



Open License Society

www.OpenLicenseSociety.org

Unifying and systematic system development methodologies
with trustworthy embedded components

Eric.Verhulst@OpenLicenseSociety.org



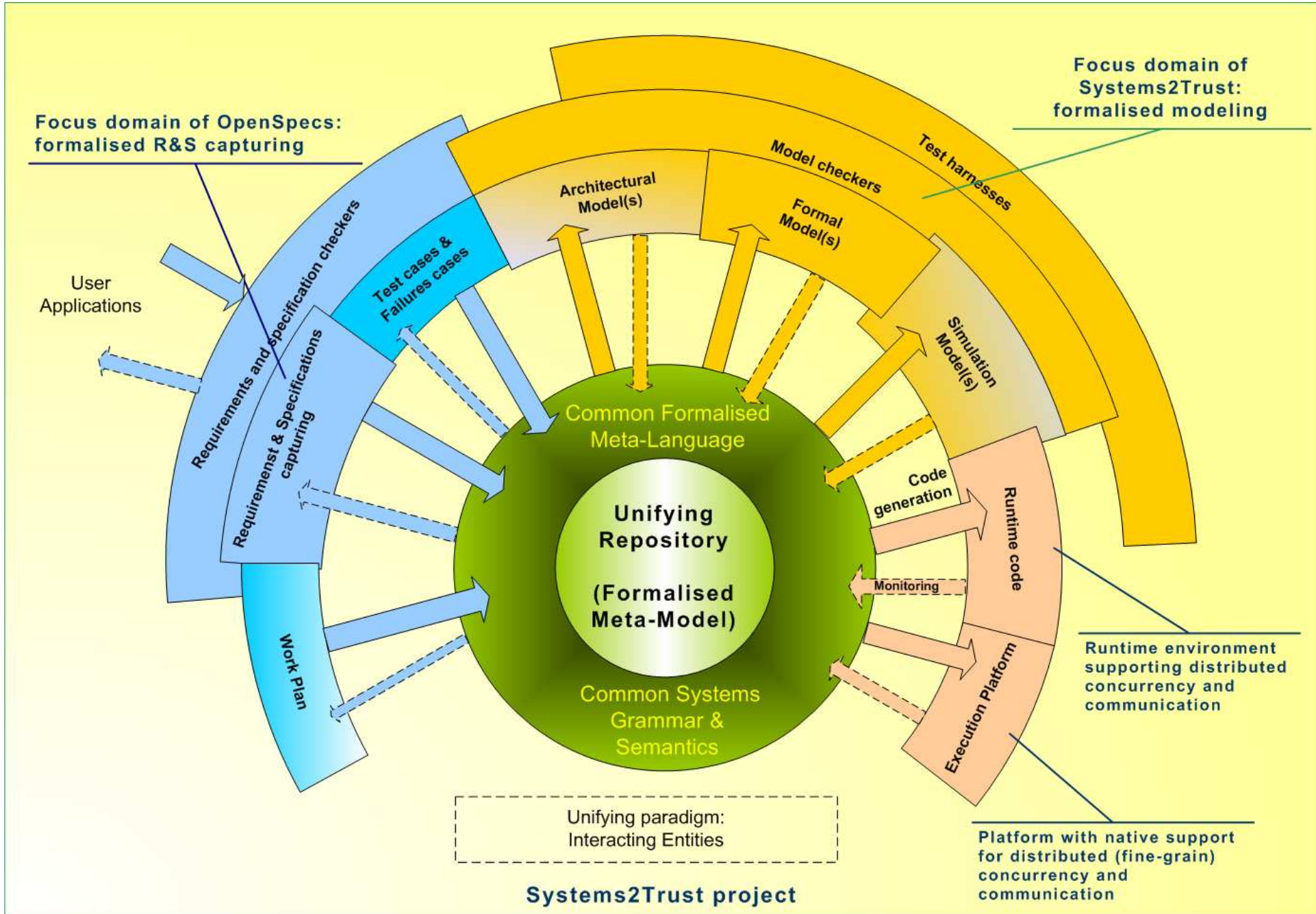
OpenComRTOS: An Ultra-Small Network Centric Embedded RTOS Designed Using Formal Modeling

