Eric Verhulst, CEO/CTO

Altreonic

occam's Rule Applied: Separation of Concerns as a Key to Trustworthy Systems Engineering for Complex Systems

Altreonic profile

- 30 years of aero-space-defense experience (**Eonic Systems NV**)
  - Specialised in parallel **Virtuoso** Real-Time Operating System
  - Used from 1 to 1600 processors (sonar, radar) to 12000 nodes
  - Used by ESA (Virtuoso RTOS on Rosetta mission)
  - Virtuoso acquired by Wind River Systems Inc. in 2001
- **Altreonic**: created as new spin-off in 2008 after R&D
  - Focus on **trustworthy scalable embedded systems**
    - Safety, Security, Usability, Privacy
  - Using formal methods => **OpenComRTOS Designer & VirtuosoNext**
  - Portal based environment to support SE: **GoedelWorks**
  - Unique "Open Technology License" model
  - Competitive advantage to develop **KURT novel e-vehicle**

The emergence of a CSP mindset

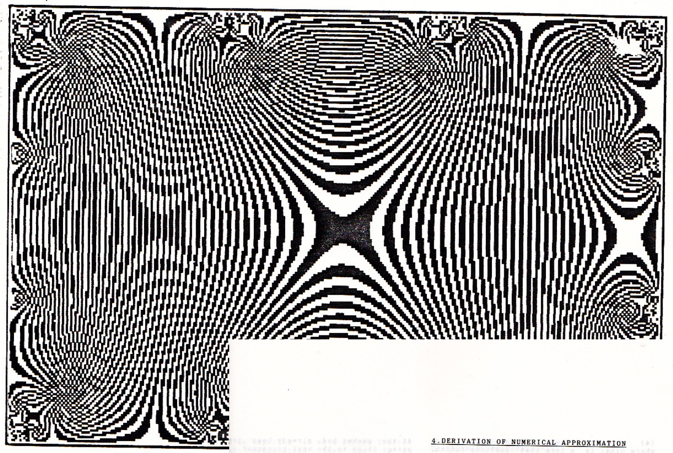
- How to build a holographic radar?
  - 640x480 = 307200 input sensors and phase shifters
  - Virtual beam steering gives high resolution
  - Needs very precise timing synchronisation
  - Straightforward but heavy processing and high bandwidth communication in real-time
  - Clearly not a job for a 5 MHz single processor

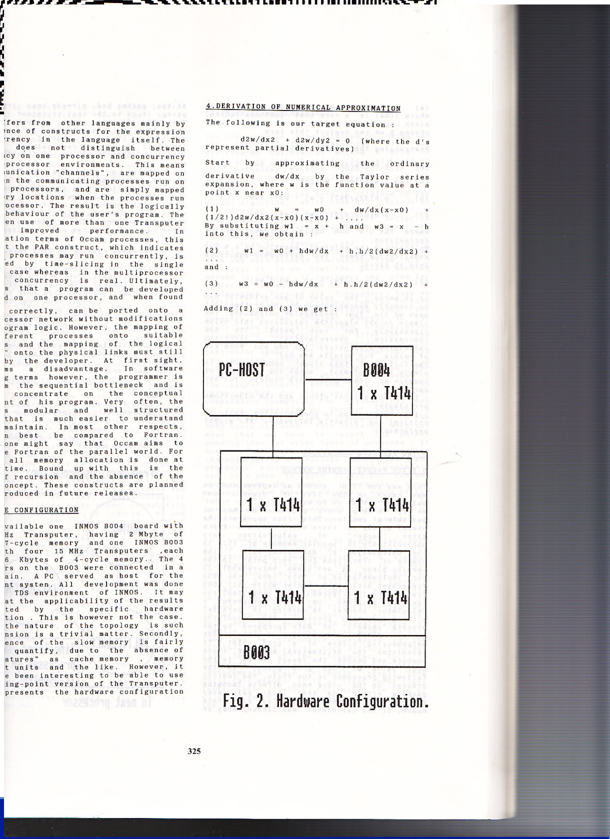
=> parallel computer

- = parallel computation + parallel communication
- GigaFlops, GigaBytes/sec.

Preambule: Laplace

- What is the potential in a point given boundary conditions?
- Hard way: solve n linear equations
- Easy way: let it iterate
  - needs more processing
- Is it a stable solution? Ignored.





ffers from other languages mainly by
ence of constructs for the expression
rency in the language itself. The
does not distinguish between
cy on one processor and concurrency
processor environments. This means
unication "channels", are mapped on
n the communicating processes run on
processors, and are simply mapped
ry locations when the processes run
ocessor. The result is the logically
behaviour of the user's program. The
en use of more than one Transputer
improved performance. In
ation terms of Occam processes, this
t the PAR construct, which indicates
processes may run concurrently, is
ed by time-slicing in the single
case whereas in the multiprocessor
concurrency is real. Ultimately,
s that a program can be developed
d on one processor, and when found

correctly, can be ported onto a
cessor network without modifications
ogram logic. However, the mapping of
ferent processes onto suitable
s and the mapping of the logical
" onto the physical links must still
by the developer. At first sight,
ms a disadvantage. In software
g terms however, the programmer is
m the sequential bottleneck and is
concentrate on the conceptual
nt of his program. Very often, the
s modular and well structured
that is much easier to understand
maintain. In most other respects,
n best be compared to Fortran.
one might say that Occam aims to
e Fortran of the parallel world. For
all memory allocation is done at
time. Bound up with this is the
f recursion and the absence of the
oncept. These constructs are planned
roduced in future releases.

E CONFIGURATION

vailable one INMOS B004 board with
Hz Transputer, having 2 Mbyte of
7-cycle memory and one INMOS B003
th four 15 MHz Transputers ,each
6 Kbytes of 4-cycle memory.. The 4
rs on the B003 were connected in a
ain. A PC served as host for the
nt systen. All development was done
TDS environment of INMOS. It may
at the applicability of the results
ted by the specific hardware
tion . This is however not the case.
the nature of the topology is such
nsion is a trivial matter. Secondly,
ence of the slow memory is fairly
quantify, due to the absence of
atures" as cache memory , memory
t units and the like. However, it
e been interesting to be able to use
ing-point version of the Transputer.
presents the hardware configuration

4.DERIVATION OF NUMERICAL APPROXIMATION

The following is our target equation :

d2w/dx2 + d2w/dy2 = 0 (where the d's represent partial derivatives)

Start by approximating the ordinary derivative dw/dx by the Taylor series expansion, where w is the function value at a point x near x0:

(1) w = w0 + dw/dx(x-x0) + (1/2!)d2w/dx2(x-x0)(x-x0) + ...
By substituting w1 = x + h and w3 = x - h into this, we obtain :

(2) w1 = w0 + hdw/dx + h.h/2(dw2/dx2) + ...
and :

(3) w3 = w0 - hdw/dx + h.h/2(dw2/dx2) + ...

Adding (2) and (3) we get :

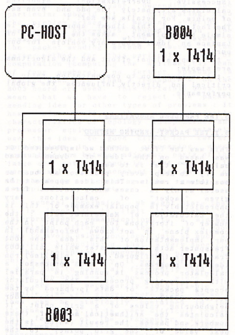


Fig. 2. Hardware Configuration.

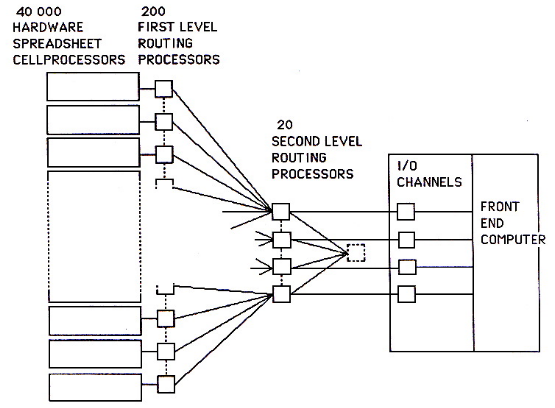
325

Super Lilly

- Idea for a parallel super computer
- Ref.: "*Super-Lilly : A Concept for a User Friendly Supercomputer." Second European Simulation Conference. October 1986.*
- Connect processors using optical fibers writing directly into memory
- Practical?

Super Lilly

- Key question:
  - how do you program it?



40 000 HARDWARE SPREADSHEET CELLPROCESSORS
200 FIRST LEVEL ROUTING PROCESSORS
20 SECOND LEVEL ROUTING PROCESSORS
I/O CHANNELS
FRONT END COMPUTER

FIG. 2. GENERAL LAYOUT.

