MP-SoC June 2007

Formal modeling and a network centric Real-Time Operating System in less than 2K Bytes as a generic base for MP-SoC and Process Oriented Programming

www.OpenLicenseSociety.org
www.melexis.com

Unifying and systematic system development methologies
with trustworthy embedded components

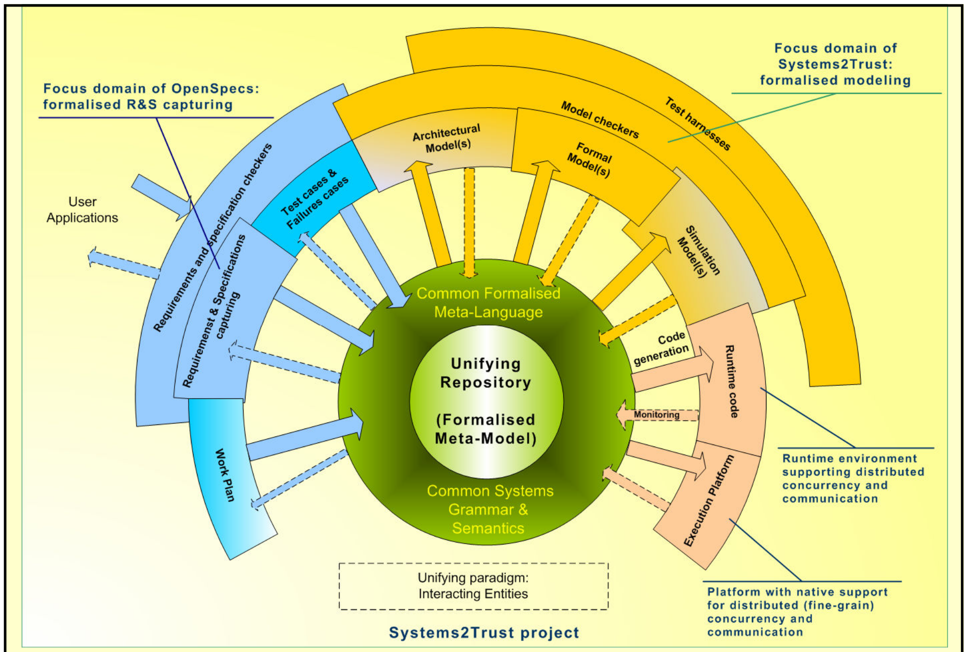
Eric.Verhulst@OpenLicenseSociety.org

Who is Open License Society?

- Privately funded R&D institute
  - Leuven (BE), Berdyansk (UA)
- 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 (formal developed)
  - Implies ‘**Trustworthy Components**’
    - => **Open License** (source code + all design, test, …. docs)
- Focus:
  - Embedded Systems:
    - Constraints driven development
    - Real-time, distributed, hardware & software, …

Some keywords

- Unified semantics:
  - For all “views” (from requirements to platform)
  - Full behavior (at all levels)
  - Interface definition (is more than syntax)
  - ”protocols” rather than messages
- System’s grammar
  - Defining a formalised language
- Meta-modeling
  - Raising the level of abstraction first
- Interacting Entities
  - Then define the architectural level=
    - Entities and Interactions
    - Applies to almost any system domain



Focus domain of OpenSpecs: formalised R&S capturing

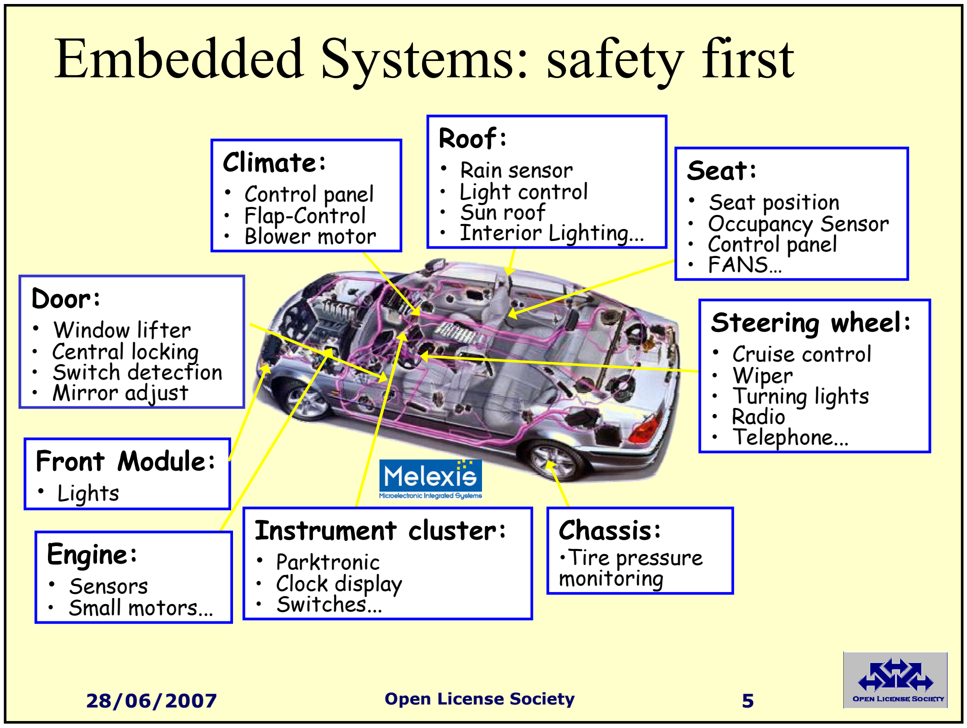
Focus domain of Systems2Trust: formalised modeling

Runtime environment supporting distributed concurrency and communication

Platform with native support for distributed (fine-grain) concurrency and communication

Systems2Trust project

User Applications
Requirements and specification checkers
Requirement & Specifications capturing
Test cases & Failures cases
Work Plan
Architectural Model(s)
Model checkers
Test harnesses
Formal Model(s)
Simulation Model(s)
Code generation
Runtime code
Monitoring
Execution Platform
Common Formalised Meta-Language
Unifying Repository
(Formalised Meta-Model)
Common Systems Grammar & Semantics
Unifying paradigm: Interacting Entities



Embedded Systems: safety first

Climate:
- Control panel
- Flap-Control
- Blower motor

Roof:
- Rain sensor
- Light control
- Sun roof
- Interior Lighting…

Seat:
- Seat position
- Occupancy Sensor
- Control panel
- FANS…

Door:
- Window lifter
- Central locking
- Switch detection
- Mirror adjust

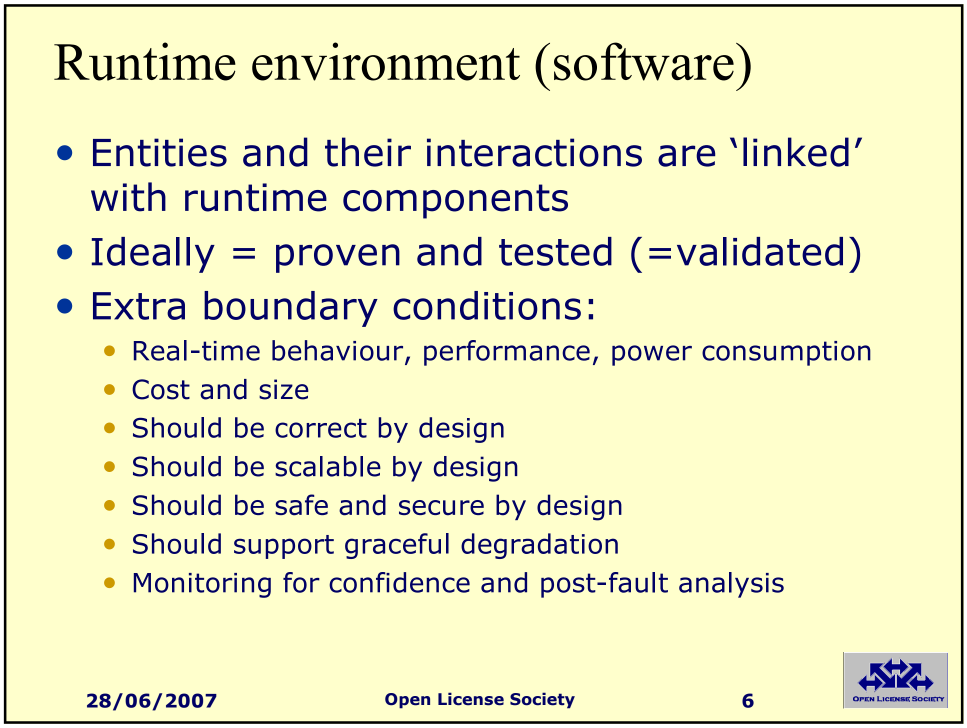
Steering wheel:
- Cruise control
- Wiper
- Turning lights
- Radio
- Telephone…