Eric Verhulst and Gjalt de Jong
Open License Society
Leuven, Belgium

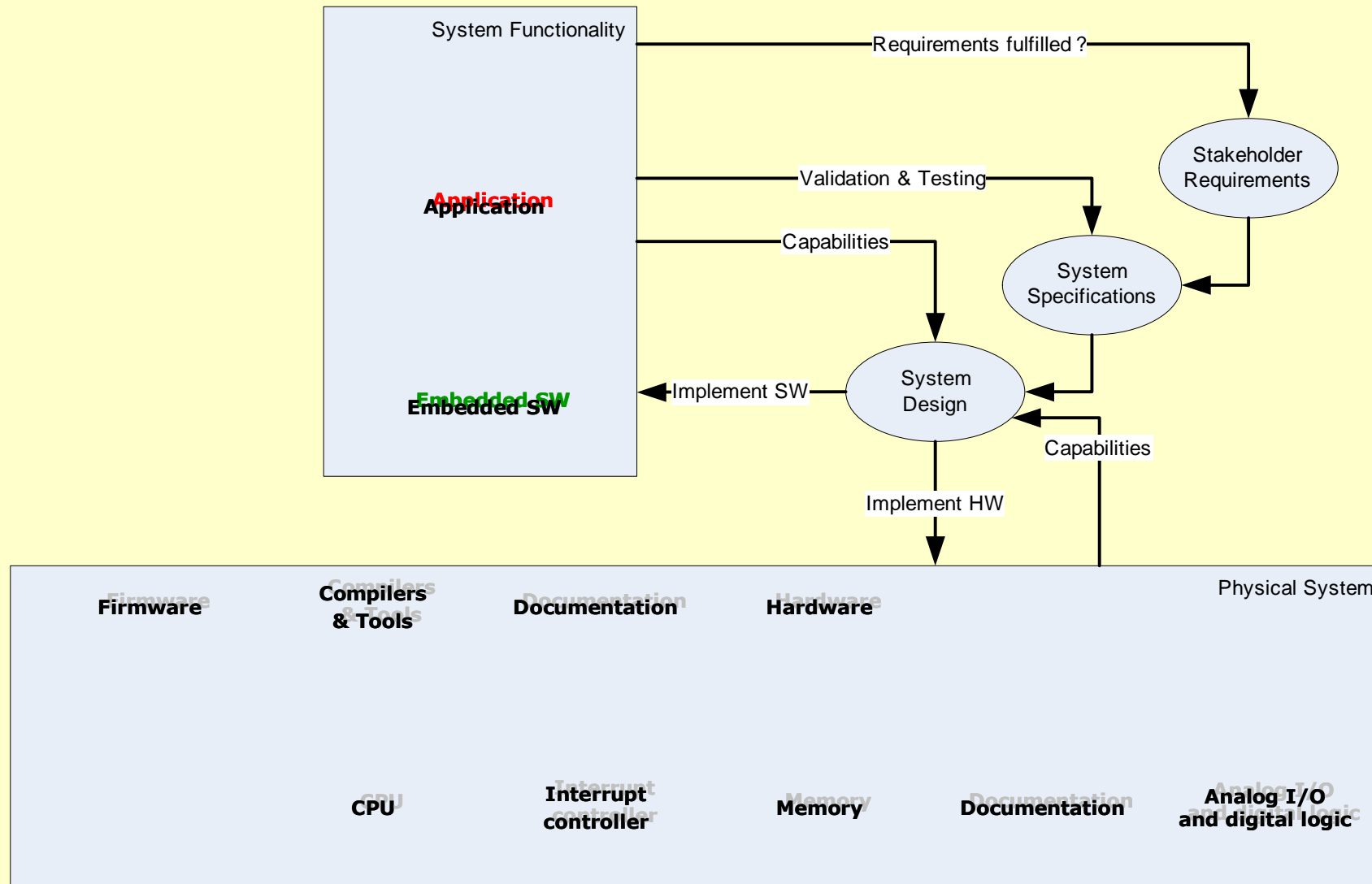
{eric.verhulst,gjalt.dejong}@OpenLicenseSociety.org

Who is Open License Society?

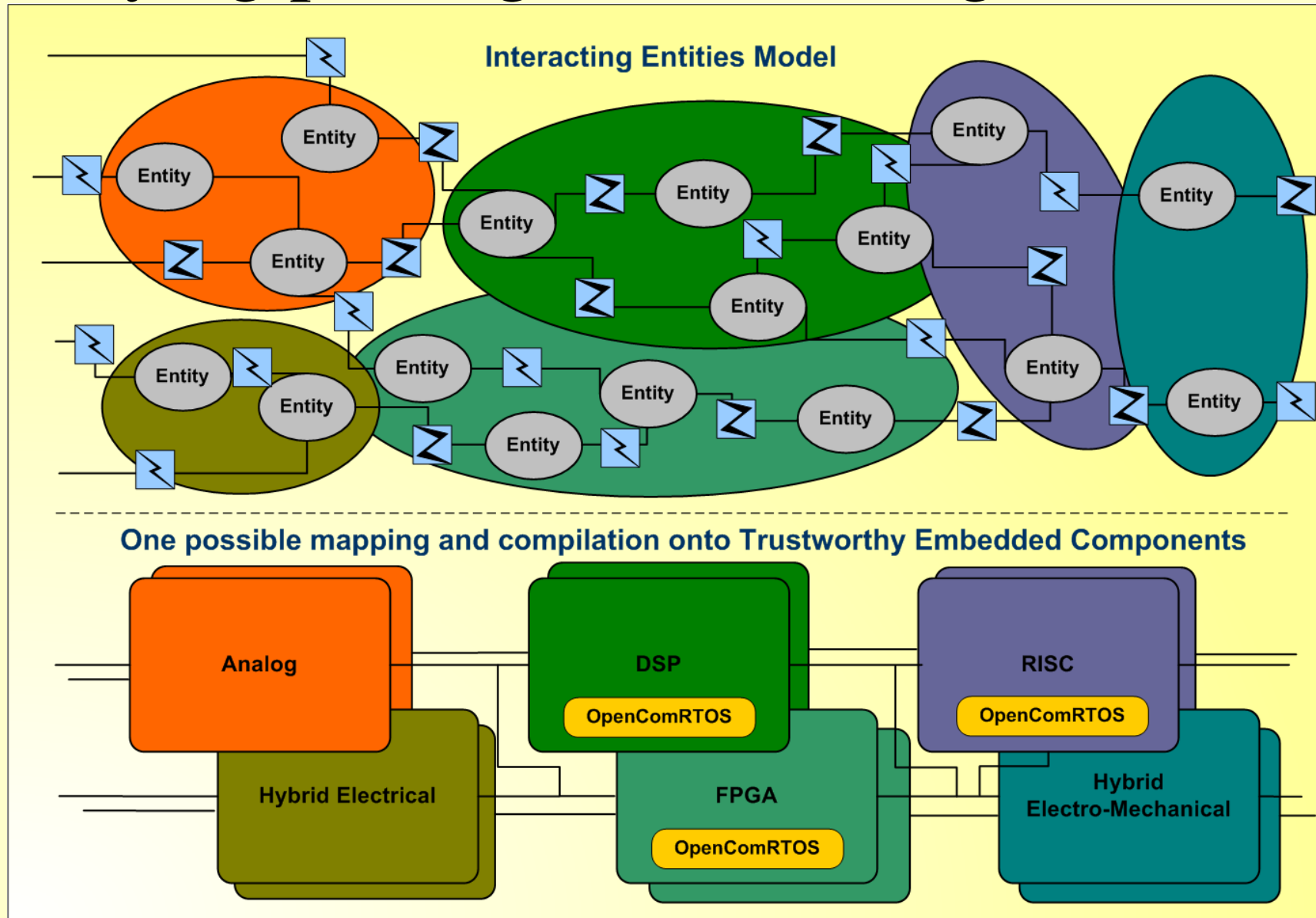
- Privately funded R&D institute
 - Leuven (BE), Berdyansk (UA)
 - Industrial sponsors
 - IWT project funding for OpenComRTOS
- Why: 70 % of all SE projects do not deliver
- Objectives
 - Systematic & Unified Systems Engineering Methodology
 - 'Interacting Entities' paradigm at all levels:
 - OpenComRTOS 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, ...