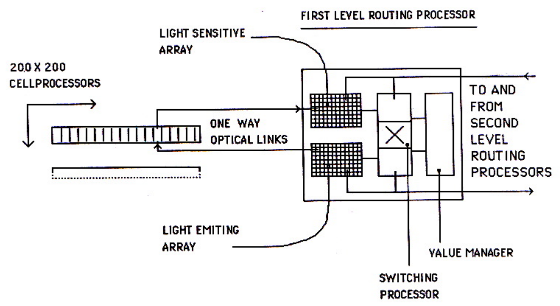


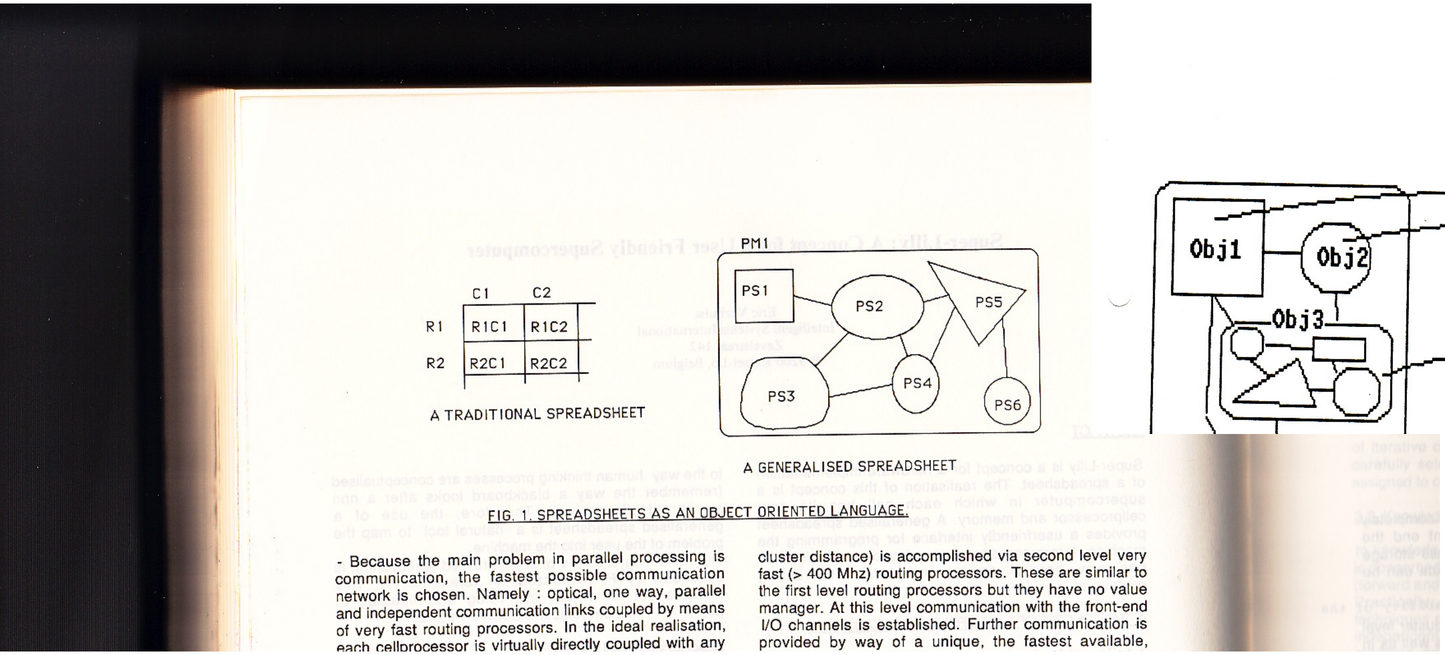
FIG.3. FIRST LEVEL ROUTING PROCESSOR.

409

Handy Lilly (1)

- Inspired by spreadsheet and Laplace problem
  - *"Generalized Spreadsheets for Programming Parallel Computers." FBVI congress. January 1987.*
  - *"Handy Lilly : a high level user friendly programming environment for solving Partial Differential Equations on a network of Transputers."*
  - *ESPRIT II proposal 2405. April 1988.*
  - *"Solving the Equation of Laplace on a Network of Transputers." Paper presented at the Simulation Conference in June 88, Nice, France.*
  - *"A Prototype High Level Programming Environment for Solving Partial Differential Equations on a Network of Transputers." Paper presented at USER1 (Simulation Society), Ostend, September 1988.*
  - *"Solving partial differential equations on a transputer network.". Paper presented at "The Use of Supercomputers in Theoretical Science IV" Workshop. Antwerp, UIA, June 1988. Published by PLENUM*

Handy Lilly (2)



C1
C2
R1
R1C1
R1C2
R2
R2C1
R2C2
PM1
PS1
PS2
PS3
PS4
PS5
PS6

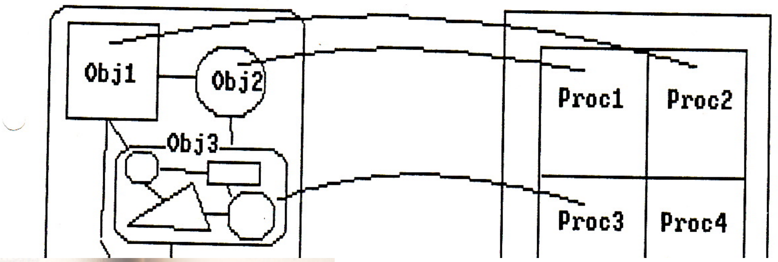
A TRADITIONAL SPREADSHEET

A GENERALISED SPREADSHEET

FIG. 1. SPREADSHEETS AS AN OBJECT ORIENTED LANGUAGE.

- Because the main problem in parallel processing is communication, the fastest possible communication network is chosen. Namely : optical, one way, parallel and independent communication links coupled by means of very fast routing processors. In the ideal realisation, each cellprocessor is virtually directly coupled with any

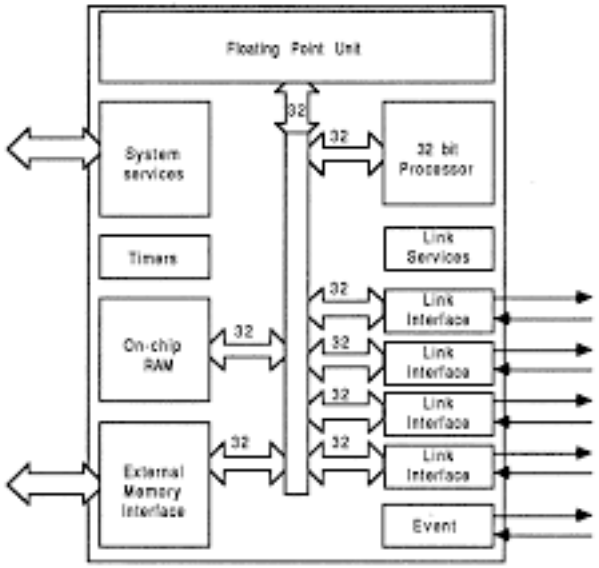
cluster distance) is accomplished via second level very fast (> 400 Mhz) routing processors. These are similar to the first level routing processors but they have no value manager. At this level communication with the front-end I/O channels is established. Further communication is provided by way of a unique, the fastest available,



- Each cell is a processor, a reference to another cell is a message containing cell values
- Works because order of calculations is given
- Parallelisation by splitting up in frames
- Bottleneck: exchange of overlapping border values

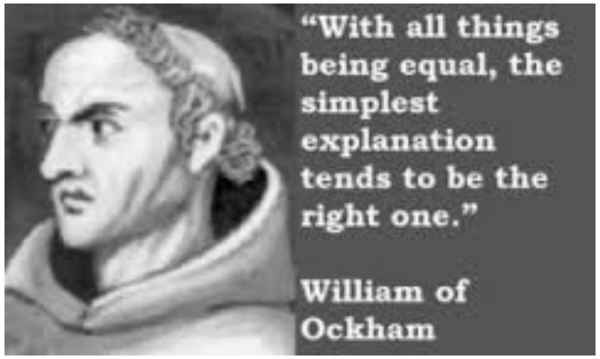
The transputer enters the scene

- Exciting news about a real parallel processor
- It had links! It had processes!
- There were boards (expensive)
  - INMOS B004, B003
- Taking a loan to start playing
- That's a good reason to start a small business: Reselling transputer boards and Meiko's



With it came occam

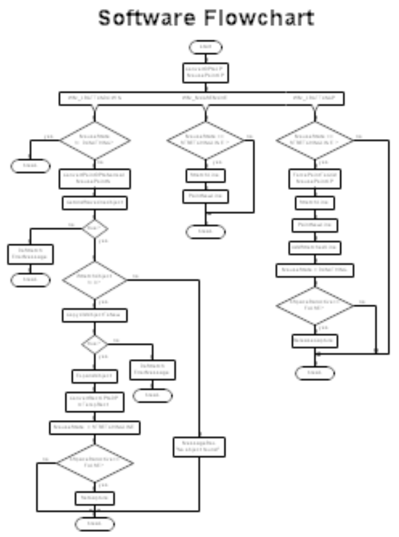
- Funny, radical language, simpler than Pascal
- Processes: PAR, SEQ
- Channels: In? Out! ALT in?
- Data types: Bytes only (sic)
- Unforgiving scoping indents! (great)
- MP support:
  - Connect the links
  - Reroute in the processes when remapping
  - Very tedious plumbing (channels connect directly)


"With all things being equal, the simplest explanation tends to be the right one."
William of Ockham

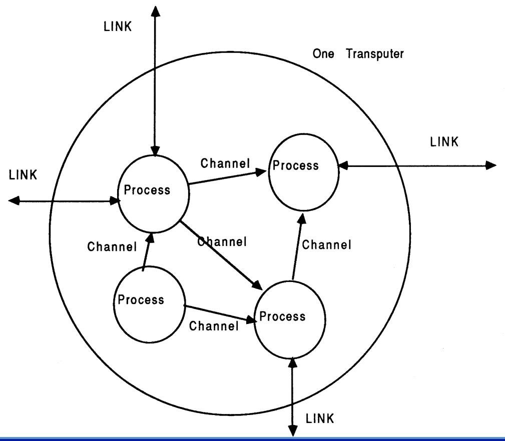
Enters Peter, the master of occam

- Pascal was OK, so occam should be easy?
- After one month, still struggling!?
- Compiler was very "strict"
- Very frustrating
- I didn't know what was happening
- So, I signed up for an occam programming workshop at Canterbury with Peter