Front Module:
- Lights

Engine:
- Sensors
- Small motors…

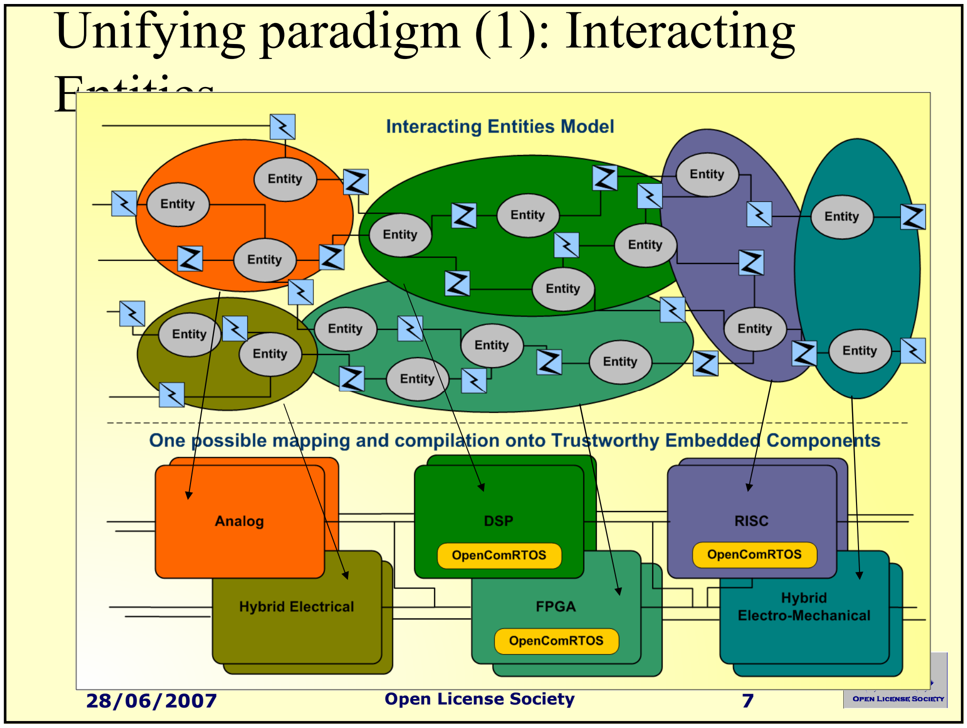
Instrument cluster:
- Parktronic
- Clock display
- Switches…

Chassis:
·Tire pressure monitoring



Runtime environment (software)

- Entities and their interactions are ‘linked’ with runtime components
- Ideally = proven and tested (=validated)
- Extra boundary conditions:
  - Real-time behaviour, performance, power consumption
  - Cost and size
  - Should be correct by design
  - Should be scalable by design
  - Should be safe and secure by design
  - Should support graceful degradation
  - Monitoring for confidence and post-fault analysis



Unifying paradigm (1): Interacting Entities

Interacting Entities Model

Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity
Entity

One possible mapping and compilation onto Trustworthy Embedded Components

Analog

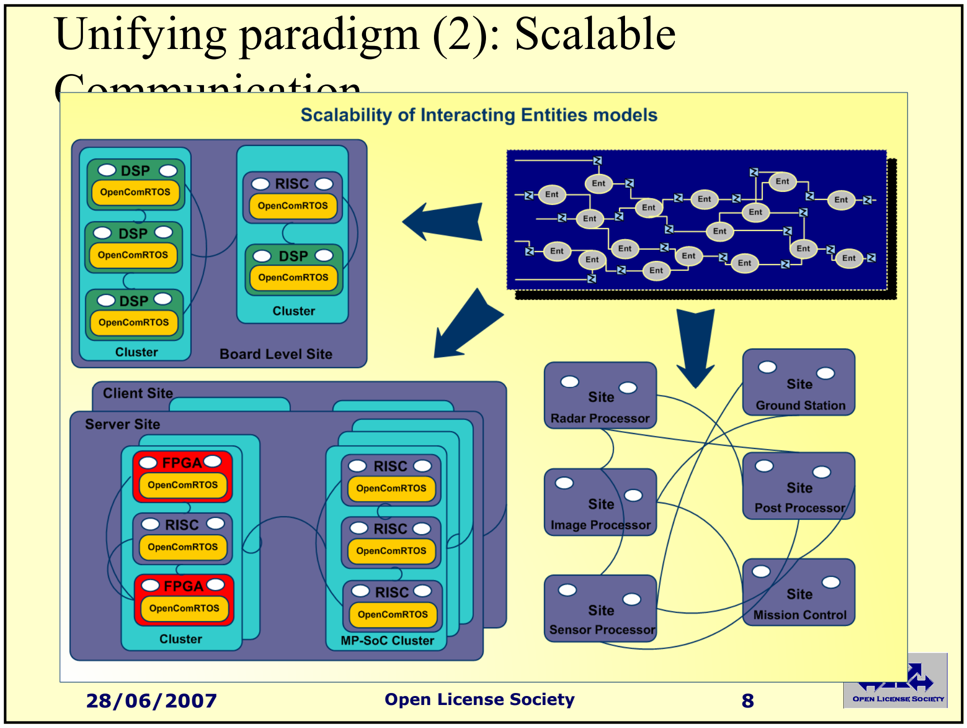
DSP
OpenComRTOS

RISC
OpenComRTOS

Hybrid Electrical

FPGA
OpenComRTOS

Hybrid
Electro-Mechanical



Unifying paradigm (2): Scalable Communication

Scalability of Interacting Entities models

DSP
OpenComRTOS
DSP
OpenComRTOS
DSP
OpenComRTOS
Cluster

RISC
OpenComRTOS
DSP
OpenComRTOS
Cluster

Board Level Site

Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent Ent

Client Site
Server Site

FPGA
OpenComRTOS
RISC
OpenComRTOS
FPGA
OpenComRTOS
Cluster

RISC
OpenComRTOS
RISC
OpenComRTOS
RISC
OpenComRTOS
MP-SoC Cluster

Site
Radar Processor

Site
Ground Station

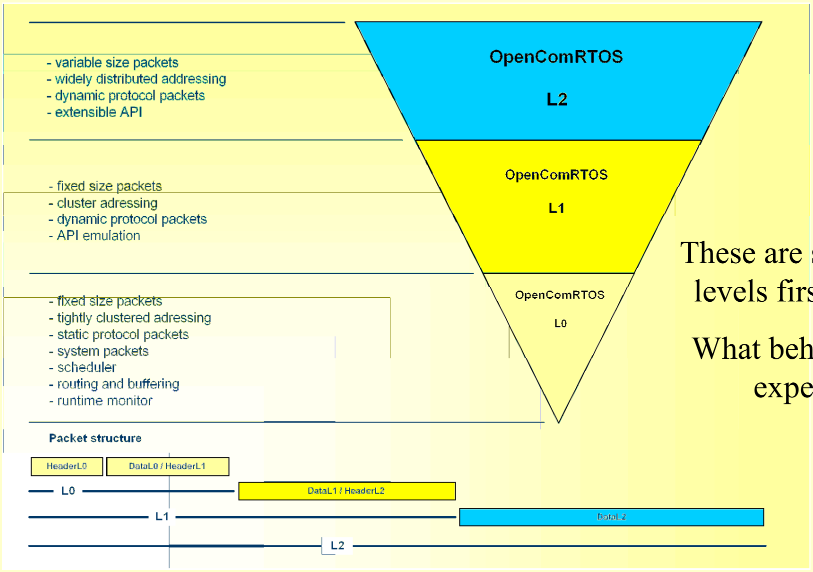
Site
Image Processor

Site
Post Processor

Site
Sensor Processor

Site
Mission Control

OpenComRTOS: formally developed



- variable size packets
- widely distributed addressing
- dynamic protocol packets
- extensible API

