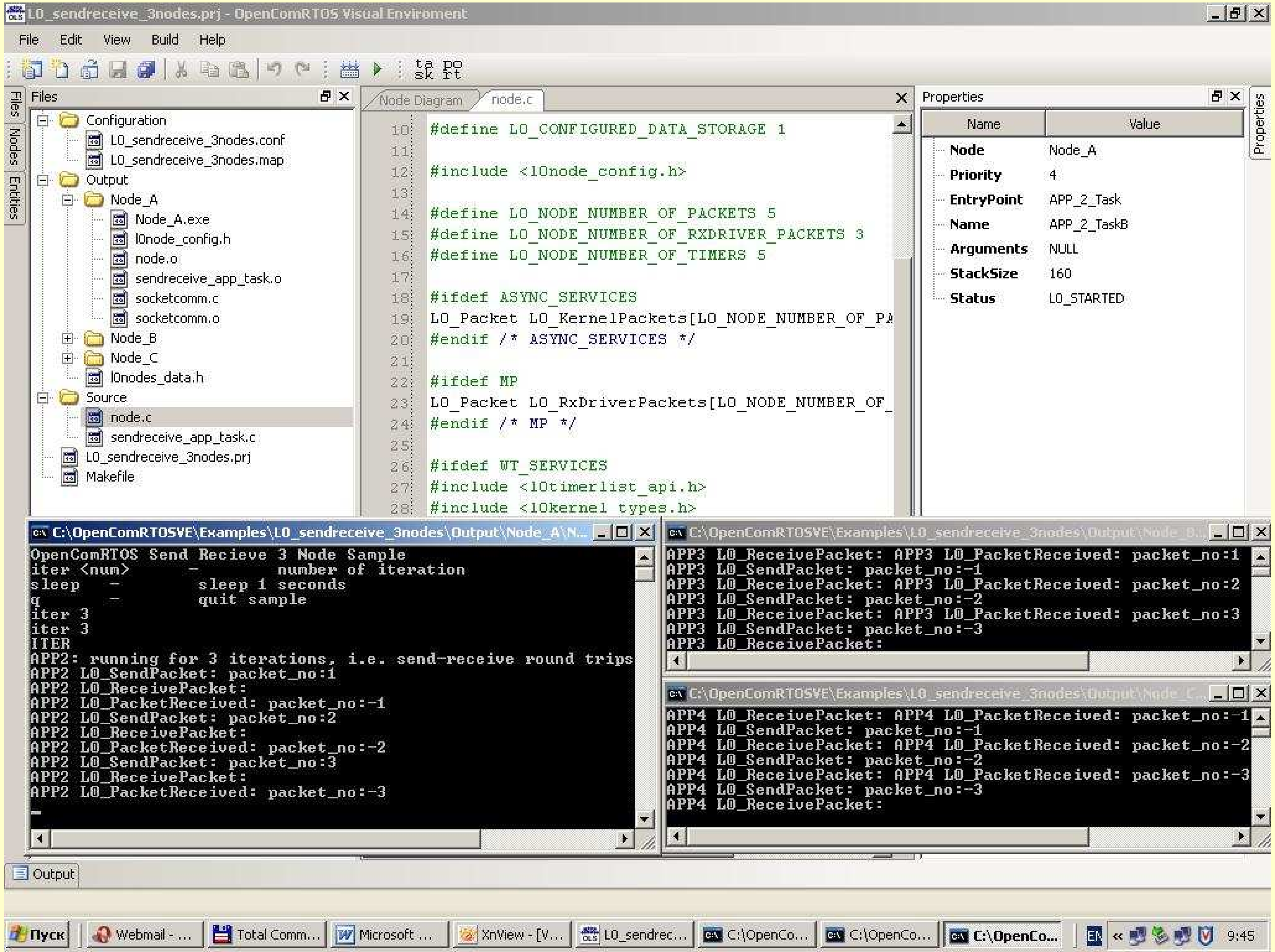
Application structure definition



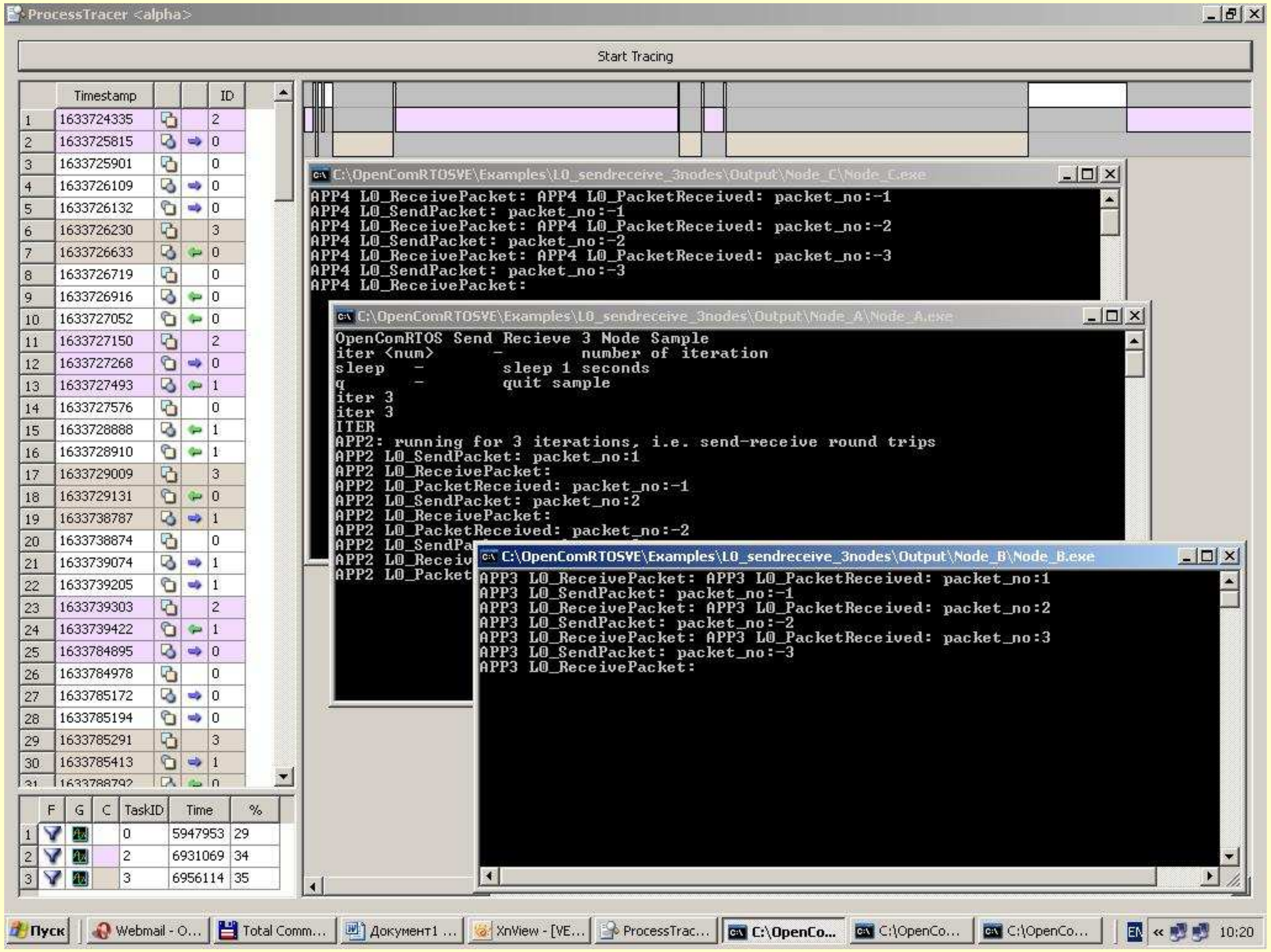
Task source coding



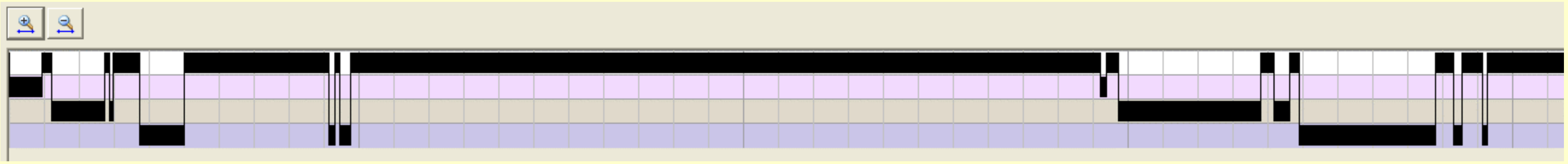
Compiling and running

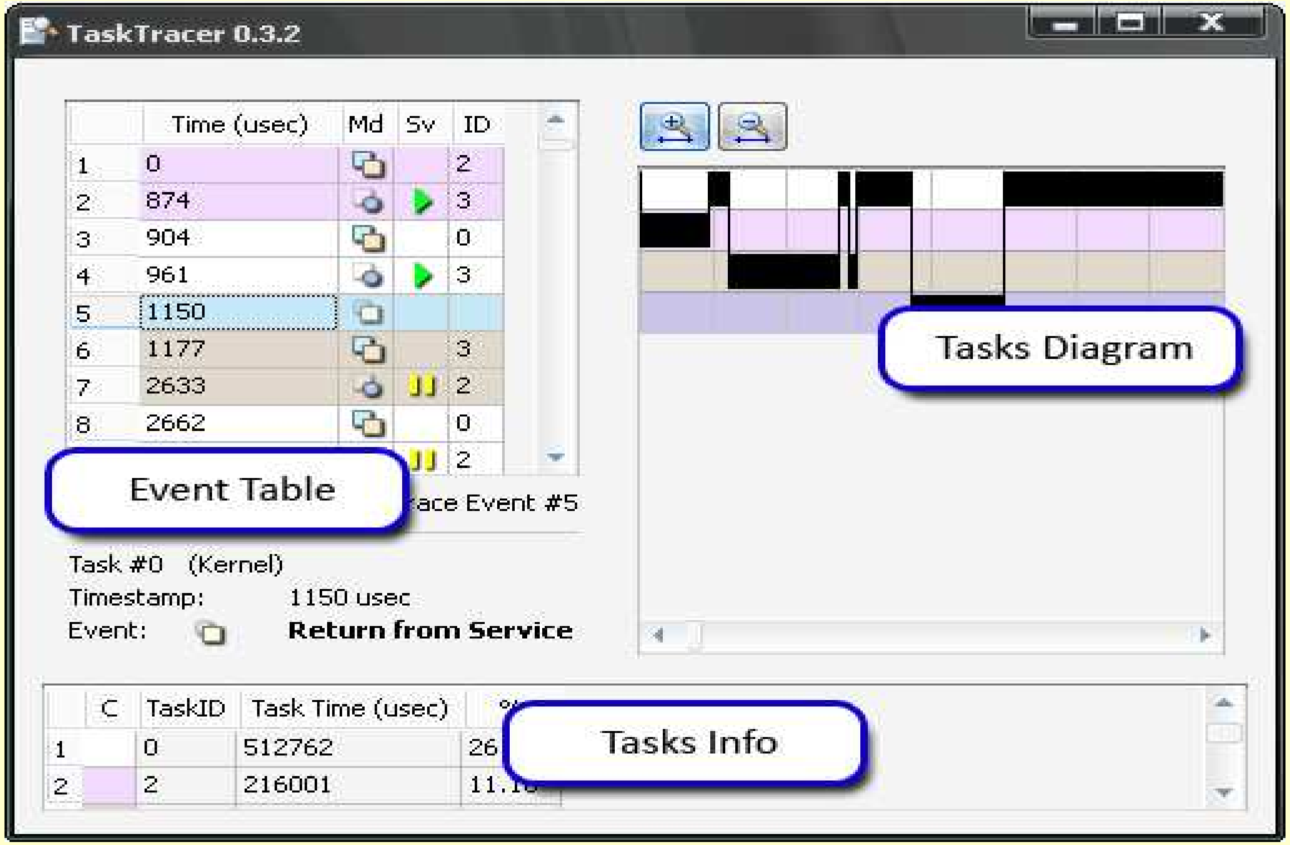


Tracing


ProcessTracer <alpha>
Start Tracing
Timestamp ID
1 1633724335 2
2 1633725815 0
3 1633725901 0
4 1633726109 0
5 1633726132 0
6 1633726230 3
7 1633726633 0
8 1633726719 0
9 1633726916 0
10 1633727052 0
11 1633727150 2
12 1633727268 0
13 1633727493 1
14 1633727576 0
15 1633728888 1
16 1633728910 1
17 1633729009 3
18 1633729131 0
19 1633738787 1
20 1633738874 0
21 1633739074 1