The CSP click



- Day one: nice intro, but nothing new
- Day two: first exercise, disaster struck
- Diagnosis: **if-then-else syndrome**
- Brains are trained/conditioned to tackle the whole state space at once => can’t handle it
- Separation of concerns:
  - What are the functions? => processes
  - How are they related? => channels
  - Rest is plumbing it all together



TROS: Transputer Real-Time Operating System

- Project with Uni Leuven & 4 students
- "TROS : a real-time fault tolerant operating system for Transputers." Paper presented at the 11th occam user group. Edingburgh. October 1989.
- A fault tolerant transputer OS written in occam
- Triplication and voting, error-resilient communication, application unchanged
- It worked (but very slowly)
- Coding it was a nightmare (no data structures)
- => moving to C

Towards a Real Real-Time OS

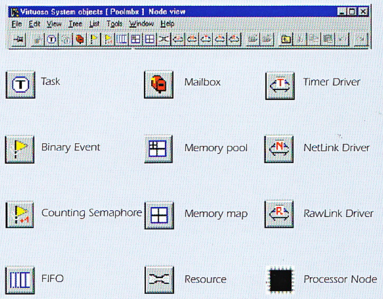
- Challenge 1: context switch
  - "forbidden" by INMOS
  - Assembler was just "discovered"
  - Solution: swap the Lo Process Queue and map to N priorities (Rate Monotonic Scheduler)
- Challenge 2:
  - Find a suitable RTOS => RTXC
  - Porting not that difficult
  - But SP semantics only:
    - No MP awareness
    - Services passed pointers

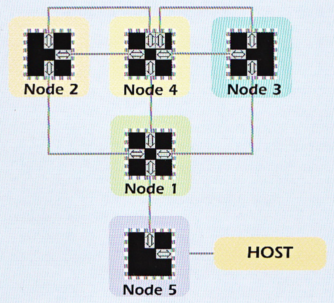
This became the Virtuoso RTOS

- Adapt service “semantics” to be MP compatible
  - Pass by value, not pass by reference
  - Packets replace remote function calls
  - Semaphores, FIFOs, ….
- How to make it MP and transparent?
  - If remote, put service request and params in a packet across the wire
  - Simple routing: look-up table for next link
  - Took about 2 months
- First RTOS on transputer in 1991, Sunnyvale

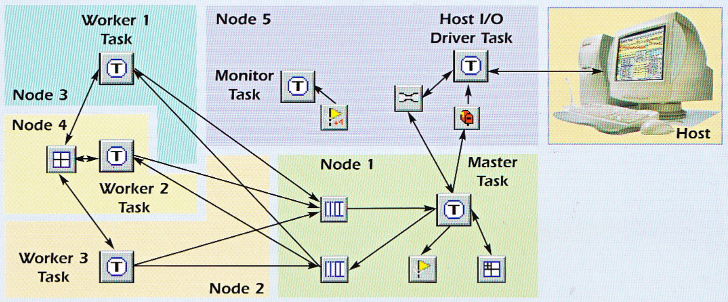
Virtual Single Processor model – Nanokernel+microkernel

Multitasking for efficiency and modularity
Today's DSP applications rely increasingly on complex interactions between individual functions Where computations take place and how the results are communicated have become key issues. Multitasking is the natural way to map such applications into programs.
For example, when executing multiple functions on multiple datastreams, multitasking gets more out of the hardware by prioritizing execution and by overlapping communication with computation. This means no cycles are wasted waiting for events to happen.
Task-centered programs are also easier to modularize, allowing several developers to work on the same project with clearly defined interfaces.

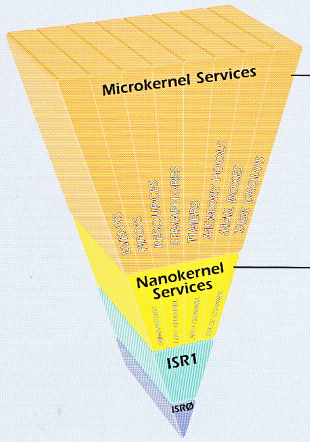




Distributed semantics for true scalability
When using the multitasking core of Virtuoso VSP, the logical blocks of the application are mapped onto one or more tasks. Tasks are system objects, and just like semaphores, fifos, mailboxes and other system objects, they can be distributed over the available processors using the Virtuoso Project Manager. Moving any system object from one to another processor is done by a few clicks of the mouse. This is VSP and distributed semantics in action. It means there is no need for an application programmer to know everything about the hardware configuration before starting to write code.
And once written, the code is portable between different configurations and between different processors and boards. From the software side, adding or removing processors is reduced to a simple remapping of objects in the project file.



Virtuoso's layered architecture
Virtuoso uses a unique layered architecture optimized for signal processing systems. The main layers, the Application Kernel and the System Kernel, work harmoniously together to maximize the performance of each individual part of the system. Virtuoso's ISR layers allow nesting of interrupts, even on DSPs that do not have the support in hardware.



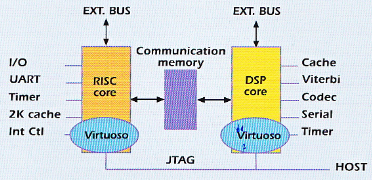
Microkernel for application level

The application level features a fully pre-emptive scheduler that matches the performance of even the most powerful DSPs. Yet the Application Kernel services have been kept lean and clean to minimize the learning curve and to reduce the development effort.

Nanokernel for system level

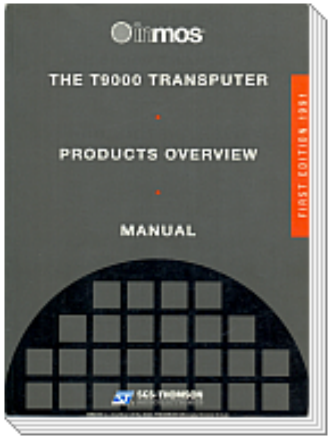
The nanokernel level is provided specifically for developing low latency peripheral drivers and to obtain a low overhead at the system level. Using lightweight processes, it offers extremely fast context switching and very small code size from a prioritized, non-preemptive scheduler.
It has a direct interface to the ISR levels and channel based application level communication and synchronization services.

Virtuoso™ for specialized applications
Virtuoso is used in systems as diverse as space, defense and commercial products. In these applications, Virtuoso's SoftStealth™ technology can generate a runtime system of as small as 2-3 KWords, code size that only careful hand coding can obtain.
In other, more specialized systems, Virtuoso has been tailored with hardware specific extensions. This considerably reduces the risks and development costs of a dedicated RTOS.
For the next-generation asynchronous ASICs, hardware and software co-design will be a must. Starting from a multi-tasking software point of view is the only natural way to go. With Virtuoso you can start today.



And then came the T9000 (not)

- T2, T4, T8 were brilliant designs (20 – 35 MHz)
- But semicon is about “faster and faster”
- T9000 was over-designed:
  - Too complex for semicon technology (to run at the specified speed)
  - Design tools could not handle it



- No bridging version with e.g. 100 MHz T8
- Marketing mistakes: super computing vs. embedded, alienation third parties (IQ module.)

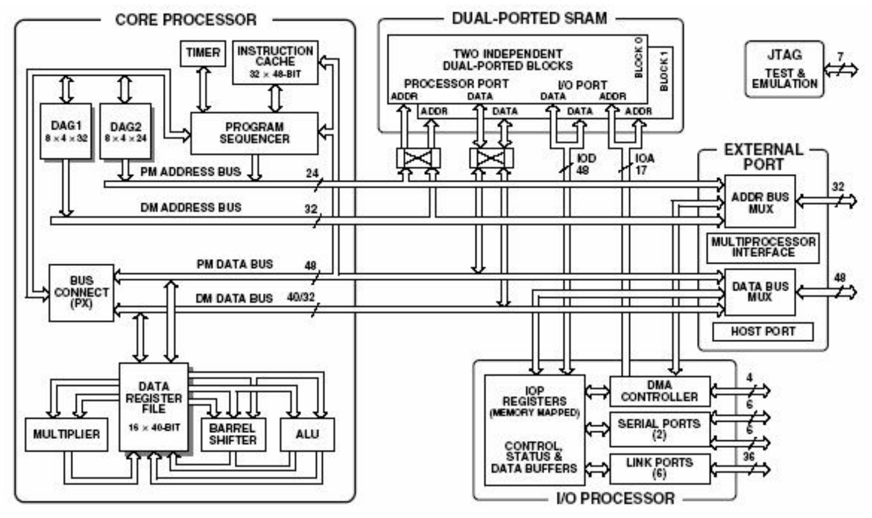
Along came TI with the C40

- 40 MHz DSP with links and DMAs
- Harder to program:
  - Heavy context
  - Too many HW options
  - Silicon bugs