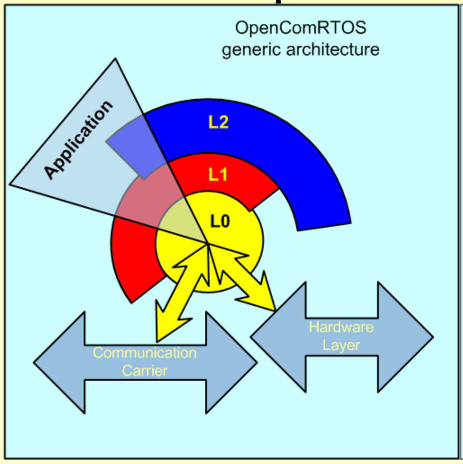
- fixed size packets
- cluster adressing
- dynamic protocol packets
- API emulation

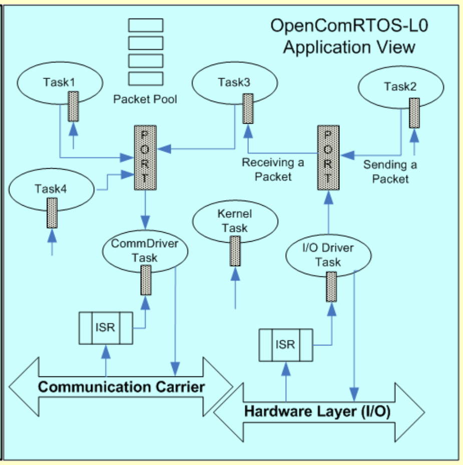
- fixed size packets
- tightly clustered adressing
- static protocol packets
- system packets
- scheduler
- routing and buffering
- runtime monitor

These are semantic levels first of all:

What behavior to expect?

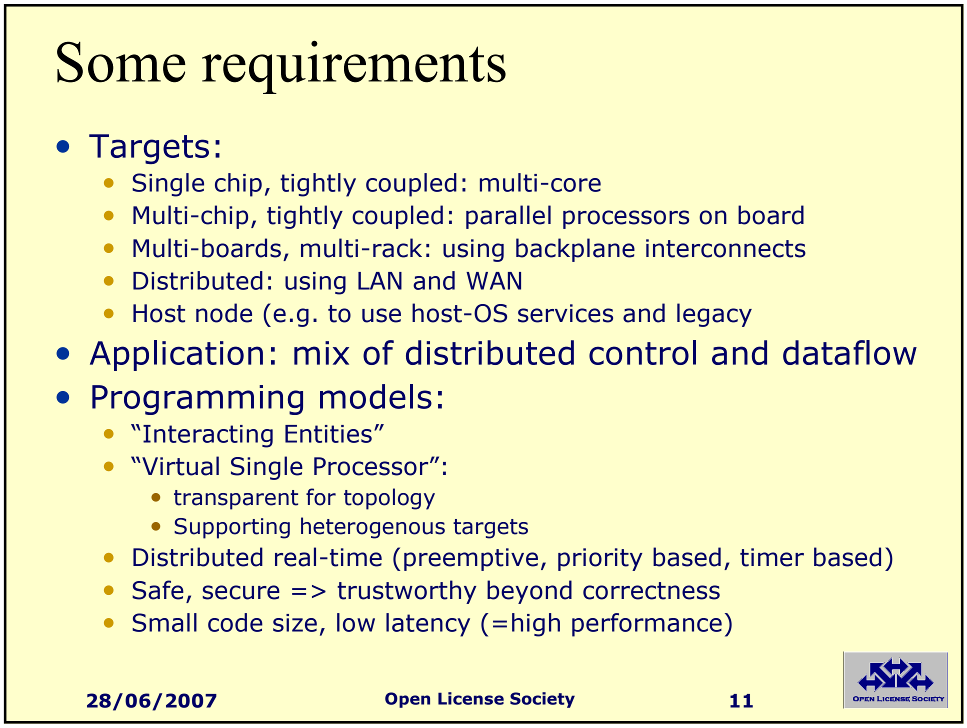
Generic Open-Comm-RTOS





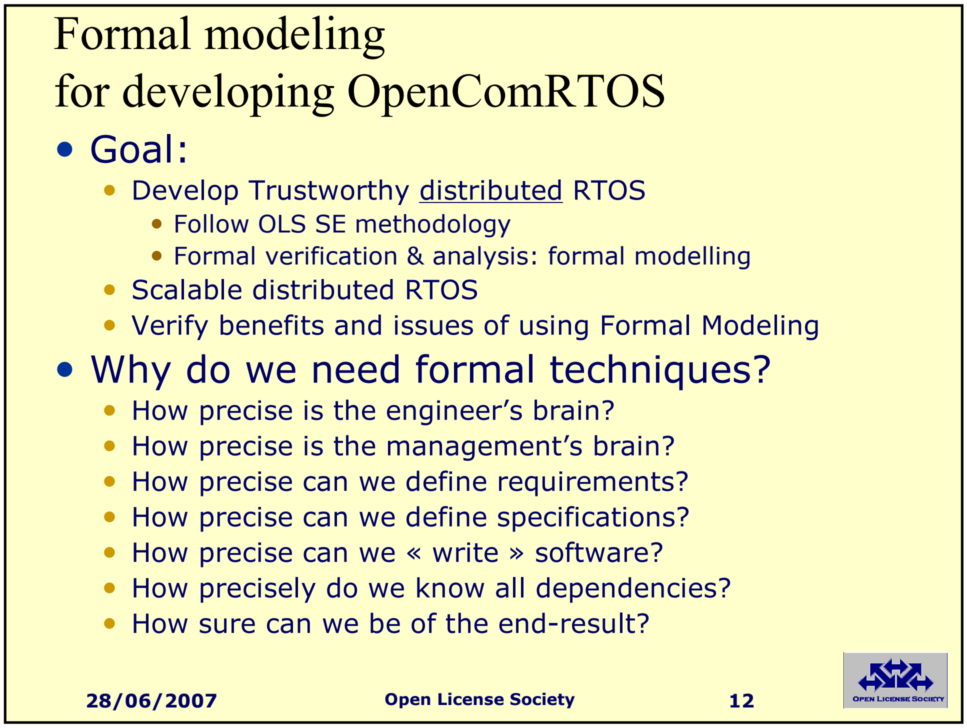
Based on (scalable) “packet switching” at all levels

Tasks (entities) and interactions decoupled



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 (e.g. to use host-OS services and legacy
- Application: mix of distributed control and dataflow
- Programming models:
  - "Interacting Entities"
  - "Virtual Single Processor":
    - transparent for topology
    - Supporting heterogenous targets
  - Distributed real-time (preemptive, priority based, timer based)
  - Safe, secure => trustworthy beyond correctness
  - Small code size, low latency (=high performance)



Formal modeling
for developing OpenComRTOS

- 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
- Why do we need formal techniques?
  - How precise is the engineer's brain?
  - How precise is the management's brain?
  - How precise can we define requirements?
  - How precise can we define specifications?
  - How precise can we « write » software?
  - How precisely do we know all dependencies?
  - How sure can we be of the end-result?

Can we trust our mind ?

- How many « F » did you find ?

FINISHED FILES ARE THE RE
SULT OF YEARS OF SCIENTIF-
IC STUDY COMBINED WITH
THE EXPERIENCE OF YEARS

Did you see the similarity with source code (debugging) ?