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 distributed 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 grammar

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**

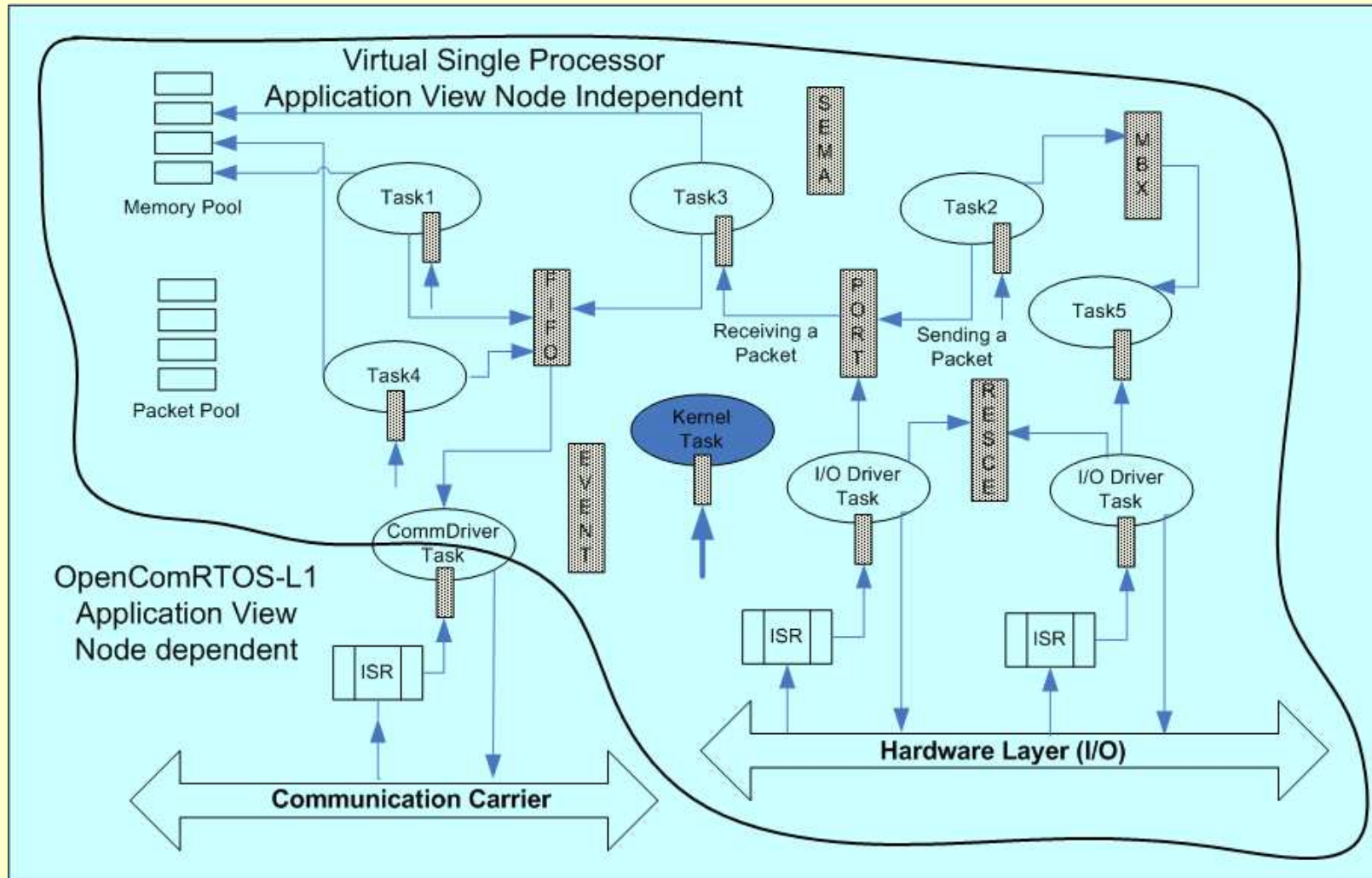
IdleTask **OR**

RxTask **OR**

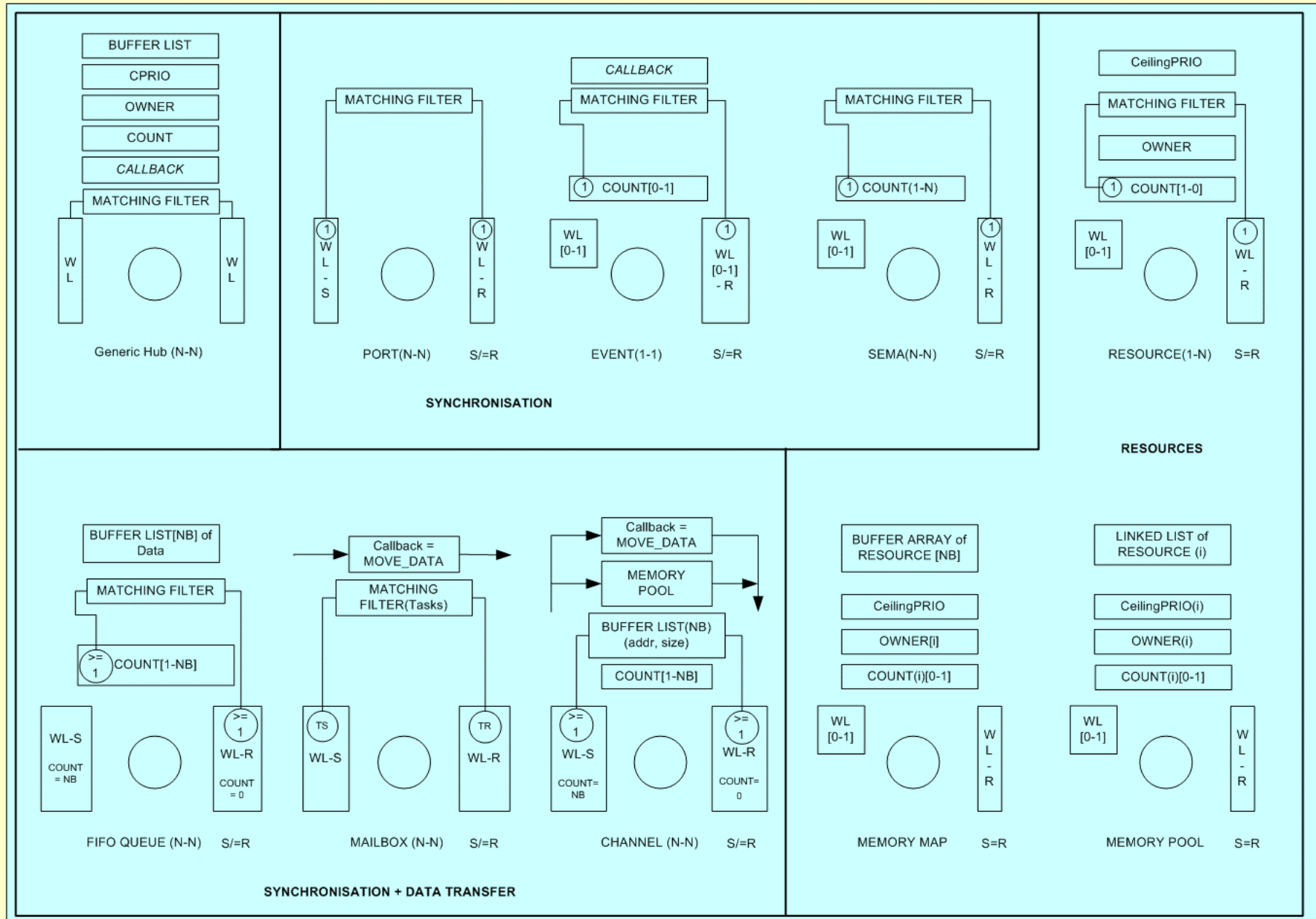
TxTask

Etc.

