The flying train

Was it IEC - 61508 Safety Certified?

First winter snow has stopped the eurostar high speed train running for 3 days. It couldn't cope with the temperature difference between the warm tunnel and the frigid air. The high speed train between Paris and Amsterdam was stopped on its tracks because the safety system overreacted. The fail-safe mechanism forced the driver to reboot the engine while passengers were kept waiting for three hours. A spokesperson said that even extensive testing had never found these issues. They also experienced problems with the new communication system.

Electronics and software are increasingly replacing and enhancing mechanical solutions. The issue is the state space explosion.

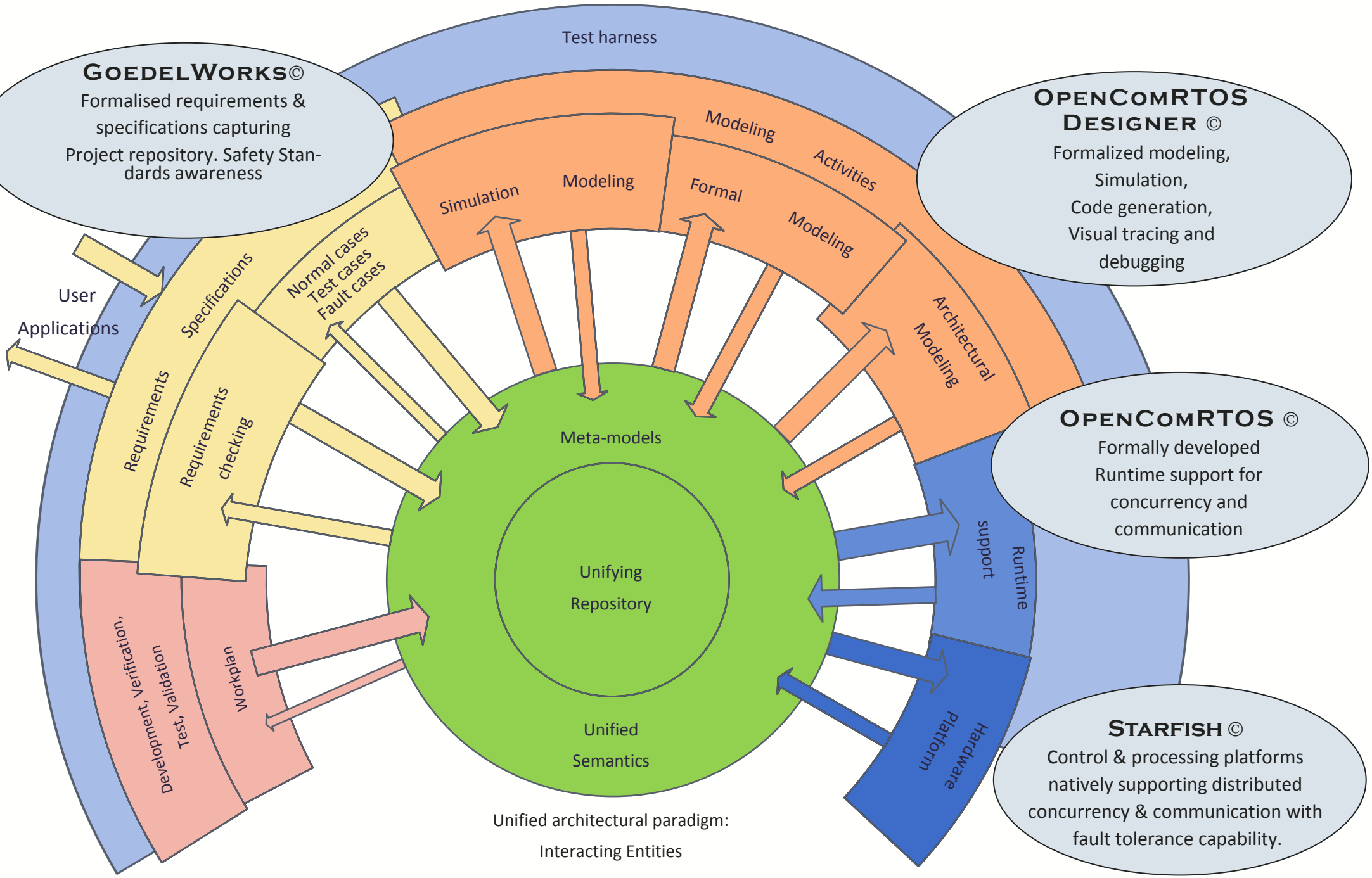
Contrary to mechanical counterparts, software knows no graceful degradation. How can engineering tackle this challenge?