Formal modeling tools

- Default mathematical approach:
  - Correctness by proof
    - Labor and time intensive
    - Needs specialists
    - (Human) Error prone process
  - Tools needed
    - State space is exponentially large
    - Issues always in « hidden corners »
    - Allow incremental process
  - Requirements:
    - Support state machines
    - Support concurrency and communication
    - Low notational barrier

Formal modeling tools: selected options

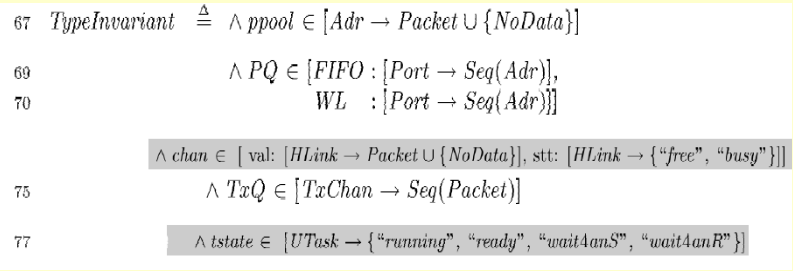
- Investigated:
  - SPIN, B, CSP/FDR, **TLA+/TLC**
- Outcome of process:
  - SPIN OK, initially preferred, good documentation, wide user base, but very C-like style
  - CSP: hard notation, FDR not readily available
  - B: waiting for Event B, incremental approach and compositionality very good
  - TLA+/TLC
    - Based on Temporal Logic
    - Mathematical notation, but standard
    - Works for any domain (SW, HW, …)
    - (but not for large models)

Benefits of TLA+/TLC

- TLA+/TLC home page on http://research.microsoft.com/users/lamport/tla/tla.html
- Initial models reflected “programming style”
  - That’s the way the mind works (after being conditioned …)
  - > 28 successive models from 2 pages to 25 pages
    - Initially very abstract, neglecting details
    - All successive models were correct, why ?
      - Iterative, incremental process!
      - Takes 15 minutes from one model to the next
    - Interplay between software architects and formal modeling engineer
      - Architectural model polluted by programming concepts
      - Abstraction from TLA helped to find these issues
      - Result: much cleaner, safer and performant architecture
- TLA models do not prove software is correct (! ?)
  - TLC proves that Formal **Models** are correct

Formally modeled

```
TypeInvariant  == /\ ppool \in [Adr-> Packet \union {NoData}]
                  /\ PQ \in [ FIFO : [Port -> Seq(Adr)],
                              WL   : [Port -> Seq(Adr)]]
                 /\ chan \in [ val: [HLink -> Packet \union {NoData}],
                                 stt: [HLink -> {"free","busy"}]]
                  /\ TxQ \in [TxChan -> Seq(Packet)]
                  \*    /\ tstate \in [UTask ->{"running","ready","wait4anS","wait4anR"}]
```



$$TypeInvariant \triangleq \wedge ppool \in [Adr \rightarrow Packet \cup \{NoData\}]$$

$$\wedge PQ \in [FIFO : [Port \rightarrow Seq(Adr)],$$

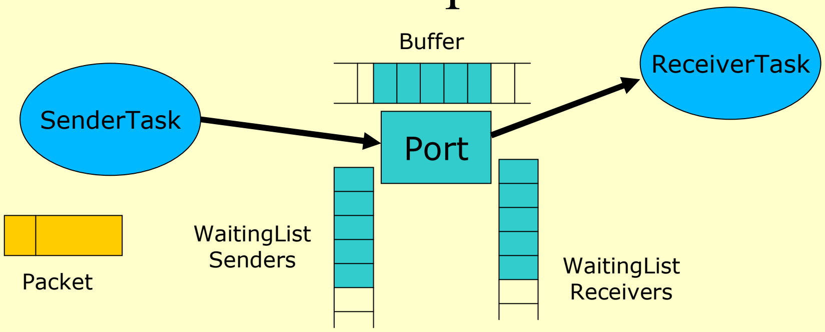
$$WL : [Port \rightarrow Seq(Adr)]]$$

$$\wedge chan \in [\text{val}: [HLink \rightarrow Packet \cup \{NoData\}], \text{stt}: [HLink \rightarrow \{\text{“}free\text{”}, \text{“}busy\text{”}\}]]$$

$$\wedge TxQ \in [TxChan \rightarrow Seq(Packet)]$$

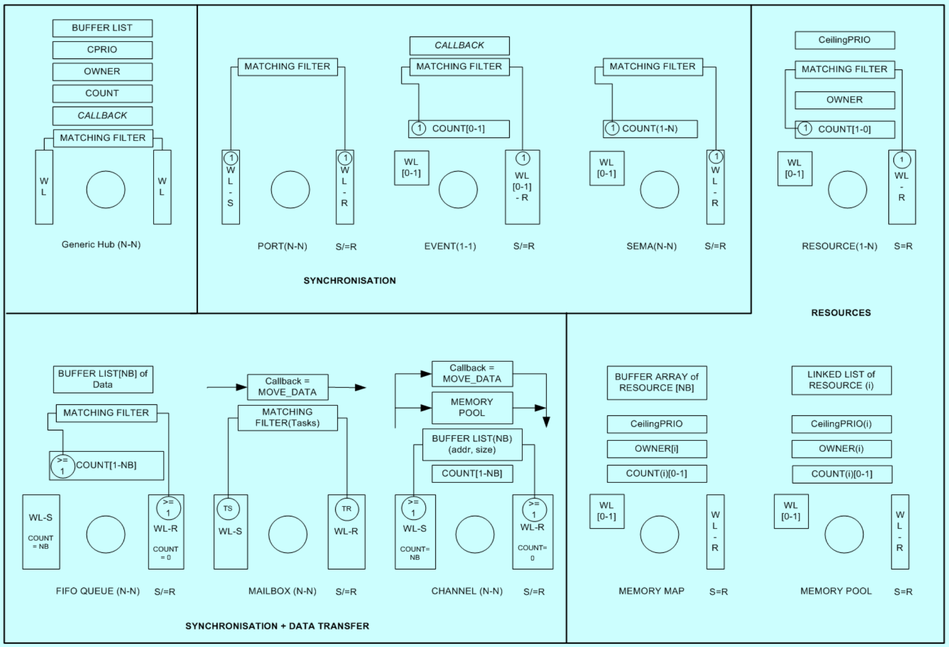
$$\wedge tstate \in [UTask \rightarrow \{\text{“}running\text{”}, \text{“}ready\text{”}, \text{“}wait4anS\text{”}, \text{“}wait4anR\text{”}\}]$$

One result as example



- Need for either FIFO Buffer or WaitingList
  - Both (abstract) models are the same
  - Natural language is imprecise, semantics are context driven
- Benefits:
  - Infinite buffering until no more memory (for Packets)
  - Overflow-free buffering

