



Open License Society

www.OpenLicenseSociety.org

Unifying and systematic system development methodologies
with trustworthy embedded components

Eric.Verhulst@OpenLicenseSociety.org

01/09/2007

Open License Society

1



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

01/09/2007

Open License Society

2



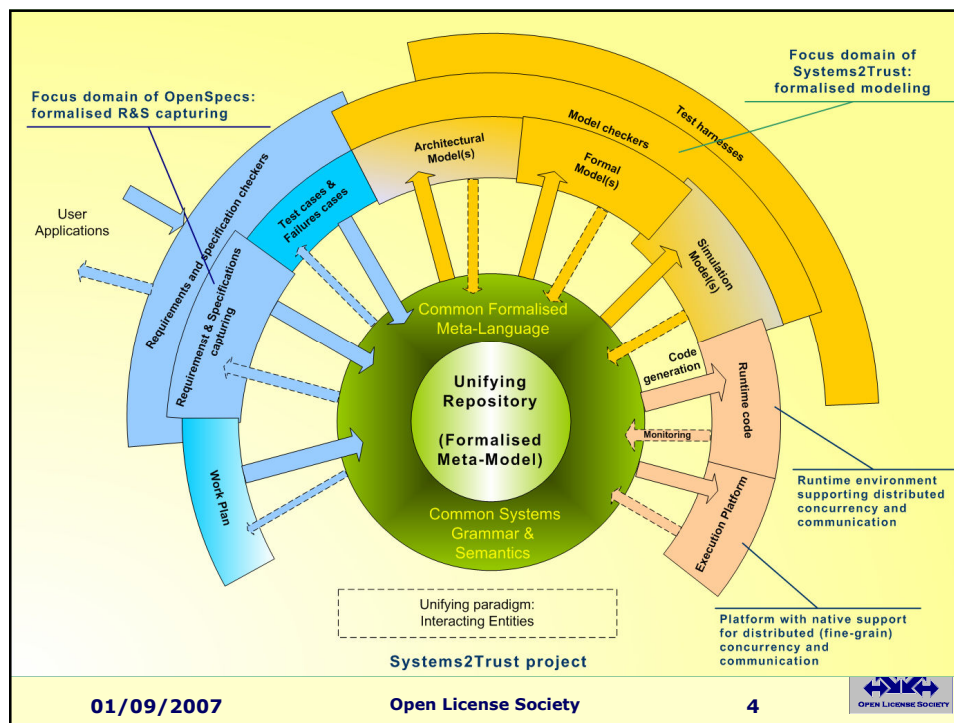
Who is Open License Society?

- 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, ...