- New RTOS architecture required (latency!)
  - Nano-kernel for driver layer (in asm)
  - Micro-kernel for application layer (in C + asm)
- Took about a full year

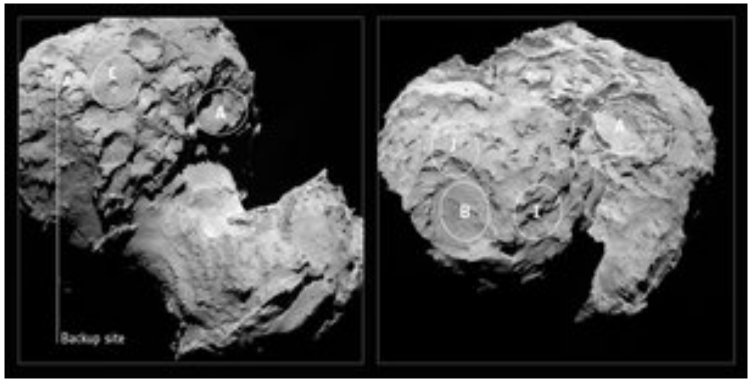
ADI did as well with a SHARC

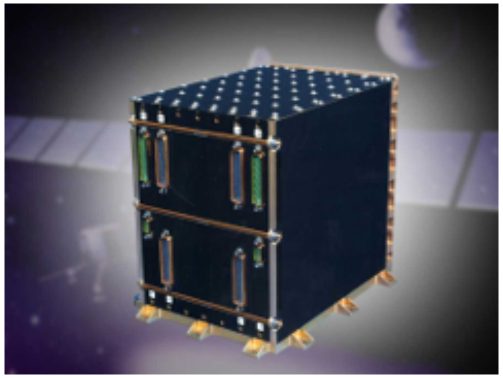
- A DSP with 4 links, DMAs and 128 KB memory
- Even more complex than C40
- But very, very fast (at the time)
- Used in sonar with 1600 DSPs



Going into space

- 21020 RT: radiation hardened DSPs (20 MHz)
  - With SMCS SpaceWire (=T9000) links
- Boards developed by EADS
- Used in multiple scientific missions
  - Giotto
  - Rosetta: landing on a comet in 2014





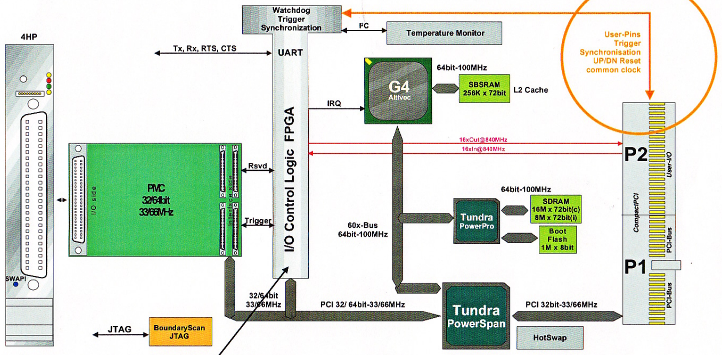
And the TigerSHARC (came too late)

- Design review in 1996, first chips in 2001
- Similar issues as T9000 saga
- We developed the ATLAS DSP computer:
  - SHARC
  - TI C6000
  - Freescale PowerPC + FPGA + MMU + PCI Contrl:
    - It became very complex
    - General purpose processors benefit from volume
    - Less real-time (cache dependency)

Atlas DSP computer with “links”

Eonic

Atlas3-G4 Block Diagram



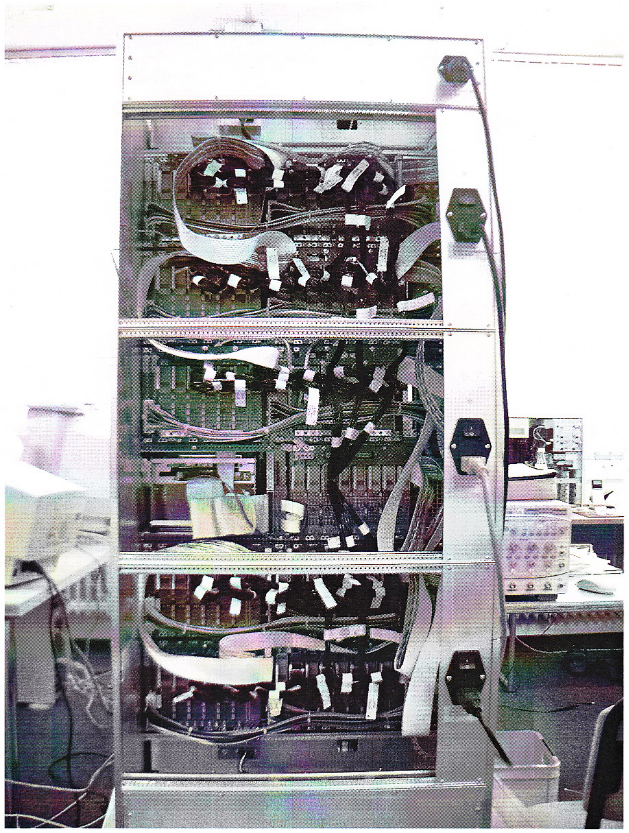
Software-driven Hardware

The ALTERA FPGA: LINK-DMA, clock, trigger, sync., algorithms, ...

May-2001

12

Eonic CONFIDENTIAL



Wind River takes it all

- Acquired the technology in 2001
- Gracefully killed it 3 years later (sic)
- What's next ?
  - After the sabbatical
  - After being interim CEO
  - After the Tundra disaster:
    - Bleeding edge vs. leading edge

Trustworthy Systems Engineering

- Confusion: what do people really want / need ?
- Mixing up problem domain with a known solution
- Thinking about what can go wrong
  - Law of Murphy always applies
- Language: semantics
- Nobody sees everything, but thinks his view is the dominant one
- Productivity!
  - Time to safety >= time to market: conflict?
- Software can be error-free, hardware never is

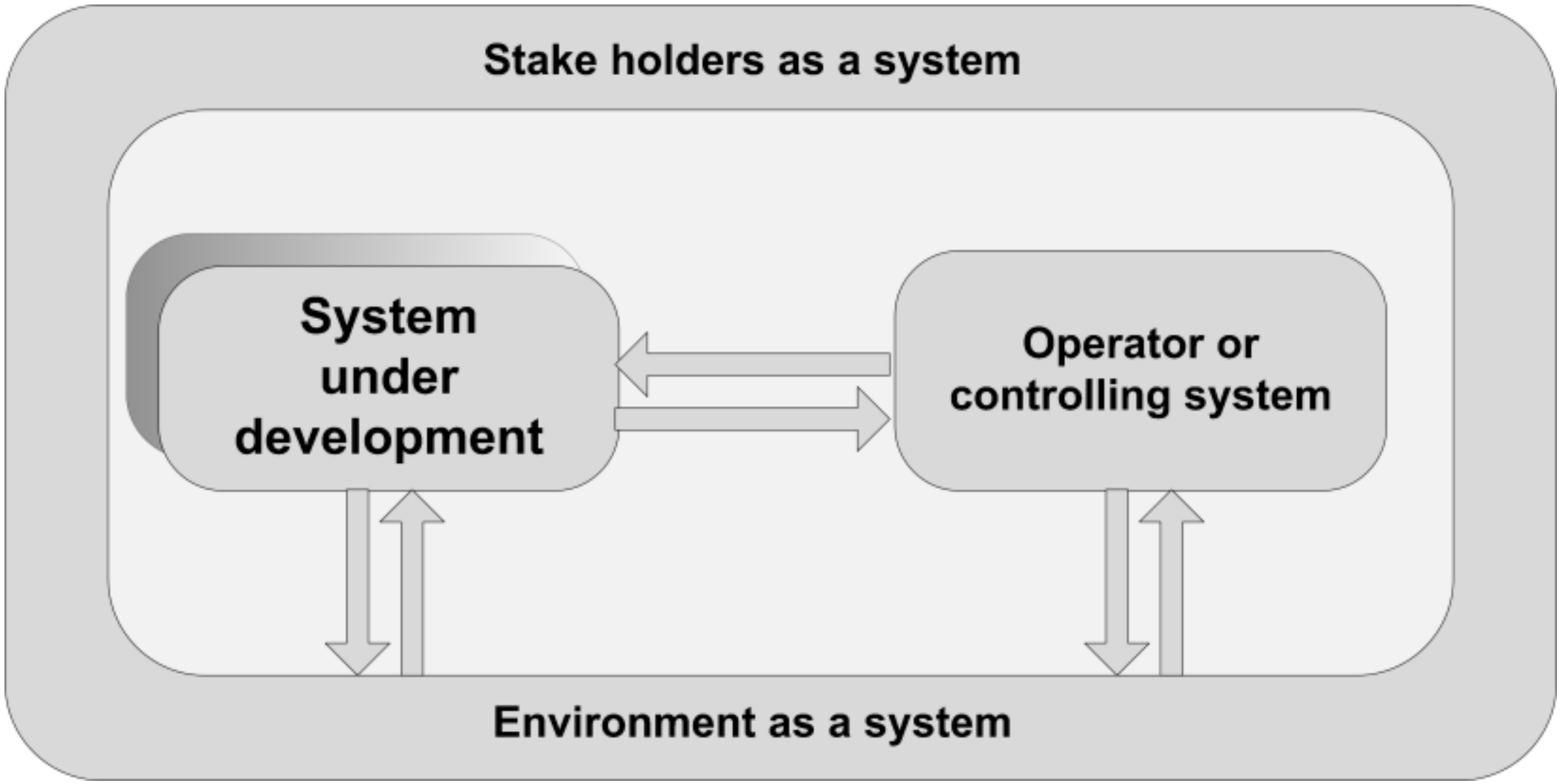
Meta-modeling vs. modeling

- Model-driven engineering
  - Model then implement
- Is UML modeling?
  - Or a description? How precise is UML?
- Modeling means abstraction! (> 1 level)
- Models need to be univoque!
- If a system is an implemented model, how do we describe its properties and architecture in a univoque way?