All (typical) RTOS Entities: variations on a theme



Generic hub: => define your own entities and interactions

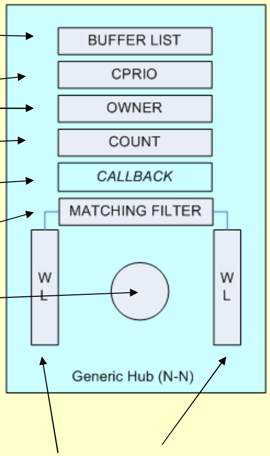
When buffering

With resource entities

Pending synchronisation

Upon Synchronisation

Synchronisation

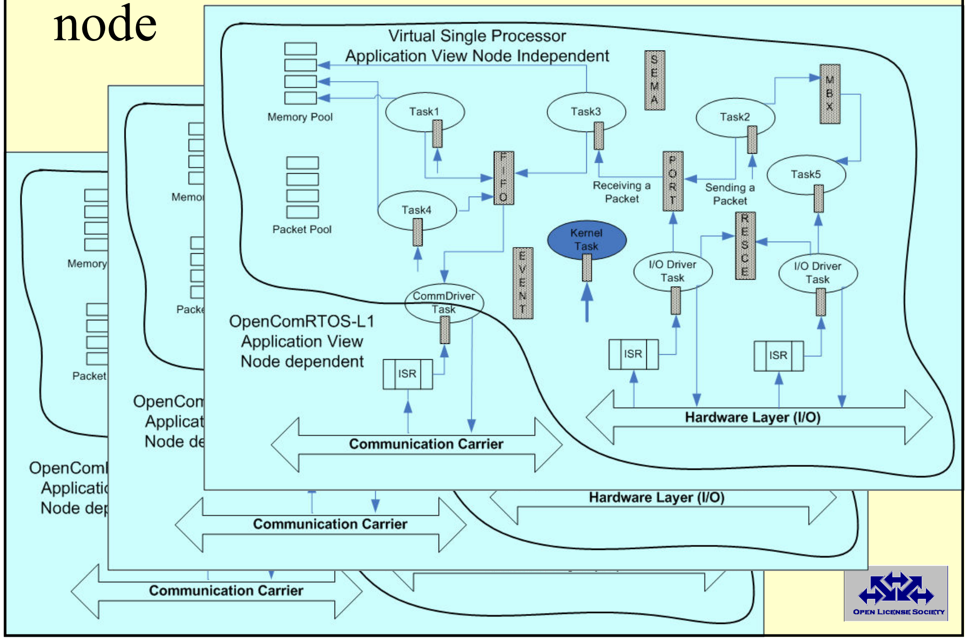


Waiting lists “senders/receivers”

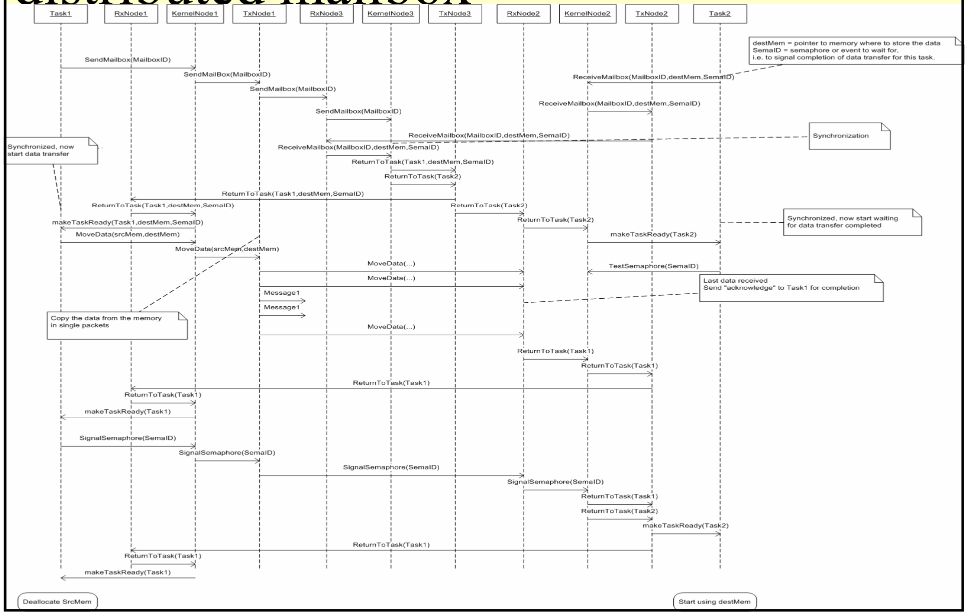
L1 entities

L1 Entity	Semantics
Event	Synchronisation on Boolean value. Waiting list on both sides.
Counting Semaphore	Synchronisation with counter allowing asynchronous signaling.
Port	Synchronisation with exchange of a Packet.
FIFO queue	Buffered communication of Packets. Synchronisation when queue is full or empty.
Resource	Event used to create a logical critical section. Resources have an owner Task when locked
Memory Pool	Linked list of memory blocks protected with a resource
Mailbox	Synchronising entity with matching filter on Task ID. Communication happens as side-effect.
Channel	Asynchronous communication between Tasks with buffering using memory pools. Communication as a side-effect.

any entity can be mapped onto any node



Example of Interaction diagram: distributed mailbox



destMem = pointer to memory where to store the data
SemaID = semaphore or event to wait for,
i.e. to signal completion of data transfer for this task.

Synchronization

Synchronized, now start data transfer

Synchronized, now start waiting for data transfer completed

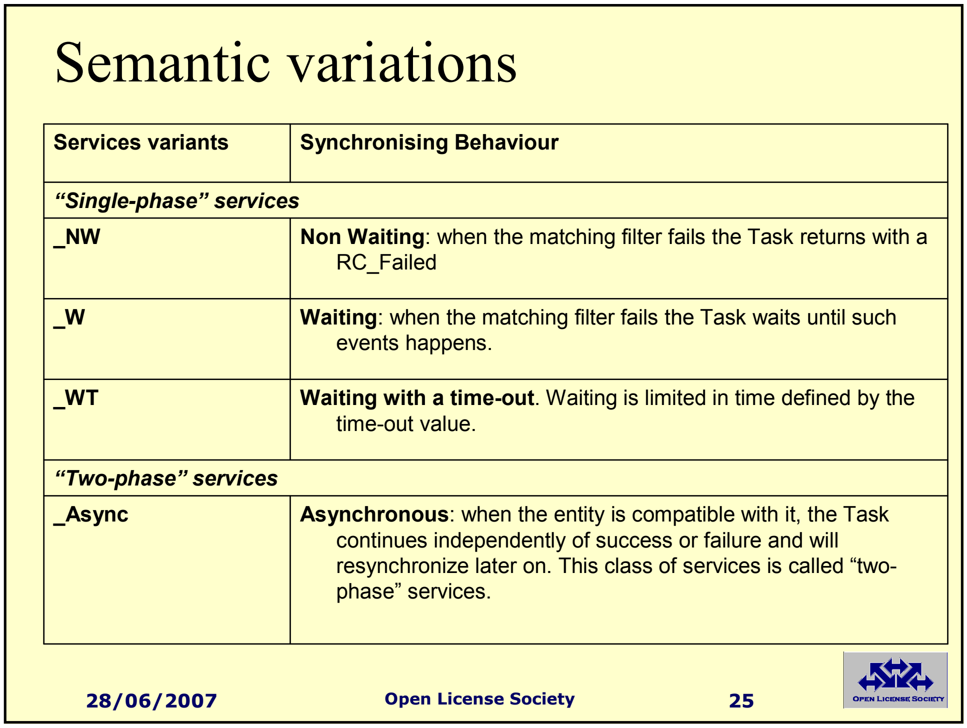
Last data received
Send "acknowledge" to Task1 for completion