22 1633739205 1
23 1633739303 2
24 1633739422 1
25 1633784895 0
26 1633784978 0
27 1633785172 0
28 1633785194 0
29 1633785291 3
30 1633785413 1
F G C TaskID Time %
1 0 5947953 29
2 2 6931069 34
3 3 6956114 35
C:\OpenComRTOSVE\Examples\L0_sendreceive_3nodes\Output\Node_C\Node_C.exe
APP4 L0_ReceivePacket: APP4 L0_PacketReceived: packet_no:-1
APP4 L0_SendPacket: packet_no:-1
APP4 L0_ReceivePacket: APP4 L0_PacketReceived: packet_no:-2
APP4 L0_SendPacket: packet_no:-2
APP4 L0_ReceivePacket: APP4 L0_PacketReceived: packet_no:-3
APP4 L0_SendPacket: packet_no:-3
APP4 L0_ReceivePacket:
C:\OpenComRTOSVE\Examples\L0_sendreceive_3nodes\Output\Node_A\Node_A.exe
OpenComRTOS Send Recieve 3 Node Sample
iter <num> - number of iteration
sleep - sleep 1 seconds
q - quit sample
iter 3
iter 3
ITER
APP2: running for 3 iterations, i.e. send-receive round trips
APP2 L0_SendPacket: packet_no:1
APP2 L0_ReceivePacket:
APP2 L0_PacketReceived: packet_no:-1
APP2 L0_SendPacket: packet_no:2
APP2 L0_ReceivePacket:
APP2 L0_PacketReceived: packet_no:-2
C:\OpenComRTOSVE\Examples\L0_sendreceive_3nodes\Output\Node_B\Node_B.exe
APP3 L0_ReceivePacket: APP3 L0_PacketReceived: packet_no:1
APP3 L0_SendPacket: packet_no:-1
APP3 L0_ReceivePacket: APP3 L0_PacketReceived: packet_no:2
APP3 L0_SendPacket: packet_no:-2
APP3 L0_ReceivePacket: APP3 L0_PacketReceived: packet_no:3
APP3 L0_SendPacket: packet_no:-3
APP3 L0_ReceivePacket:
Пуск Webmail - O... Total Comm... Документ1 ... XnView - [VE... ProcessTrac... C:\OpenCo... C:\OpenCo... C:\OpenCo... 10:20

OpenComRTOS Event Tracer





Conclusion

- OpenComRTOS breaks new grounds for distributed real-time processing:
  - Developed using formal methods
  - Based on packet switching
  - Ultra-small, still same functionality
  - Scalable and extensible by use of meta-model
  - From multicore to widely distributed systems
- More info:
  - www.OpenLicenseSociety.org