From Deep Space to Deep Sea

Altreonic

www. Altreonic.com

A coherent approach to systems and safety engineering



Unified architectural paradigm:

Interacting Entities

Elegance & Engineering

Scalable Smart Systems

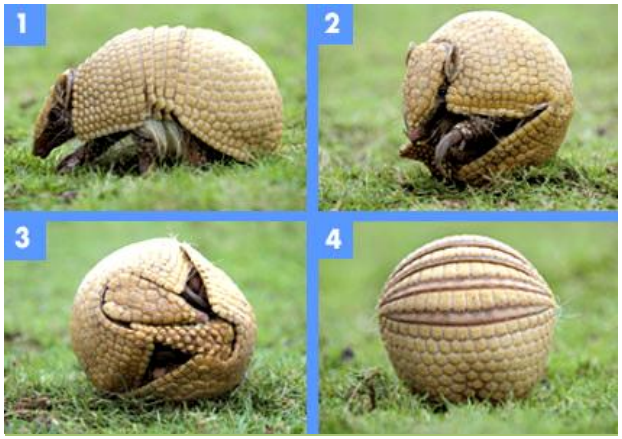
Trustworthy Software

The Future Has Arrived

An iterative V-Cycle

Systems and software engineering is a multi-domain activity. The key however is the human factor. People from different domains speak different languages, even if they mean the same thing. This leads to misunderstanding and is one of the prime reasons why projects are late or fail. Altreonic's "**Unified Semantics**" approach recognizes this from the beginning, reducing the need for continuous translations between the domains.

Engineering projects can be challenging because of the complexity involved. The system can have many composing sub-systems that must fit and work together. The same applies to the engineering organization. Many skills come together and must continuously communicate and coordinate to reach the goals. To manage this challenge, decomposition and separation of concerns helps.



Altreonic's "**Interacting Entities**" paradigm is a common sense but universal approach for dealing with it. Given the complexity of systems and software engineering, a systematic approach is needed to reach a successful delivery of a system. Determining what is the right system is as important as developing it right. Reducing the complexity means reducing it up front, which is also more cost-efficient.

Keep it Simple but Smart.

Nevertheless, further stages will always discover issues that were overlooked or that will provide new insights. Hence we advocate a **fine-grain iterative V-model**. Requirements only become final when the system is released to production. Using a modular architecture and using dependency graphs, it also becomes more cost-efficient to develop and certify a family of products.

What means trustworthy?

Trust in engineering is necessary because ultimately the systems and products developed are to be used by humans. Trustworthy can be decomposed in four components:

Safety— This is the traditional property that is associated with high quality and reliability. However, high quality is not enough to achieve safety. Hence Altreonic decomposes use cases in three type: normal cases, test cases and fault cases. Any systems must be thought out from the beginning to satisfy all of them. If not, in the worst case human lives can be at stake.

Security— While traditionally associated with the IT sector, embedded devices and systems are now everywhere. Whereas safety is dealing with inherent risks and hazards, security deals with maliciously introduced faults. Also fellow human beings can be a risk factor.

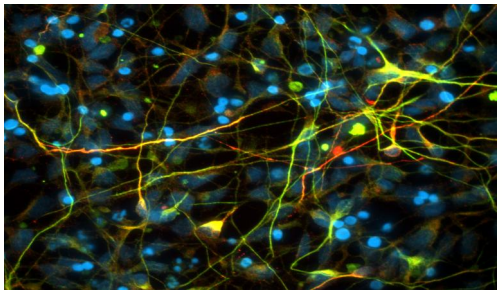
Privacy— While automating our lives, embedded systems increasingly need to register personal data that people provide because they trust the system. Again, malicious misuse of this data must be prevented and is a risk factor that must be taken into account.

Usability— The more we interact with embedded devices, the more we presume they will work and interact as we expect, certainly when we need them in a critical situation. This is a domain where human and machine become part of a larger system and hence speaking the same "semantics" is crucial for acceptance of the new technology.

""Concept" is a difficult concept". L. Wittgenstein

GoedelWorks ©

Developed as a multi-user web portal, GoedelWorks integrates Altreonic's methodology using a straightforward systems engineering information model, combining the process as well as the project view. It can support a project from early conception till release of the product or system. Safety Standards awareness (IEC-61508, IEC-62061, ISO-26262, ISO-13849, ISO-25119 and ISO-15998) supports pre-certification.



R&D background

ASIL, with Flanders Drive Automotive cluster on a common Safety Integrity Level engineering (IWT).

EVOLVE, on Evolutionary Verification and Certification of safety critical systems (ITEA).