Copy the data from the memory in single packets

Clean architecture gives small code

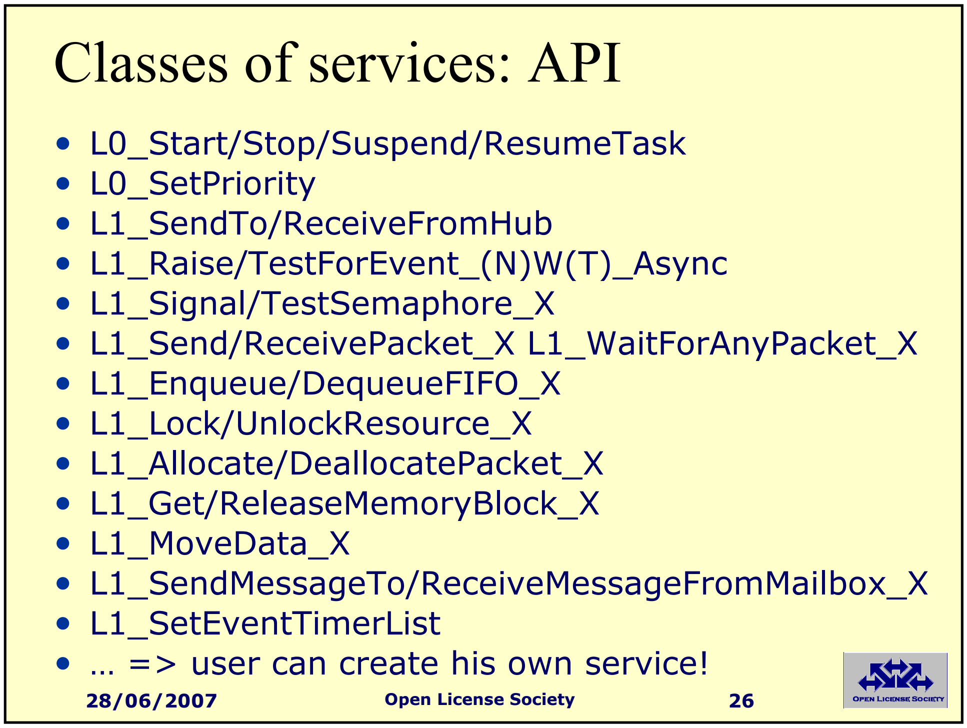
OpenComRTOS L1 code size figures (MLX16)				
	MP FULL		SP SMALL	
	L0	L1	L0	L1
L0 Port	162		132	
L1 Hub shared		574		400
L1 Port		4		4
L1 Event		68		70
L1 Semaphore		54		54
L1 Resource		104		104
L1 FIFO		232		232
L1 Resource List		184		184
Total L1 services		1220		1048
Grand Total	**3150**	**4532**	**996**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



Semantic variations

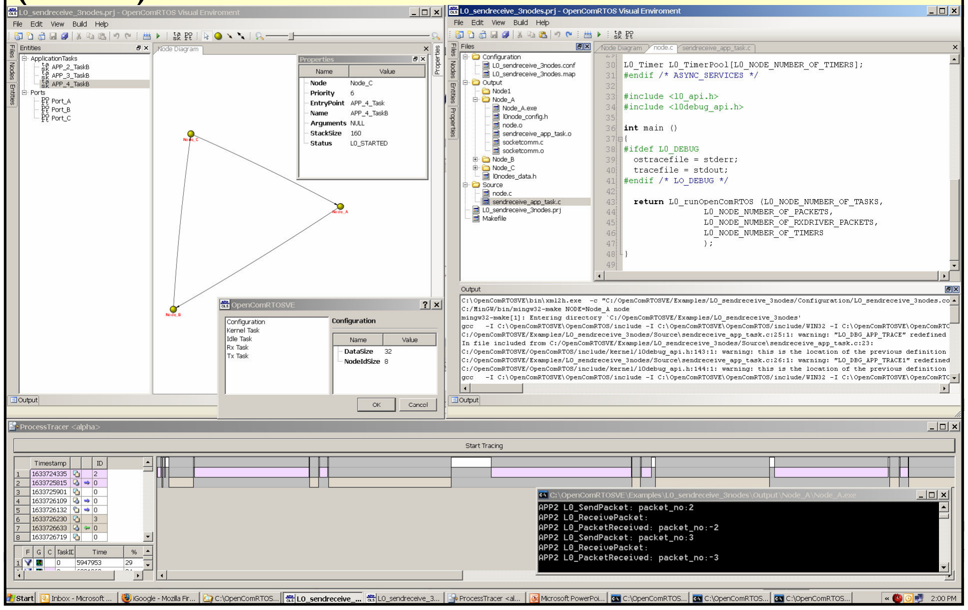
Services variants	**Synchronising Behaviour**
“Single-phase” services	
_NW	**Non Waiting**: when the matching filter fails the Task returns with a RC_Failed
_W	**Waiting**: when the matching filter fails the Task waits until such events happens.
_WT	**Waiting with a time-out**. Waiting is limited in time defined by the time-out value.
“Two-phase” services	
_Async	**Asynchronous**: when the entity is compatible with it, the Task continues independently of success or failure and will resynchronize later on. This class of services is called “two-phase” services.



Classes of services: API

- L0_Start/Stop/Suspend/ResumeTask
- L0_SetPriority
- L1_SendTo/ReceiveFromHub
- L1_Raise/TestForEvent_(N)W(T)_Async
- L1_Signal/TestSemaphore_X
- L1_Send/ReceivePacket_X L1_WaitForAnyPacket_X
- L1_Enqueue/DequeueFIFO_X
- L1_Lock/UnlockResource_X
- L1_Allocate/DeallocatePacket_X
- L1_Get/ReleaseMemoryBlock_X
- L1_MoveData_X
- L1_SendMessageTo/ReceiveMessageFromMailbox_X
- L1_SetEventTimerList
- … => user can create his own service!

OpenComRTOS Visual Environment (beta)



Properties

Name	Value
Node	Node_C
Priority	6
EntryPoint	APP_4_Task
Name	APP_4_TaskB
Arguments	NULL
StackSize	160
Status	L0_STARTED

OpenComRTOSVE

Configuration
Kernel Task
Idle Task
Rx Task
Tx Task

Configuration

Name	Value
DataSize	32
NodeIdSize	8

OK Cancel

```
L0_Timer L0_TimerPool[L0_NODE_NUMBER_OF_TIMERS];
#endif /* ASYNC_SERVICES */

#include <l0_api.h>
#include <l0debug_api.h>

int main ()
{
#ifdef L0_DEBUG
  ostracefile = stderr;
  tracefile = stdout;
#endif /* L0_DEBUG */

  return L0_runOpenComRTOS (L0_NODE_NUMBER_OF_TASKS,
                  L0_NODE_NUMBER_OF_PACKETS,
                  L0_NODE_NUMBER_OF_RXDRIVER_PACKETS,
                  L0_NODE_NUMBER_OF_TIMERS
                  );
}
```

```
C:\OpenComRTOSVE\bin\xml2h.exe  -c "C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes/Configuration/L0_sendreceive_3nodes.co
C:/MinGW/bin/mingw32-make NODE=Node_A node
mingw32-make[1]: Entering directory `C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes'
gcc   -I C:\OpenComRTOSVE\OpenComRTOS/include -I C:\OpenComRTOSVE\OpenComRTOS/include/WIN32 -I C:\OpenComRTOSVE\OpenComRTC
C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes/Source\sendreceive_app_task.c:25:1: warning: "L0_DBG_APP_TRACE" redefined
In file included from C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes/Source\sendreceive_app_task.c:23:
C:/OpenComRTOSVE/OpenComRTOS/include/kernel/l0debug_api.h:143:1: warning: this is the location of the previous definition
C:/OpenComRTOSVE/Examples/L0_sendreceive_3nodes/Source\sendreceive_app_task.c:26:1: warning: "L0_DBG_APP_TRACE1" redefined
C:/OpenComRTOSVE/OpenComRTOS/include/kernel/l0debug_api.h:144:1: warning: this is the location of the previous definition
gcc   -I C:\OpenComRTOSVE\OpenComRTOS/include -I C:\OpenComRTOSVE\OpenComRTOS/include/WIN32 -I C:\OpenComRTOSVE\OpenComRTC
```