Key paradigms

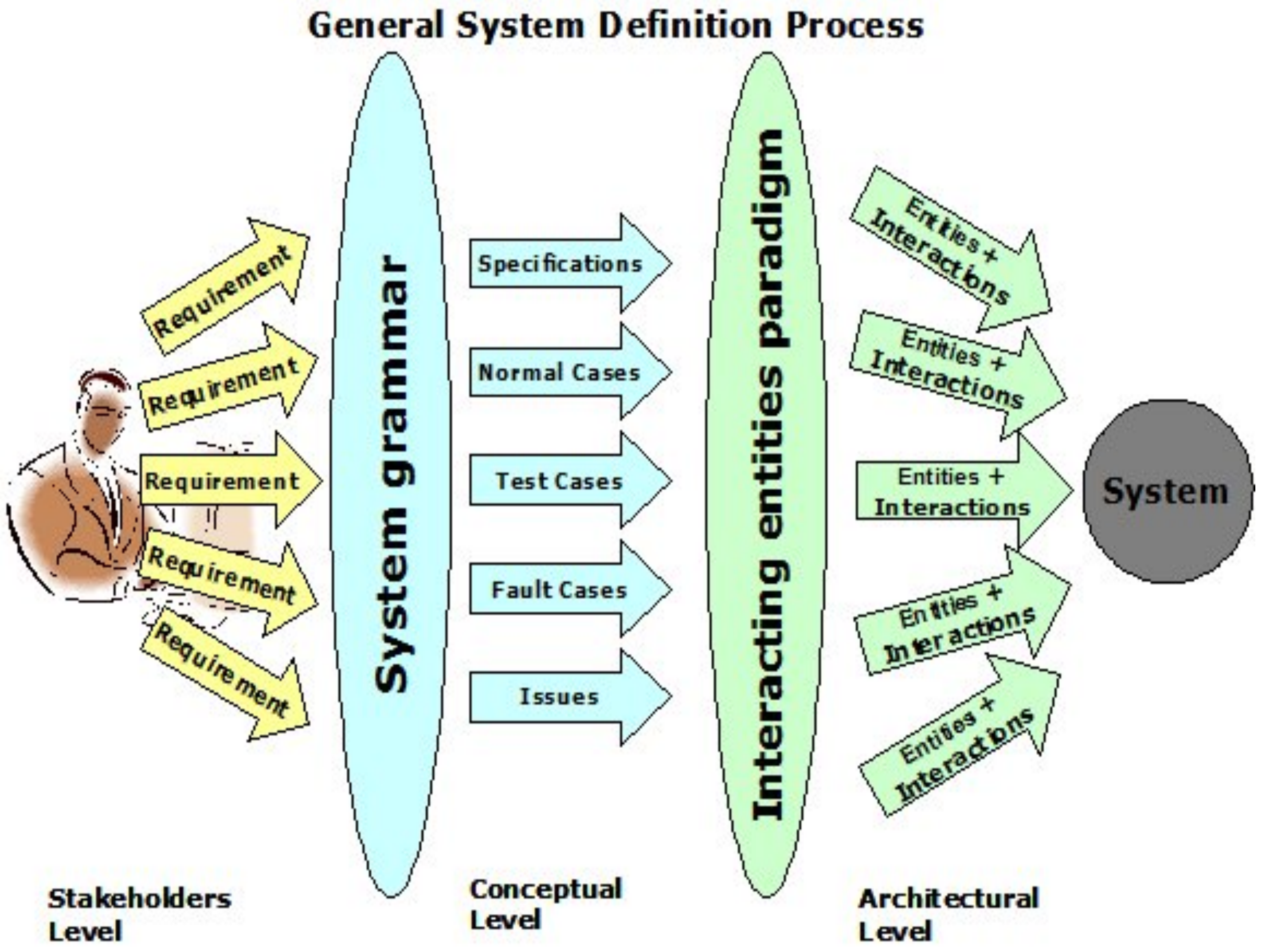
- Issue: all (point) tools and methodologies speak a different language
- One needs multiple tools to build a single system (for productivity!)
- N tools = NxN translators, but possible?
- Key paradigm: **Unified Semantics / Methodology**
  - **Speak same language everywhere**
- Key paradigm: **Interacting Entities**
  - **Meta-level description of architecture**

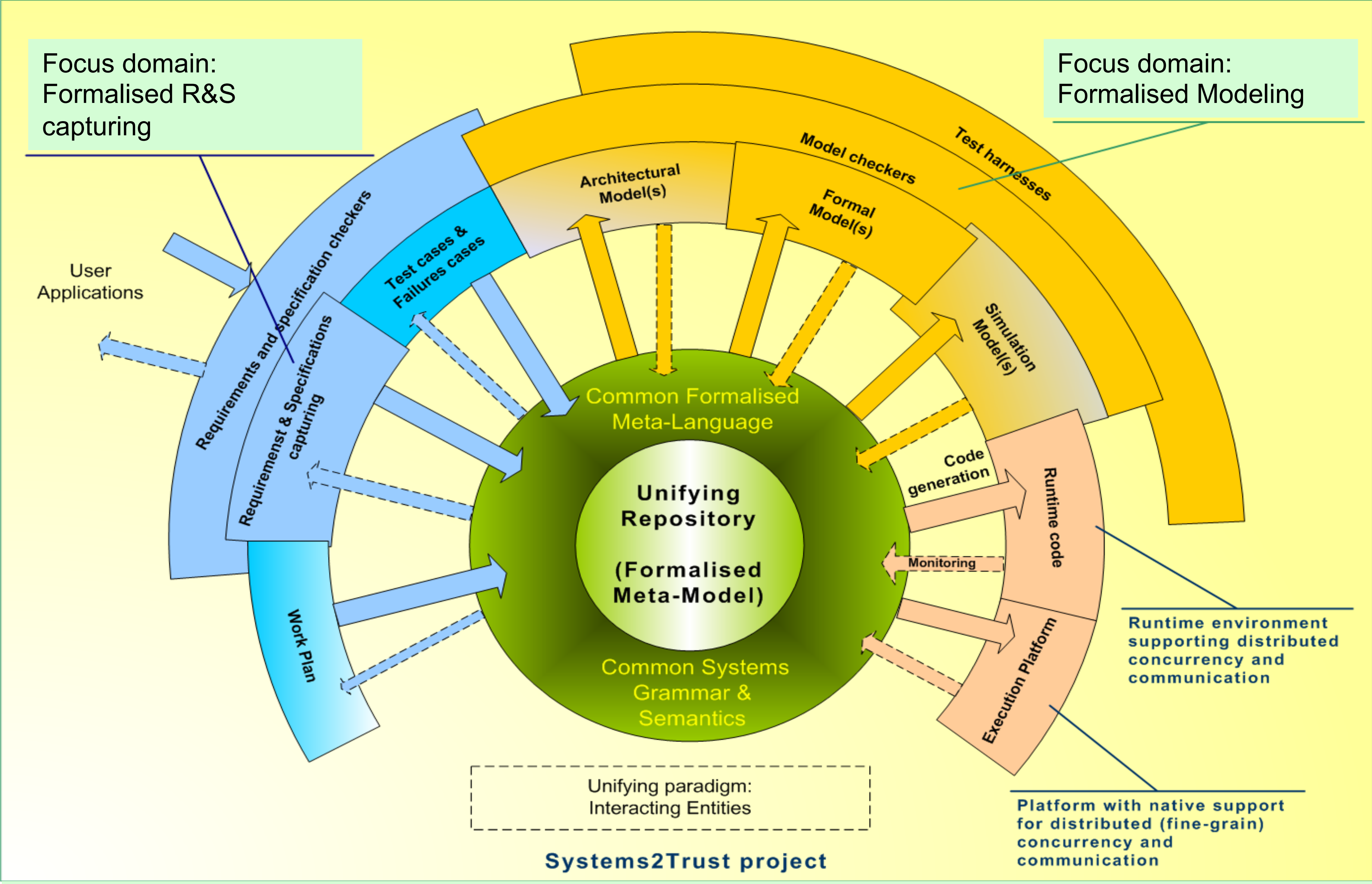
A system is never alone



See workhop on hybrid systems

Human factors: from idea to reality




Focus domain:
Formalised R&S capturing
Focus domain:
Formalised Modeling
User Applications
Requirements and specification checkers
Requiremenst & 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
Runtime environment supporting distributed concurrency and communication
Platform with native support for distributed (fine-grain) concurrency and communication
Unifying paradigm: Interacting Entities
Systems2Trust project

Unified Systems Engineering supported by GoedelWorks

Formal methods?