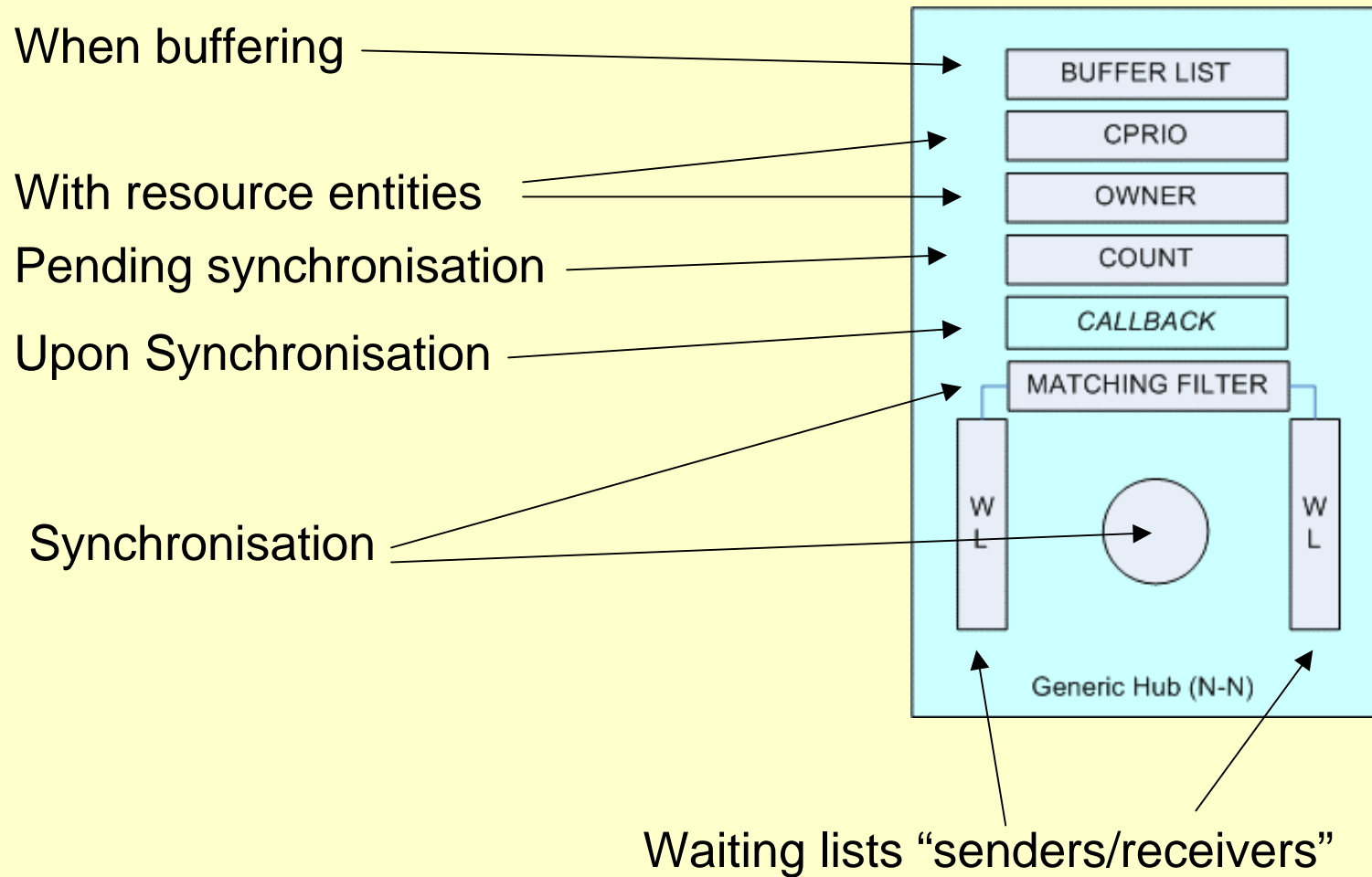
L1 application view



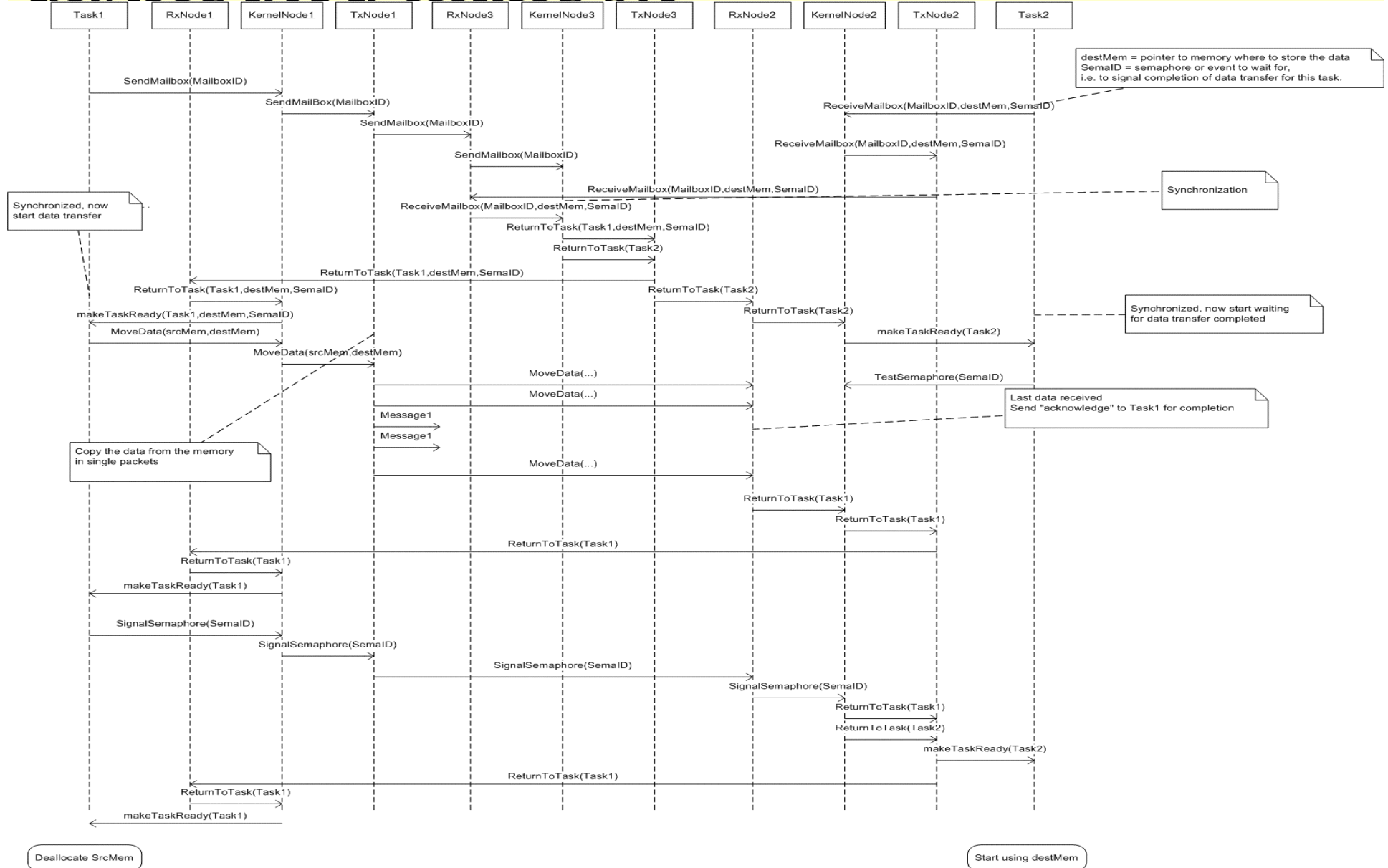
All RTOS Entities: variation on a theme



Generic hub



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

Smallest application: 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)