	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

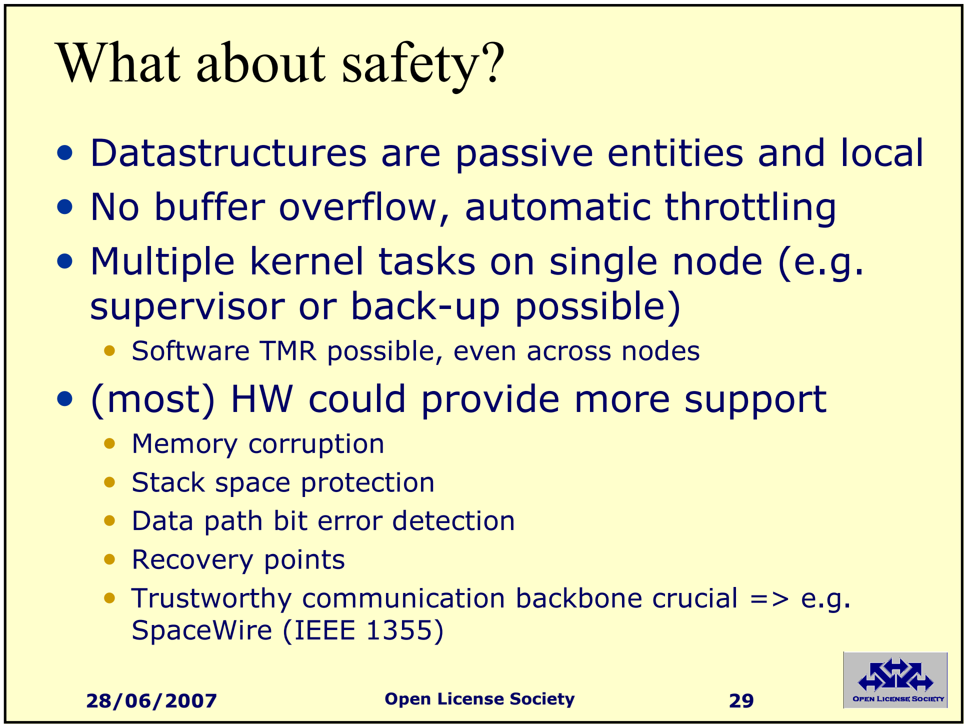
	F	G	C	TaskID	Time	%
1				0	5947953	29

C:\OpenComRTOSVE\Examples\L0_sendreceive_3nodes\Output\Node_A\Node_A.exe

```
APP2 L0_SendPacket: packet_no:2
APP2 L0_ReceivePacket:
APP2 L0_PacketReceived: packet_no:-2
APP2 L0_SendPacket: packet_no:3
APP2 L0_ReceivePacket:
APP2 L0_PacketReceived: packet_no:-3
```

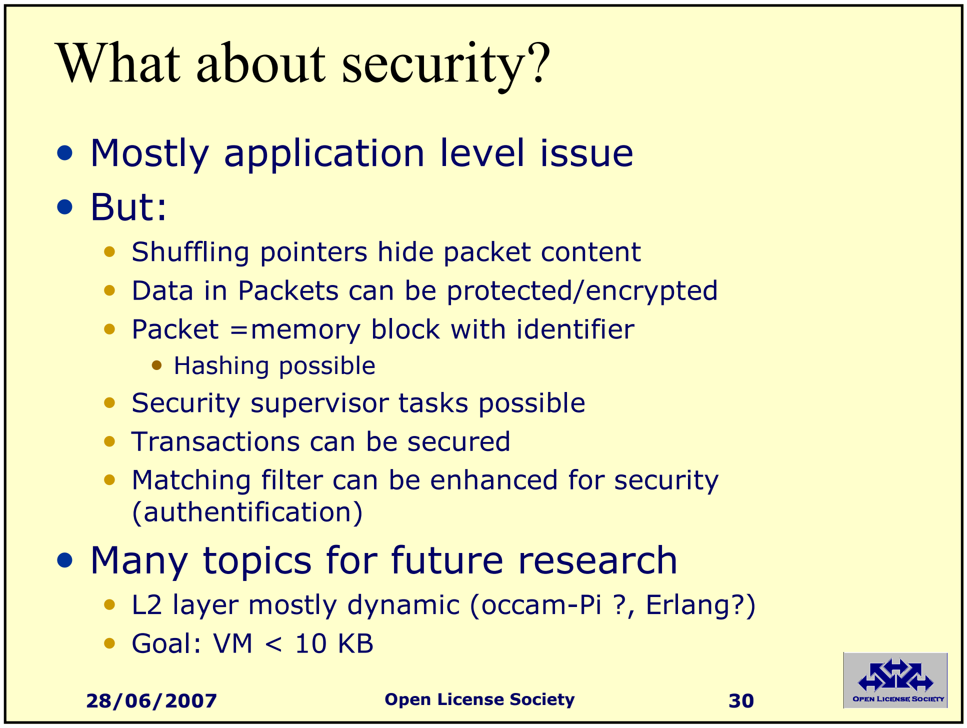
What about real-time scheduling?

- von neumann machine is resource: need to share time: => scheduling
  - Scheduling should be orthogonal to application logic
  - Timer based or priority based but preemptive
  - Priority inheritance mechanism needed, but akward to implement (code is everywhere)
- Communication backbone is resource: need to share medium
  - issues are latency, P2P bandwidth, buffering
  - => packet switching
  - => priorities inherited
- Architecture allows accepting Interrupts to be done on different CPU than the one processing the Int.



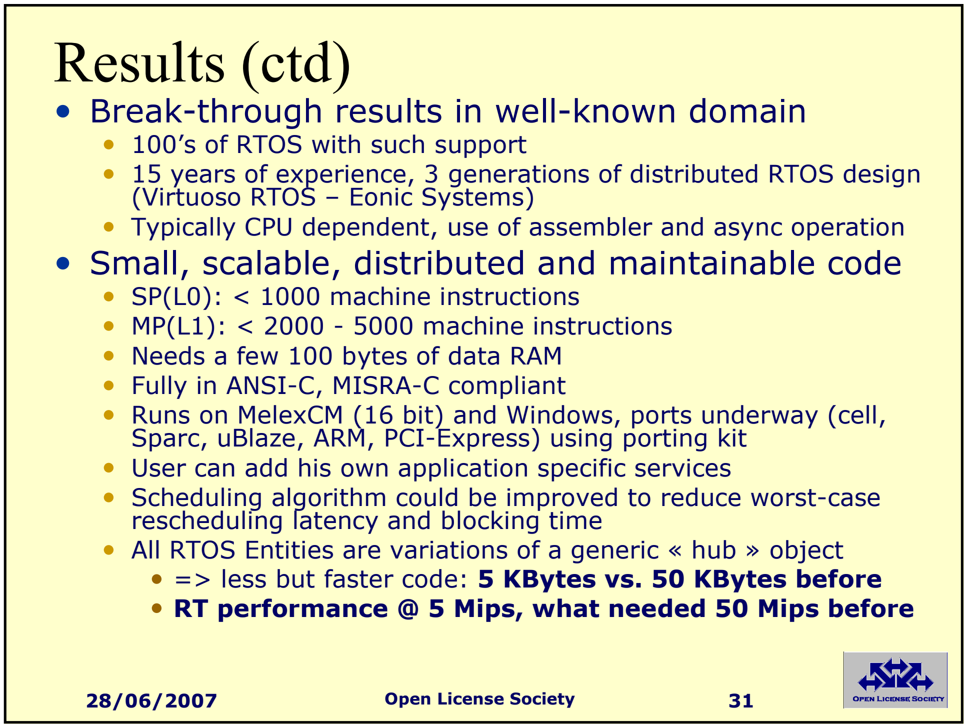
What about safety?