01/09/2007

Open License Society

3



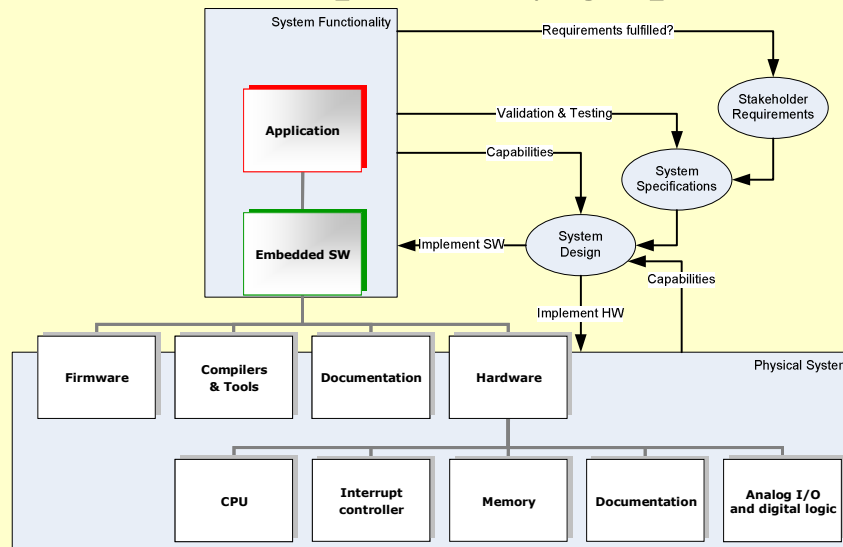
01/09/2007

Open License Society

4



SE Process dependency graph



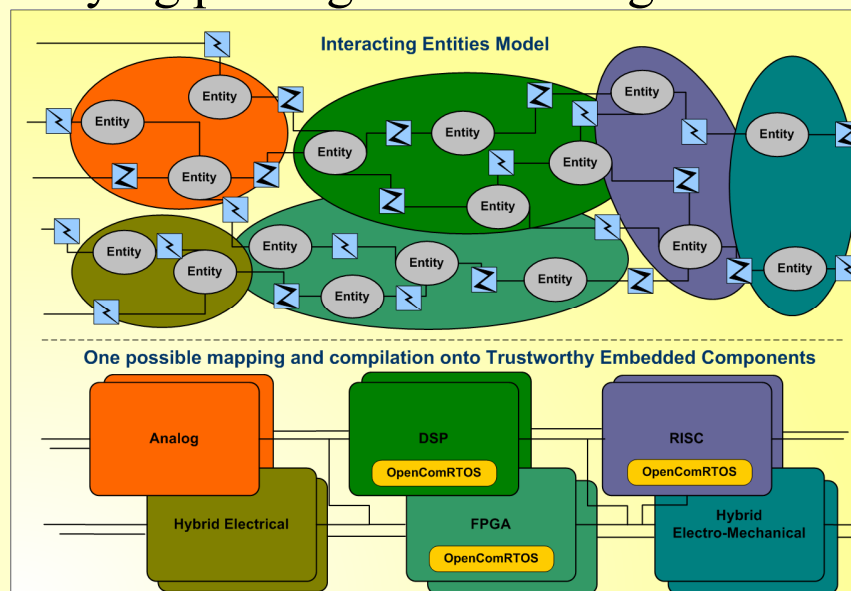
01/09/2007

Open License Society

5



Unifying paradigm: Interacting Entities



01/09/2007

Open License Society

6



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

01/09/2007

Open License Society

7



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)

01/09/2007

Open License Society

8



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.