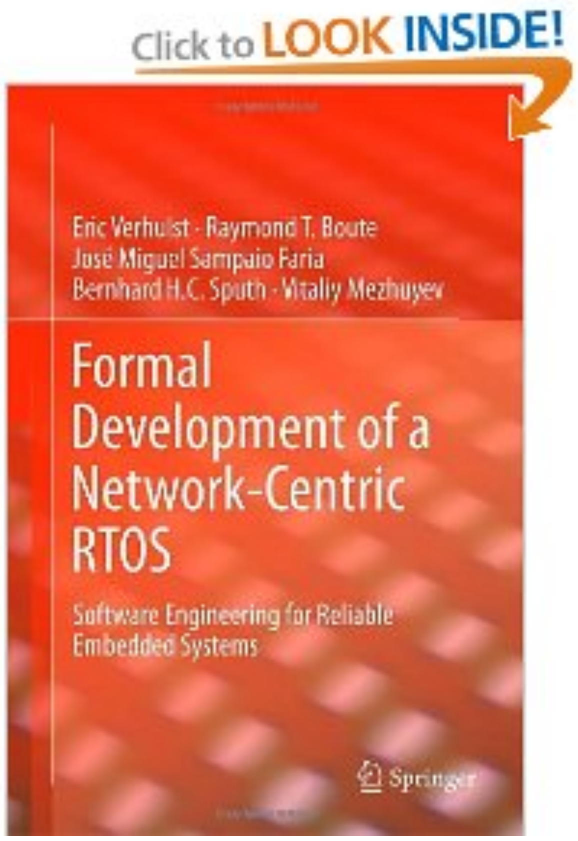
- Can formal techniques help in getting the system / software right?
- Formal = proven = correct?
- OpenComRTOS project:
  - Redevelop Virtuoso-like RTOS from scratch
  - Use formal techniques
  - Use of TLA+/TLC and a bit of UPPAAL

Results

- Best use of formal techniques is not to verify and proof, certainly not source code
  - Change one line and one can start all over
- Best use: tool for abstract ~~modeling~~ thinking
  - Thinking about “correct” / “best” behavior
  - It helps the mind to undo brainwashing experiences (carpenter’s problem: nail+hammer) & syndromes
  - Separation of concerns:
    - Clean and orthogonal architecture
- Code size dropped with a factor 10 (=> 5 to 15 KB)

From FM to an environment

- Many goals:
  - Portability
  - OpenComRTOS kernel
  - Visual Designer
  - Event Tracer
  - System Inspector
  - Safe Virtual Machine
- Springer book:

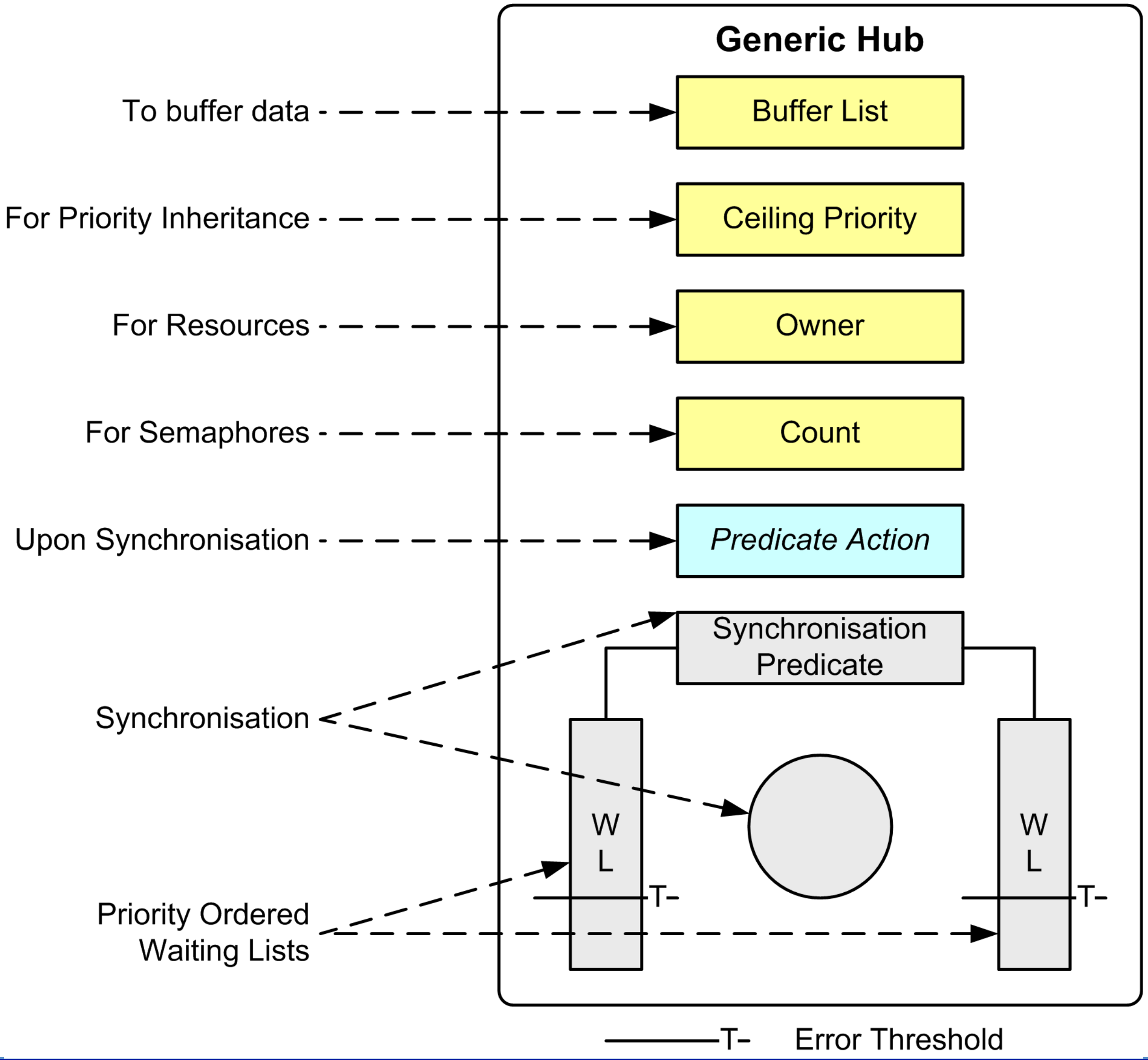

Click to LOOK INSIDE!
Eric Verhulst · Raymond T. Boute
José Miguel Sampaio Faria
Bernhard H.C. Sputh · Vitaliy Mezhuyev
Formal Development of a Network-Centric RTOS
Software Engineering for Reliable Embedded Systems
Springer

Formal Development of a Network-Centric RTOS, *Software Engineering for Reliable Embedded Systems, Verhulst, E., Boute, R.T., Faria, J.M.S., SpuTh, B.H.C., Mezhuyev, V*

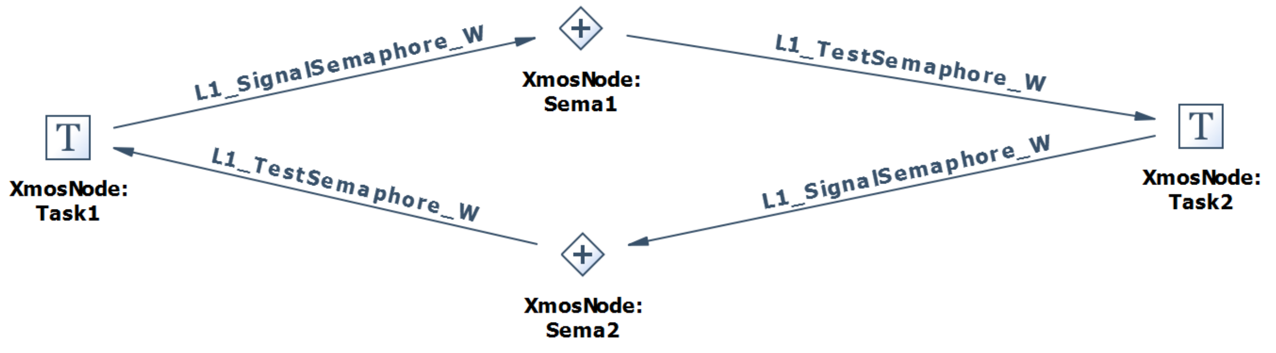
The Hub: this is not a channel

- Occam has channels, our RTOS has Hubs
  - "pragmatic superset of CSP"
- Hub = channel + behavior
- Hub = synchronisation predicate (guard) + action = guarded atomic action
- Decouples Tasks (processors) fully
- Shared functionality saves a lot of code and errors! Lower certification efforts!
- Makes it easy to add new types of Hubs:
  - Event, Semaphore, FIFO, Resource, BlackBoard, …