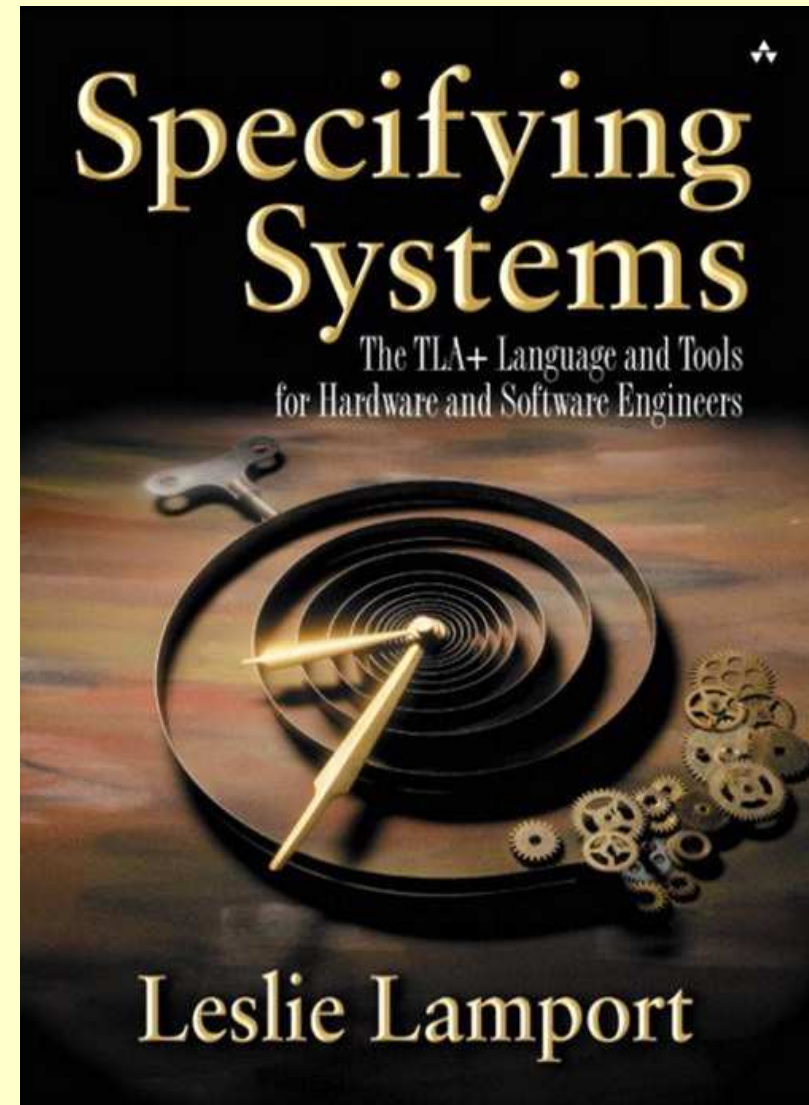
Results (ctd)

- 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

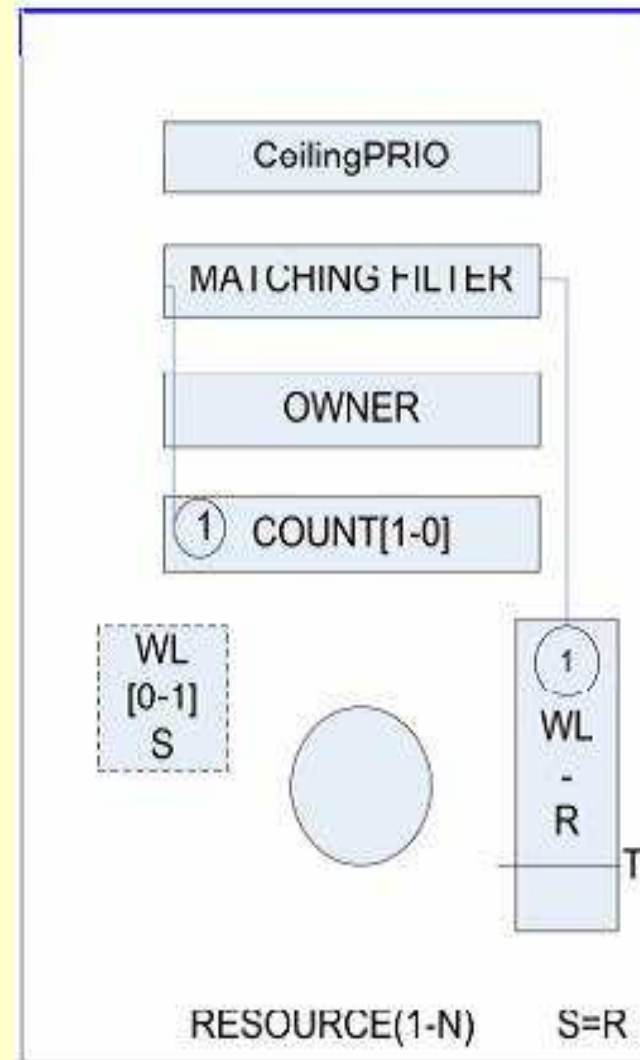
- **TLA (the Temporal Logic of Actions)** is a logic for specifying and reasoning about concurrent and reactive systems.



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

TypeInvariant \triangleq

$\wedge \text{Cardinality}(\text{ReadyList}) \leq \text{Cardinality}(\text{TaskId})$