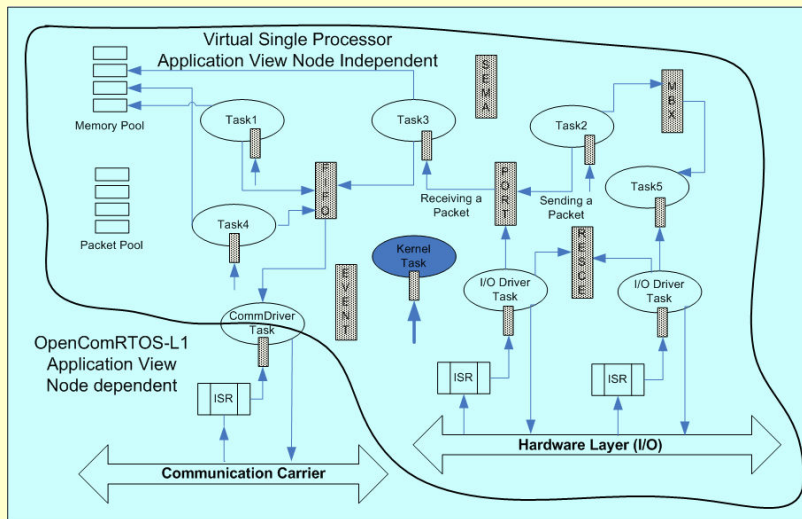
01/09/2007

Open License Society

9



L1 application view



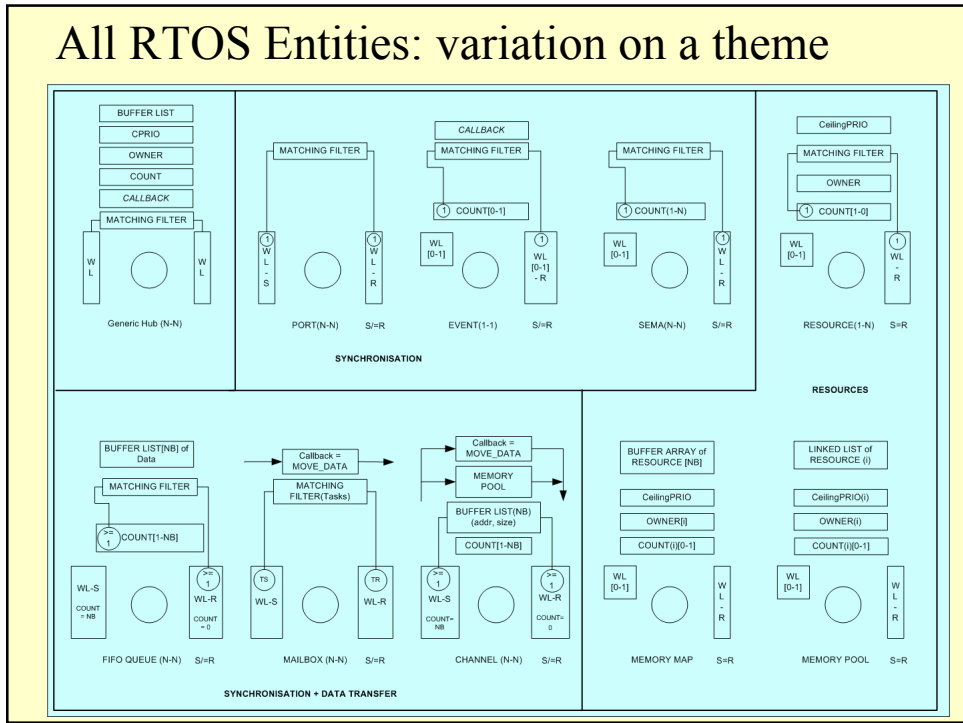
01/09/2007

Open License Society

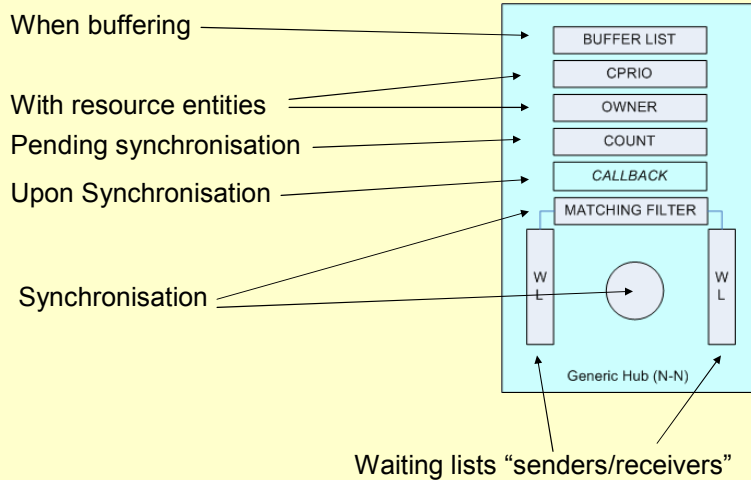
10



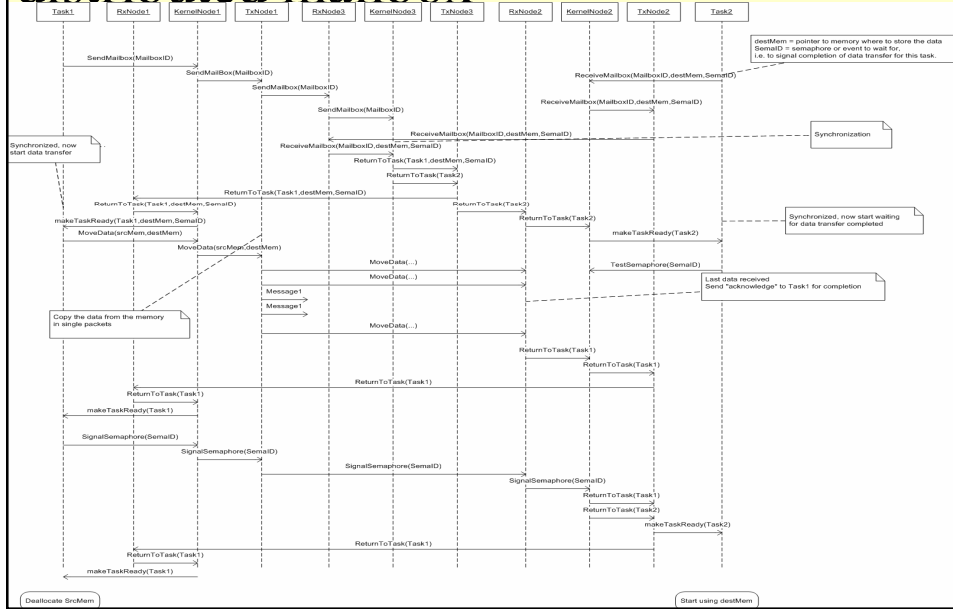
All RTOS Entities: variation on a theme



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

01/09/2007

Open License Society

15



Formal TLA models of OpenComRTOS entities

01/09/2007

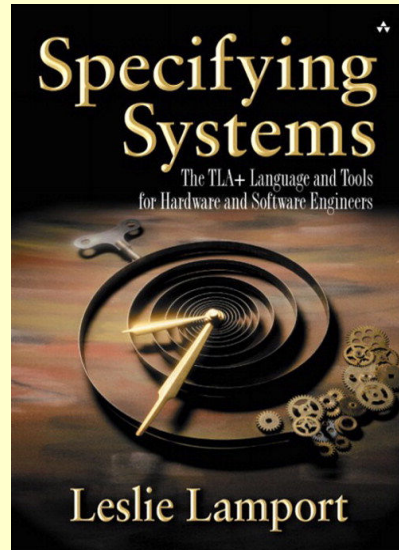
Open License Society

16



TLA

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



01/09/2007

Open License Society

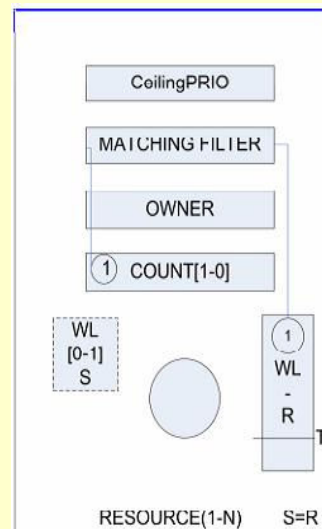
17



TLA modelling results

We modeled entities of OpenComRTOS:

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



01/09/2007

Open License Society

18

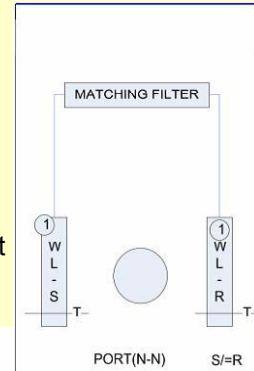


Port

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

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


01/09/2007

Open License Society

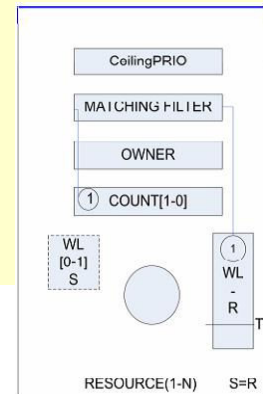
19



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

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


01/09/2007

Open License Society

20



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

```

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

```

01/09/2007

Open License Society

21



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

01/09/2007

Open License Society

22



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>

```

01/09/2007

Open License Society

23



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" />

```

01/09/2007

Open License Society

24



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

01/09/2007

Open License Society

25



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_B is at the bottom, connected to Node_A and Node_C. Node_A and Node_C are at the top, connected to each other. The 'Properties' window on the right shows a table with the following data:

Name	Value
Host	localhost
Name	Node_B

The 'Output' window at the bottom shows the following text:

```
C:\OpenComRTOSVE\bin\mal2h.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'
```