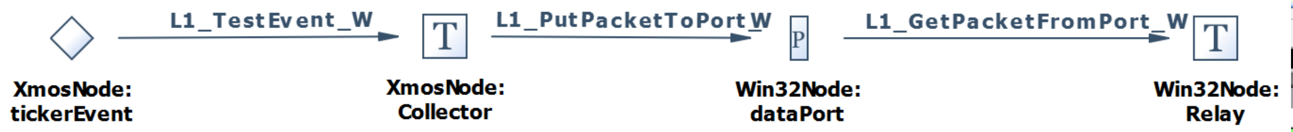
Generic Hub model

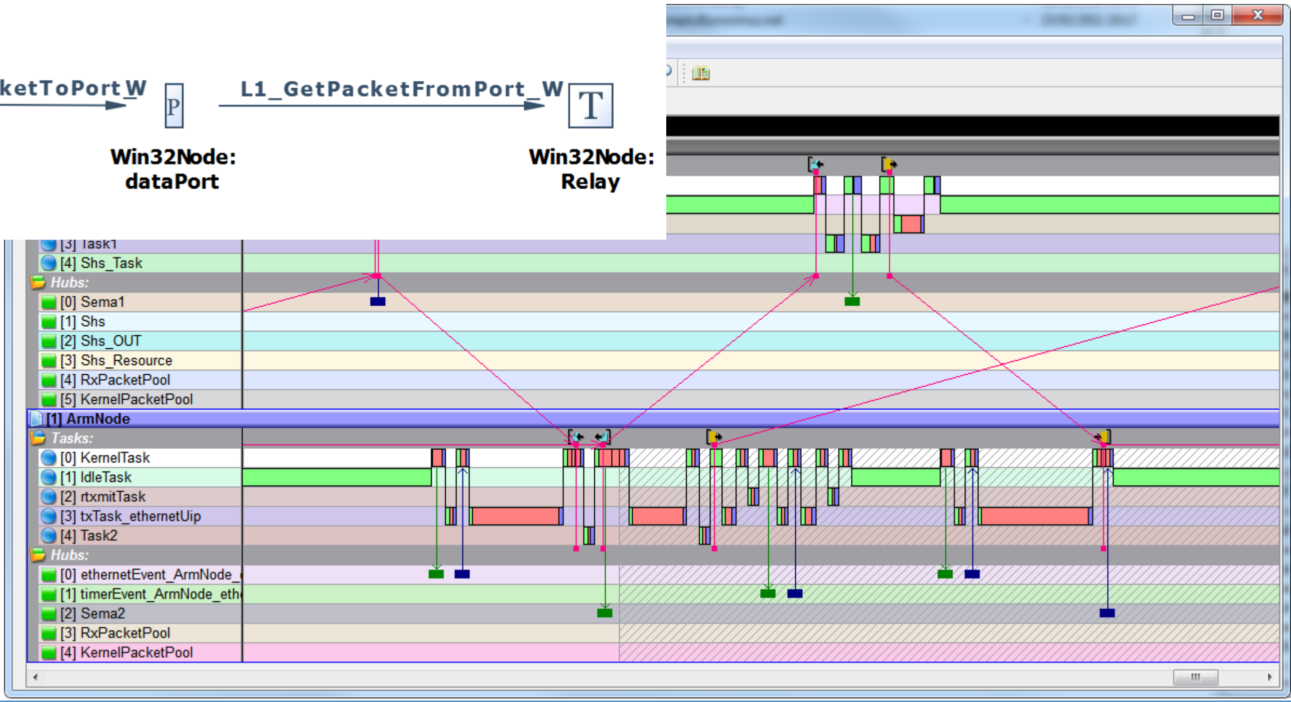


Visual programming/modeling



Topology independent:
Separate logical behavior
From temporal behavior





Then came ARRL (0 to 7):

Assured Reliability and Resilience Level

- Criterion defines properties of components/ systems taking into account fault behavior
- Leads to the notion of anti-fragile systems

ARRL 0	It might work (use as is)
ARRL 3	ARRL 2 + goes to fail-safe or reduced operational mode upon fault (requires monitoring + redundancy) – fault behavior is predictable as well as next state
ARRL 7	The component (subsystem) is part of a **system of systems** and a **process is in place** that includes continuous monitoring and improvement supervised by an **independent regulatory body**

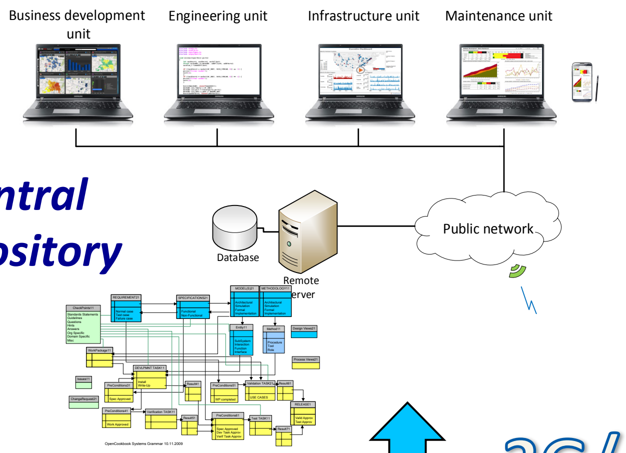
VirtuosoNext: ARRL-3 at work

- Static programming is safer but:
  - Dynamic programming is norm: software always has errors
  - Hardware is almost perfect but will fail
  - External factors compromise the QoS
  - SoCs: very powerful, complex, single point of failure
- Need to physically isolate program segments from each other (error propagation)
- Fine grain space and time partitioning