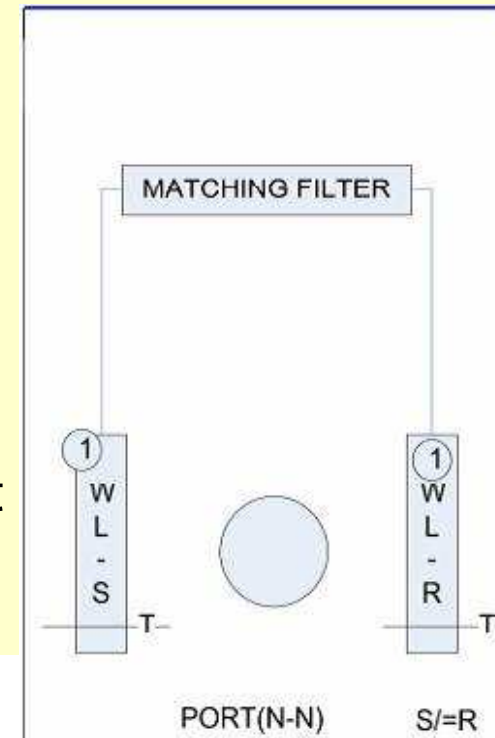
$\wedge \forall p \in \text{PortId} :$

$\text{Len}(\text{PortWL}[p]) \leq \text{Cardinality}(\text{TaskId})$

$\wedge \forall p \in \text{PortId} :$

$\forall i, j \in 1 .. \text{Len}(\text{PortWL}[p]) :$

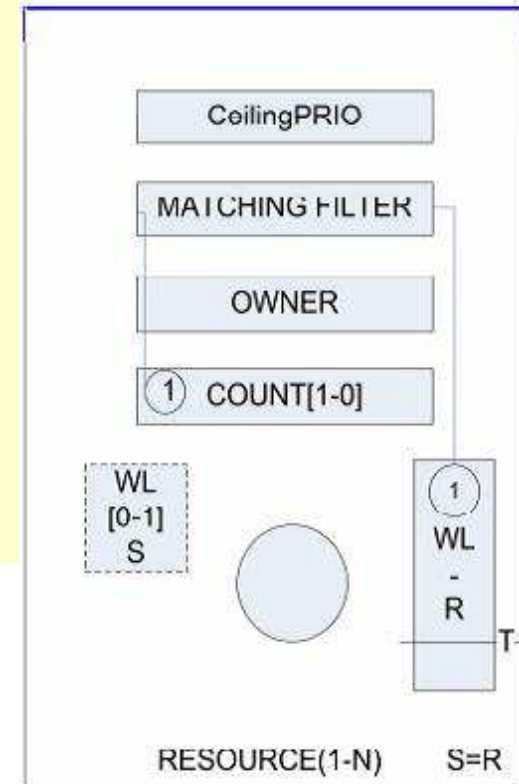
$\text{PreallocatedPacket}[\text{PortWL}[p][i]].\text{type} = \text{PreallocatedPacket}[\text{PortWL}[p][j]].\text{type}$



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

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


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

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

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 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="LO_INACTIVE:LO_STARTED" DefaultValue="LO_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>
  <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

The screenshot displays the OpenComRTOS Visual Environment interface. The main window shows a Node Diagram with three nodes: Node_A (yellow), Node_B (red), and Node_C (yellow). Node_A and Node_C are connected to Node_B, and Node_A and Node_C are also connected to each other. The Properties window on the right shows the following table:

Name	Value
Host	localhost
Name	Node_B

The Output window at the bottom shows the following text:

```
C:\OpenComRTOSVE\bin\xml2h.exe -c "C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes/Configuration/L0_sendreceive_3nodes.conf" -o Output
C:/MinGW/bin/mingw32-make NODE=Node_A node
mingw32-make[1]: Entering directory `C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes'
mingw32-make[1]: Nothing to be done for `node'.
mingw32-make[1]: Leaving directory `C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes'
C:/MinGW/bin/mingw32-make NODE=Node_B node
mingw32-make[1]: Entering directory `C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes'
mingw32-make[1]: Nothing to be done for `node'.
mingw32-make[1]: Leaving directory `C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes'
C:/MinGW/bin/mingw32-make NODE=Node_C node
mingw32-make[1]: Entering directory `C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes'
```

Application structure definition

The screenshot displays the OpenComRTOS Visual Environment interface for 'Application1.project'. The main workspace shows a topology diagram with the following components and connections:

- APP_4_TaskB** (Task) is connected to **Port_C** (Port).
- APP_2_TaskB** (Task) is connected to **Port_C** (Port).
- APP_2_TaskB** (Task) is connected to **Port_A** (Port).
- Port_B** (Port) is connected to **APP_3_TaskB** (Task).
- Port_A** (Port) is connected to **APP_3_TaskB** (Task).

The left sidebar shows the 'Entities' tree with the following structure:

- Task
 - APP_2_TaskB
 - APP_3_TaskB
 - APP_4_TaskB
- PacketPool
- Event
- Semaphore
- Resource
- FIFO
- Port
 - Port_A
 - Port_B
 - Port_C

The right sidebar shows the 'Properties' window for the selected entity, with the following details:

Name	Value
Node	Node_A
Priority	4
EntryPoint	APP_2_Task
Name	APP_2_TaskB
Arguments	NULL
StackSize	160
Status	LO_STARTED

The bottom status bar shows the system tray with the date and time: 01/09/2007 14:04.

Task source coding

The screenshot displays the OpenComRTOS Visual Environment. The main window shows the source code for the task `sendreceive_app_task.c`. The code is as follows:

```
54: while (1) {
55:     char line[BUFSIZ];
56:     int iter, no_iter;
57:
58:     if (fgets(line, BUFSIZ, stdin) == NULL) {
59:         LO_DBG_APP_TRACE ("APP2 exiting ...\n");
60:         Sleep (3); /* to give some time before closing the console ... */
61:         exit (0);
62:     }
63:     LO_DBG_APP_TRACE(line);
64:     if (...)
65:
66:
67:
68:
69:
70:     if (strncmp (line, "iter", 4) == 0) {
71:         LO_DBG_APP_TRACE ("ITER\n");
72:         if (sscanf (line, "iter %d", &no_iter) != 1) {
73:             LO_DBG_APP_TRACE ("APP2: please enter number of iterations\n");
74:             continue;
75:         }
76:         if (no_iter < 1) {
77:             LO_DBG_APP_TRACE ("APP2: please enter positive number of iterations\n");
78:             continue;
79:         }
80:     } else if (strncmp (line, "sleep", 5) == 0) {
81:         LO_DBG_APP_TRACE ("APP2 sleeping for 1 seconds\n");
82:         LO_SleepTask_WT (1 * 1000);
83:         continue;
84:     } else {
85:         /* usage */
86:         LO_DBG_APP_TRACE ("APP2: unknown command\n");
87:         continue;
88:     }
89:
90:     LO_DBG_APP_TRACE1 ("APP2: running for %d iterations, i.e. send-receive round trips\n");
91:     for (iter = 0; iter < no_iter; iter++) {
```

The interface also shows a tree view on the left with 'ApplicationTasks' (APP_2_TaskB, APP_3_TaskB, APP_4_TaskB) and 'Ports' (Port_A, Port_B, Port_C). The taskbar at the bottom includes icons for 'Пуск', 'Webmail - Opera', 'Total Commander ...', 'Microsoft Word', 'LO_sendreceive...', 'VE2 - Paint', 'XnView - [VE1.JPG]', and system tray icons for 'EM', '9:35', and 'NSE SOCIETY'.