- Datastructures are passive entities and local
- No buffer overflow, automatic throttling
- Multiple kernel tasks on single node (e.g. supervisor or back-up possible)
  - Software TMR possible, even across nodes
- (most) HW could provide more support
  - Memory corruption
  - Stack space protection
  - Data path bit error detection
  - Recovery points
  - Trustworthy communication backbone crucial => e.g. SpaceWire (IEEE 1355)



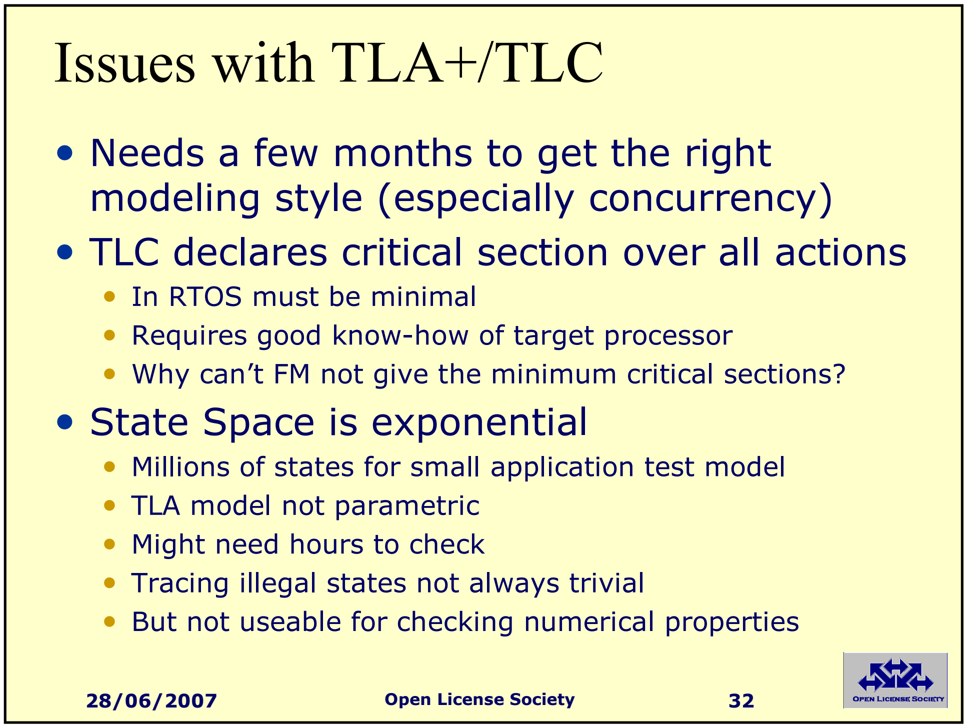
What about security?

- Mostly application level issue
- But:
  - Shuffling pointers hide packet content
  - Data in Packets can be protected/encrypted
  - Packet =memory block with identifier
    - Hashing possible
  - Security supervisor tasks possible
  - Transactions can be secured
  - Matching filter can be enhanced for security (authentification)
- Many topics for future research
  - L2 layer mostly dynamic (occam-Pi ?, Erlang?)
  - Goal: VM < 10 KB



Results (ctd)

- Break-through results in well-known domain
  - 100’s of RTOS with such support
  - 15 years of experience, 3 generations of distributed RTOS design (Virtuoso RTOS – Eonic Systems)
  - 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, ports underway (cell, Sparc, uBlaze, ARM, PCI-Express) using porting kit
  - User can add his own application specific services
  - Scheduling algorithm could 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**
    - **RT performance @ 5 Mips, what needed 50 Mips before**



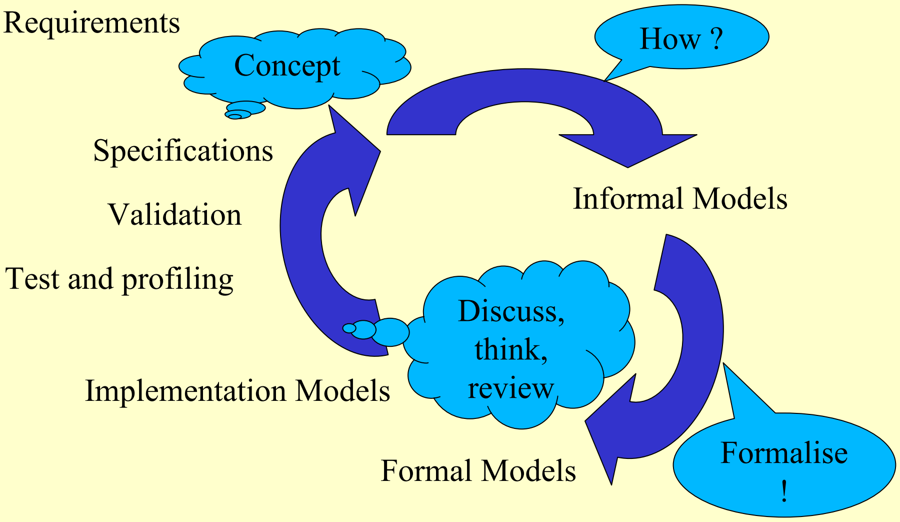
Issues with TLA+/TLC

- Needs a few months to get the right modeling style (especially concurrency)
- TLC declares critical section over all actions
  - In RTOS must be minimal
  - Requires good know-how of target processor
  - Why can’t FM not give the minimum critical sections?
- State Space is exponential
  - Millions of states for small application test model
  - TLA model not parametric
  - Might need hours to check
  - Tracing illegal states not always trivial
  - But not useable for checking numerical properties

Key observations

- Successive iterations: evolutionary
  - > 28 successive models from 2 pages to 25 pages
  - Initially very abstract, neglecting details
  - All successive models were correct, why ?
    - Iterative, incremental process!
    - Takes 15 minutes from one model to the next
- Interaction and abstraction
  - Interplay between SW architects and formal modeling engineer
  - Architectural model polluted by programming concepts
  - Abstraction from TLA helped to find these issues
  - Formalised thinking
- Much cleaner, safer and performant architecture
- Caveat: FM do not prove software is correct (! ?)
  - Proves that Formal **Models** are correct

How it really works: teamwork



Requirements

Concept

How ?

Specifications

Validation

Informal Models

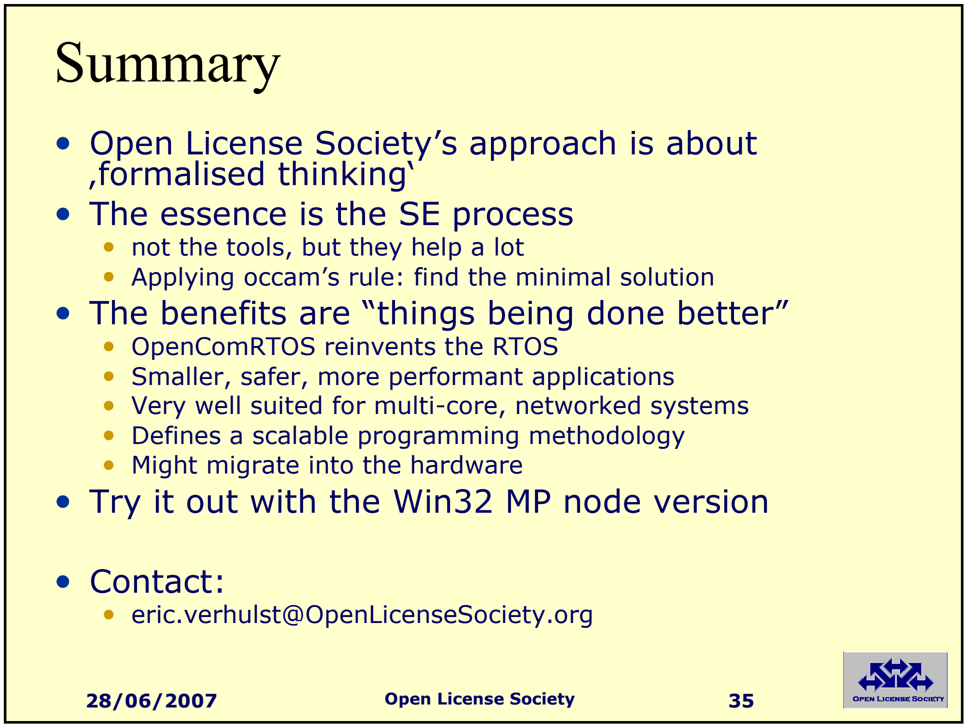
Test and profiling

Discuss, think, review

Implementation Models

Formal Models

Formalise !



Summary

- Open License Society’s approach is about ‚formalised thinking‘
- The essence is the SE process
  - not the tools, but they help a lot
  - Applying occam’s rule: find the minimal solution
- The benefits are “things being done better”
  - OpenComRTOS reinvents the RTOS
  - Smaller, safer, more performant applications
  - Very well suited for multi-core, networked systems
  - Defines a scalable programming methodology
  - Might migrate into the hardware
- Try it out with the Win32 MP node version

- Contact:
  - eric.verhulst@OpenLicenseSociety.org