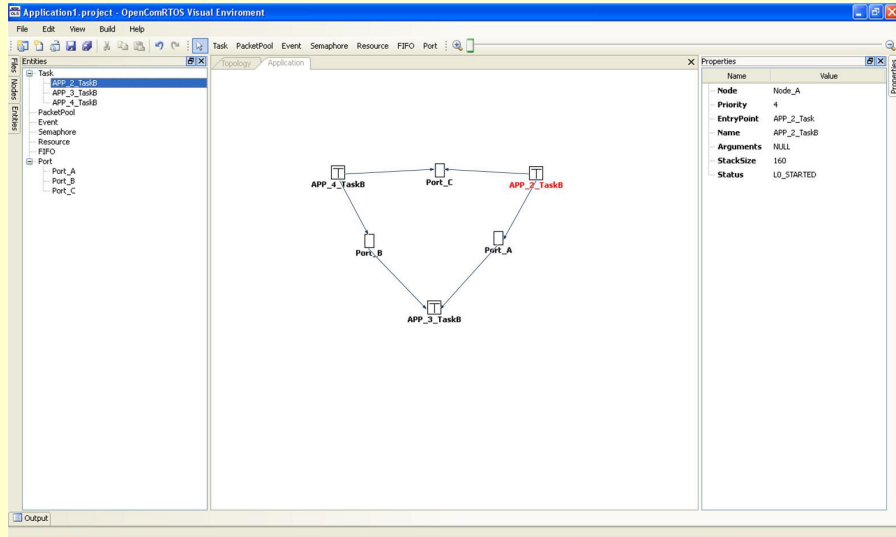
01/09/2007

Open License Society

26



Application structure definition



01/09/2007

Open License Society

27



Task source coding

```
Node Diagram: sendreceive_app_task.c
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 if (strcmp (line, "iter", 4) == 0) {
67     LO_DBG_APP_TRACE ("ITER\n");
68     if (sscanf (line, "iter: %d", &no_iter) != 1) {
69         LO_DBG_APP_TRACE ("APP2: please enter number of iterations\n");
70         continue;
71     }
72     if (no_iter < 1) {
73         LO_DBG_APP_TRACE ("APP2: please enter positive number of iterations\n");
74         continue;
75     }
76 } else if (strcmp (line, "sleep", 5) == 0) {
77     LO_DBG_APP_TRACE ("APP2 sleeping for 1 seconds\n");
78     LO_SleepTask_VT (1 * 1000);
79     continue;
80 } else {
81     /* usage */
82     LO_DBG_APP_TRACE ("APP2: unknown command\n");
83     continue;
84 }
85
86 LO_DBG_APP_TRACE1 ("APP2: running for %d iterations, i.e. send-receive round trips\n",
87 iter);
88 for (iter = 0; iter < no_iter; iter++) {
```

Compiling and running

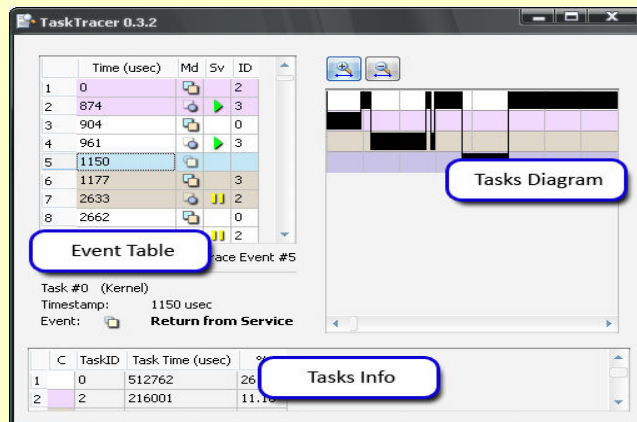
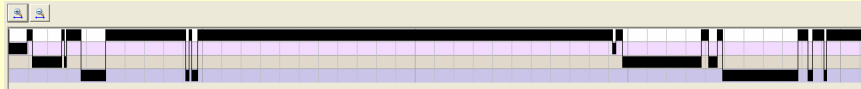
The screenshot displays the OpenComRTOS Visual Environment interface. On the left, a project tree shows the configuration files and source code for a multi-node system. The main window shows the source code for 'node.c', which defines three nodes (Node_A, Node_B, Node_C) and implements a send-receive loop for each. The Properties window on the right shows the configuration for Node_A, including its name, priority, and stack size. The bottom window shows the execution output, which includes the OpenComRTOS boot sequence, the start of the 'Send Receive 3 Node Sample' application, and the execution of three iterations of send-receive round trips between the nodes.

Tracing

The screenshot shows the ProcessTracer tool capturing the execution of the application. The main window displays a list of system events with columns for TimeStamp, ID, and a description of the event. The events include kernel-level operations such as task creation, scheduling, and packet reception. The bottom window shows the execution output, which is the same as the one in the 'Compiling and running' section, but now with a detailed trace overlaying the execution flow, showing the precise timing and sequence of events for each iteration of the send-receive loop.



OpenComRTOS Event Tracer



01/09/2007

Open License Society

31



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

01/09/2007

Open License Society

32