Compiling and running

The screenshot displays the OpenComRTOS Visual Environment interface. The main window shows the source code for 'node.c' with the following content:

```
10: #define LO_CONFIGURED_DATA_STORAGE 1
11:
12: #include <l0node_config.h>
13:
14: #define LO_NODE_NUMBER_OF_PACKETS 5
15: #define LO_NODE_NUMBER_OF_RXDRIVER_PACKETS 3
16: #define LO_NODE_NUMBER_OF_TIMERS 5
17:
18: #ifndef ASYNC_SERVICES
19: LO_Packet LO_KernelPackets[LO_NODE_NUMBER_OF_PACKETS];
20: #endif /* ASYNC_SERVICES */
21:
22: #ifndef MP
23: LO_Packet LO_RxDriverPackets[LO_NODE_NUMBER_OF_PACKETS];
24: #endif /* MP */
25:
26: #ifndef WT_SERVICES
27: #include <l0timerlist_api.h>
28: #include <l0kernel_types.h>
```

The Properties window on the right shows the following configuration:

Name	Value
Node	Node_A
Priority	4
EntryPoint	APP_2_Task
Name	APP_2_TaskB
Arguments	NULL
StackSize	160
Status	LO_STARTED

Below the code editor, three terminal windows show the execution output for Node A, Node B, and Node C. Node A's output shows a sample loop and three iterations of send-receive round trips. Node B and Node C show packet exchange logs for APP3 and APP4 respectively.

```
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 LO_SendPacket: packet_no:1
APP2 LO_ReceivePacket:
APP2 LO_PacketReceived: packet_no:-1
APP2 LO_SendPacket: packet_no:2
APP2 LO_ReceivePacket:
APP2 LO_PacketReceived: packet_no:-2
APP2 LO_SendPacket: packet_no:3
APP2 LO_ReceivePacket:
APP2 LO_PacketReceived: packet_no:-3
-

APP3 LO_ReceivePacket: APP3 LO_PacketReceived: packet_no:1
APP3 LO_SendPacket: packet_no:-1
APP3 LO_ReceivePacket: APP3 LO_PacketReceived: packet_no:2
APP3 LO_SendPacket: packet_no:-2
APP3 LO_ReceivePacket: APP3 LO_PacketReceived: packet_no:3
APP3 LO_SendPacket: packet_no:-3
APP3 LO_ReceivePacket:

APP4 LO_ReceivePacket: APP4 LO_PacketReceived: packet_no:-1
APP4 LO_SendPacket: packet_no:-1
APP4 LO_ReceivePacket: APP4 LO_PacketReceived: packet_no:-2
APP4 LO_SendPacket: packet_no:-2
APP4 LO_ReceivePacket: APP4 LO_PacketReceived: packet_no:-3
APP4 LO_SendPacket: packet_no:-3
APP4 LO_ReceivePacket:
```

Tracing

The screenshot shows the ProcessTracer application window titled "ProcessTracer <alpha>". The main window contains a "Start Tracing" button and a large table of events. The table has columns for "Timestamp", "ID", and "ID". The events are numbered 1 through 31. Below the table is a summary table with columns "F", "G", "C", "TaskID", "Time", and "%".

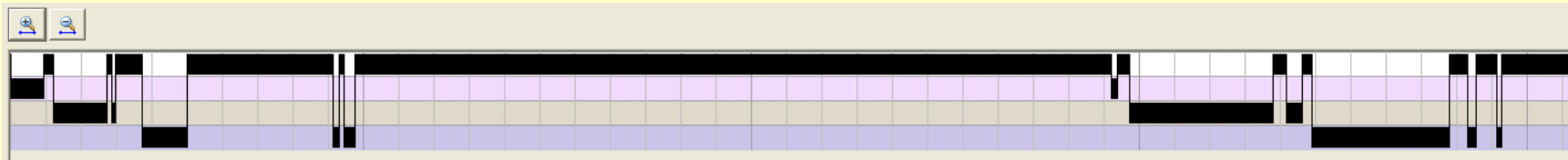
Timestamp	ID	ID
1633724335	2	2
1633725815	0	0
1633725901	0	0
1633726109	0	0
1633726132	0	0
1633726230	3	3
1633726633	0	0
1633726719	0	0
1633726916	0	0
1633727052	0	0
1633727150	2	2
1633727268	0	0
1633727493	1	1
1633727576	0	0
1633728888	1	1
1633728910	1	1
1633729009	3	3
1633729131	0	0
1633738787	1	1
1633738874	0	0
1633739074	1	1
1633739205	1	1
1633739303	2	2
1633739422	1	1
1633784895	0	0
1633784978	0	0
1633785172	0	0
1633785194	0	0
1633785291	3	3
1633785413	1	1
1633788792	0	0

F	G	C	TaskID	Time	%
1	0	5947953	29		
2	2	6931069	34		
3	3	6956114	35		

Three application windows are open, showing logs:

- Node C:** C:\OpenComRTOSVE\Examples\L0_sendreceive_3nodes\Output\Node_C\Node_C.exe. Log shows APP4 L0_ReceivePacket and APP4 L0_SendPacket events with packet numbers -1, -2, and -3.
- Node A:** C:\OpenComRTOSVE\Examples\L0_sendreceive_3nodes\Output\Node_A\Node_A.exe. Log shows "OpenComRTOS Send Recieve 3 Node Sample" and "iter <num> - number of iteration" with iterations 1, 2, and 3.
- Node B:** C:\OpenComRTOSVE\Examples\L0_sendreceive_3nodes\Output\Node_B\Node_B.exe. Log shows APP3 L0_ReceivePacket and APP3 L0_SendPacket events with packet numbers 1, 2, and 3.

OpenComRTOS Event Tracer



TaskTracer 0.3.2

	Time (usec)	Md	Sv	ID
1	0			2
2	874		▶	3
3	904			0
4	961		▶	3
5	1150			
6	1177			3
7	2633			2
8	2662			0

Event Table

Task #0 (Kernel)
Timestamp: 1150 usec
Event: Return from Service

Tasks Diagram

C	TaskID	Task Time (usec)	%
1	0	512762	26
2	2	216001	11.16

Tasks Info

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