KURT: Applying it all

GoedelWorks

Central Repository



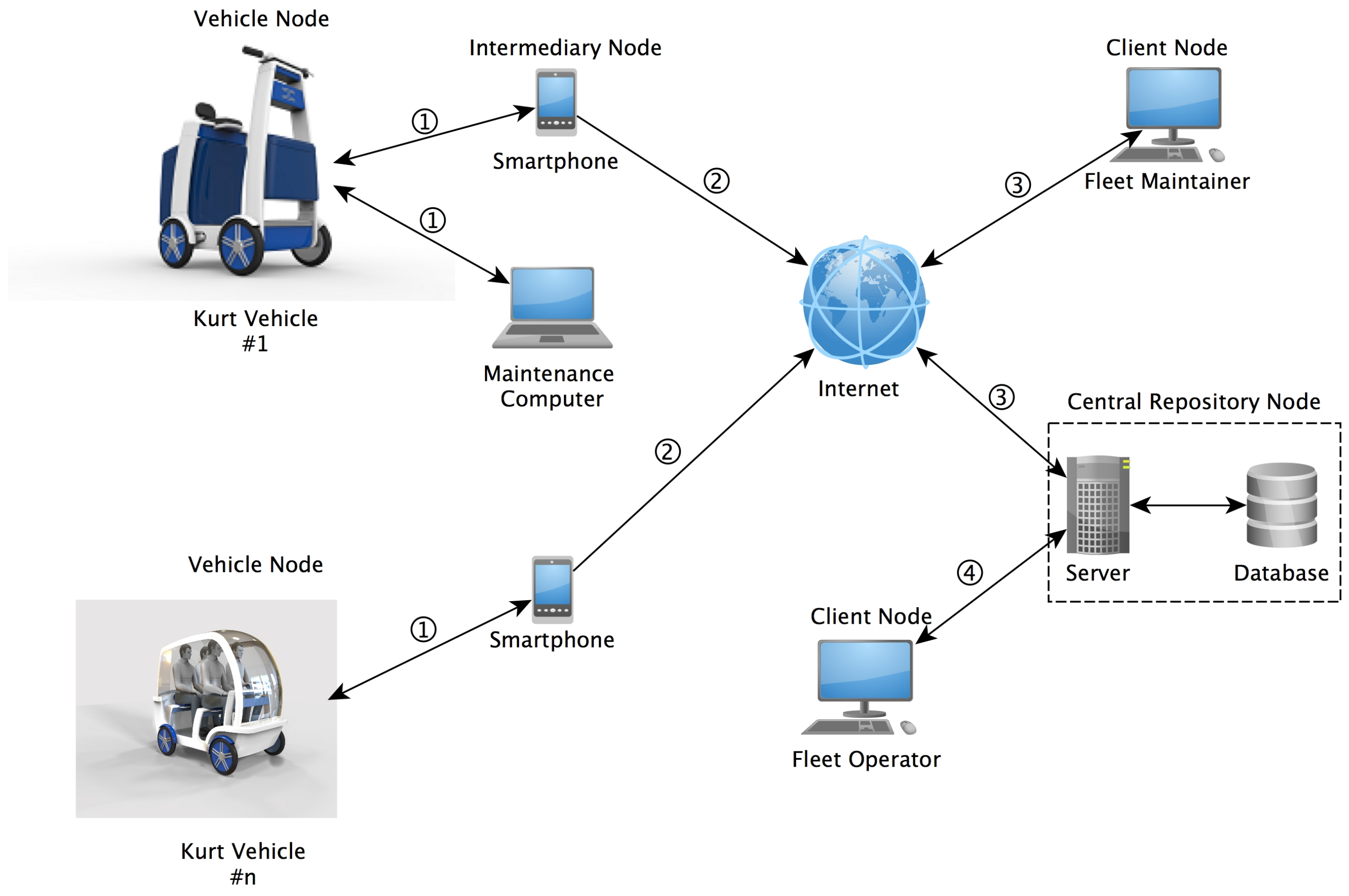
3G/4G

Visual Designer
VirtuosoNext
ARM Mx/Rx, A9 with FPGA

Operational data

Drive and Steer by wire 4x4

Life-cycle and fleet management



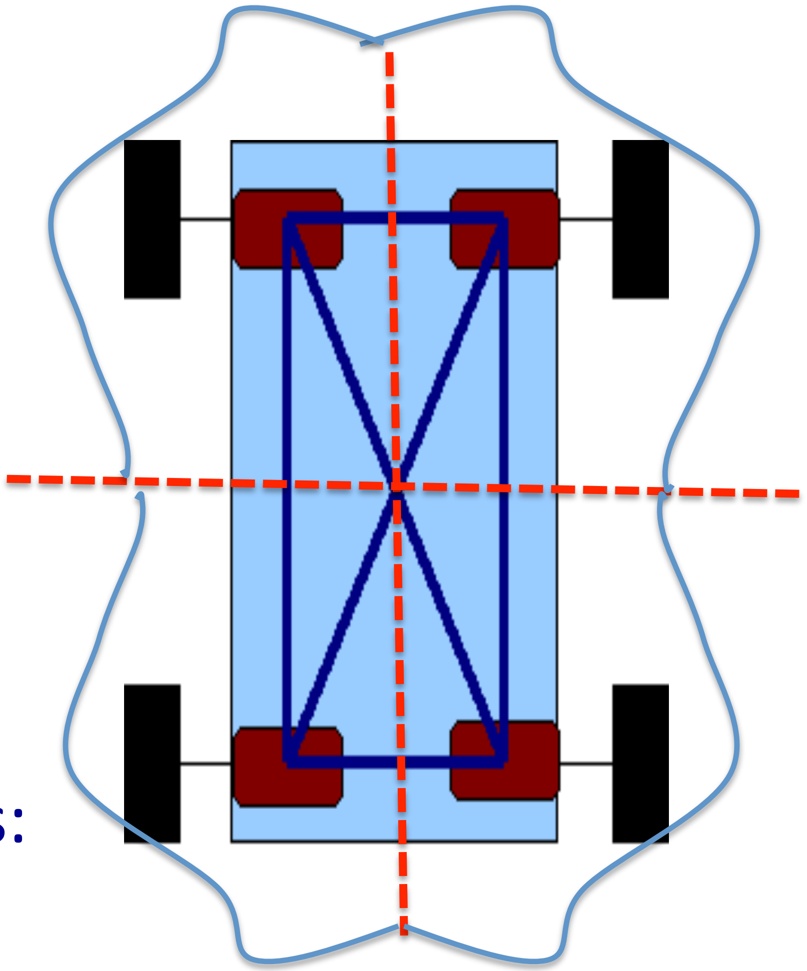
① Wireless Personal Area Network
② Mobile Wireless Broadband Network
③ Wide Area Network
④ Local Area Network

Life-cycle management links operational data with engineering data for:

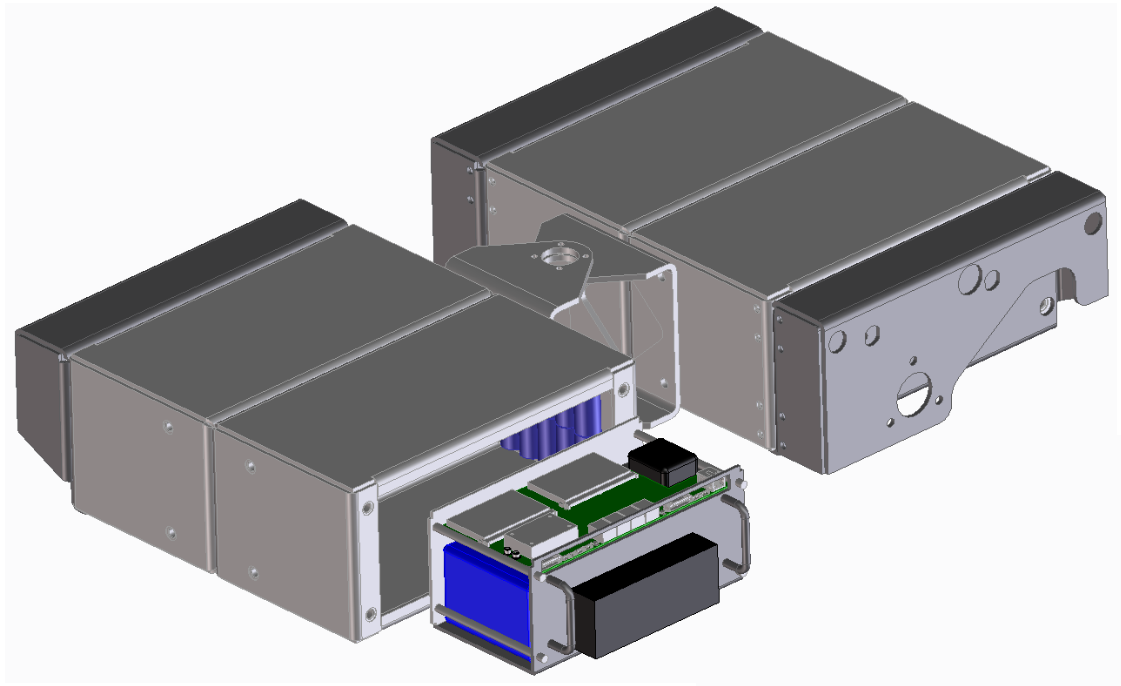
- **Continuous monitoring and qualification**
- **Preventive maintenance**
- **Fleet management**

Novel Modular and Redundant architecture

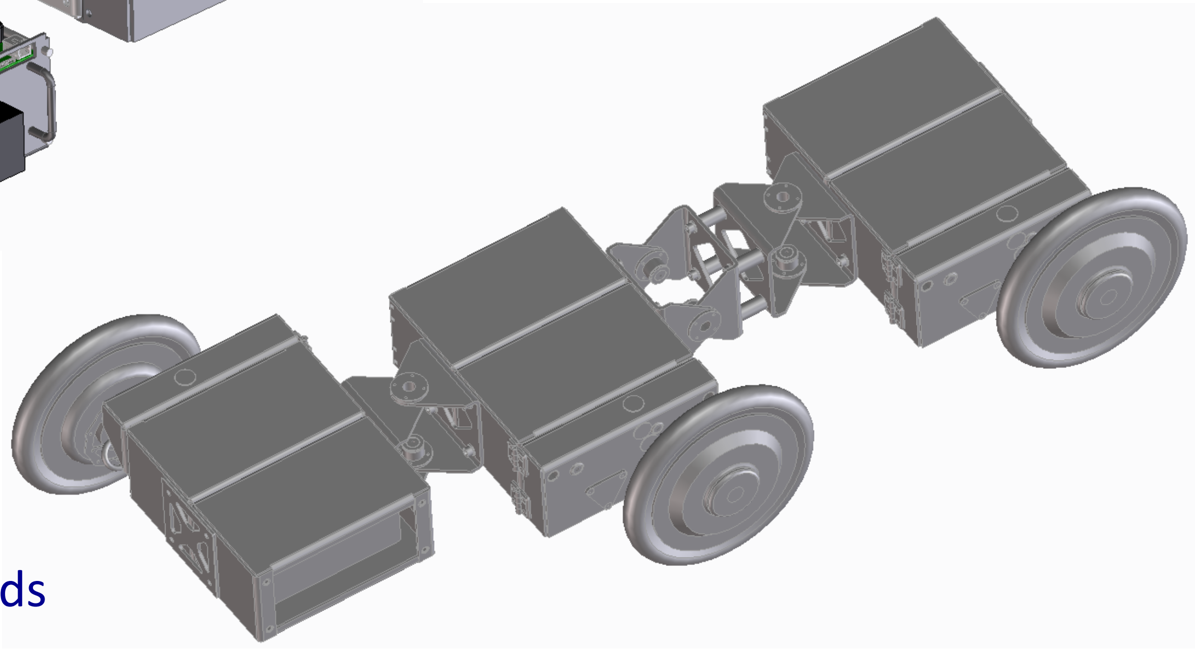
- Patent pending
- Combine reusable units
  - Economy of scale (COG!)
  - Redundancy (fault tolerant)
- Propulsion Unit =
  - Battery (LiOn, H2 FC) + motor
  - Suspension + wheels
  - Vectoring steer/drive by wire/4x4
- Smart environmental awareness:
  - Obstacle detection/avoidance
  - Assisted auto-navigation



Unique modular and scalable platform



- **Professional "mecano"**
- Propulsion platform independent from super-structure



Interacting Entities in the mechanical domain:

- Easy to assemble
- Easy to repair
- Easy to adapt to different needs

Some Lessons, some principles

- "A concept is a difficult concept" (Wittgenstein)
  - Semantics matter (also for engineers)
- KISS: Keep It Simple but Smart
  - A complex solution is a problem not well understood
  - "Too many words" is a sure sign
  - Keep it simple but never too simple (A. Einstein)
- Separation of concerns (orthogonality)
- "Fear is a mind killer" (Frank Herbert, Dune)
- "Nothing is so difficult as not deceiving oneself" (Wittgenstein)

Last slide: Next challenges:

- **Hybrid systems:** how do we formally verify them?
  - See workshop
- **Safe software by design:** Goedelworks + VirtuosoNext + SPARK/Ada?
- **Systems of Systems:** life-time evolving complexity
- **Life time QoS:** how to build 100 yrs electronics?

Thanks for your attention

Questions? Yes, we recruit CSPers

eric.verhulst (@) altreonic.com

Team: small is beautiful