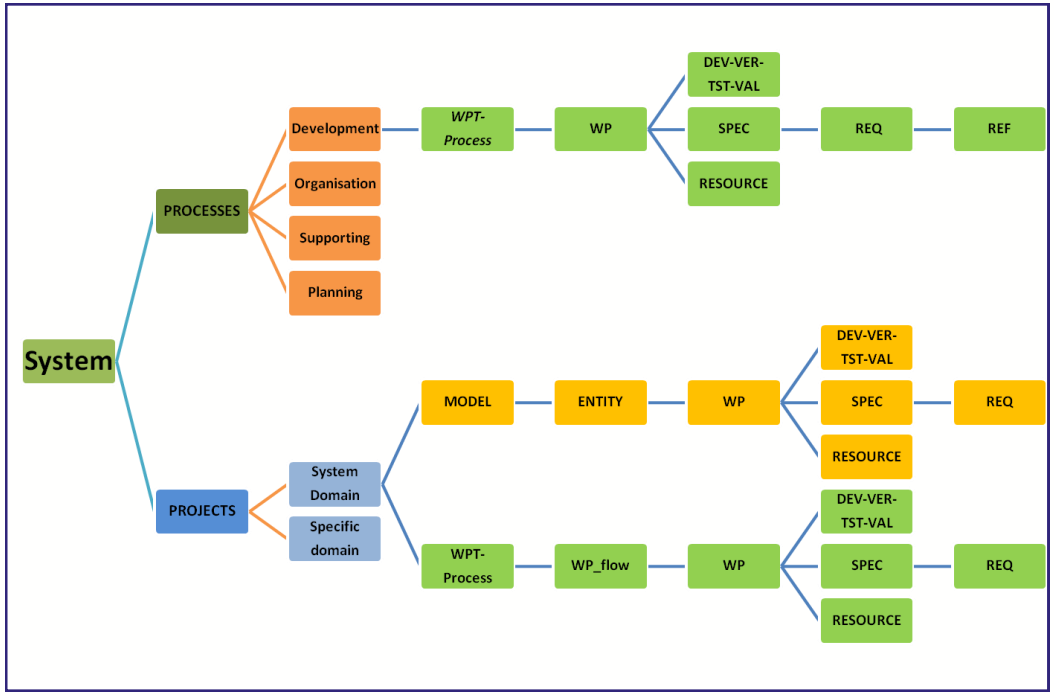
OPENCOSS, on developing a common certification methodology for Automotive, Railway and Aerospace. (FP7)

Tools, dependency trees and version management for productivity and consistency

While **Unified Semantics** and **Interacting Entities** bring you a long way in mastering complex design, in the project domain teamwork, discipline and consistency are key. Hence our tools support the methodology to boost productivity and automate whenever possible. Hence, requirements and specifications capturing comes down to incrementally building up a structured web portal, that becomes a living repository for the project.

In order to keep track and support reuse, version management and dependency trees are essential. Version and configuration management allows to have consistent snap-shots of the system. Dependency and precedence trees allow to analyze the impact of changes, as small as they can be.

Safety means that failures are anticipated by backtracking them to the original root cause. In an engineering process, the reverse happens as well. Any change in the input is reflected in the system's behavior.



Formal and formalized modeling & verification

In need of verifying C code or verifying numerical stability of algorithms?

Altreonic has experience and know-how of different formal tools and methods, such as TLA+/ TLC, CSP, UPPAAL and B. We also use tools that formally verify C code.

Only software can be error-free.

At Altreonic we don't speak of software bugs. Even faulty software will fail quite predictably as the program is a very deterministic state machine. Software however has errors, either by specification, either by design. All other faults are externally induced.

Architecture Matters

While there many solutions for a given problem, often they create more complexity than needed. This comes from the fact that the complexity reflects the learning process. In the beginning, the information looks confused and overlapping and not well structured.

Gradually, order will come forward as insight will have been gained. This comes from analyzing how it really works and taking a step back. In engineering we call this formalization. What it comes down to is building abstract models that help in getting rid of the confusing details. Complexity is a sign of a problem not well understood.

Architecture is the forgotten art of engineering. Clean architectures make a difference.

StarFish © Fault Resilient computing

There was a time when mechanical solutions were dominant. Bulky and heavy, but mostly inherently safe because operating in the continuous domain, they had the benefit of graceful degradation.

Now digital electronics and software dominates because it makes things smarter, more flexible, cheaper and lighter. The issue is that we are here in the digital domain and every clock pulse (often billions per second) a single glitch can make things go wrong.

The solution again is the architecture and concurrency. Concurrency provides more performance but also redundancy when needed.

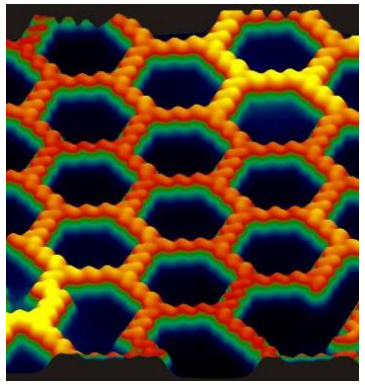
Such a safe architecture cannot be put as a layer on top of an unsafe one. The underlying basis must be correct be design. Therefore OpenComRTOS was formally developed as well as formally proven.

Less code means also lower probability of error. Scalability by design also means that transparent parallel processing makes distributing the work for redundancy is built in. No clumsy middleware to deal with gives better performance.

Safe Systems = SIL4

Making systems safe by design is not easy. It requires thinking up front about behavior that should never happen. Even when done properly, there is still the so-called common mode failure. It symbolizes that the unthinkable can happen and that it will happen is a certainty.

Therefore we disagree with safety thinking that goes for so-called fail-safe states when a fault is detected. An impaired system that is no longer fully functional is no longer a safe system.

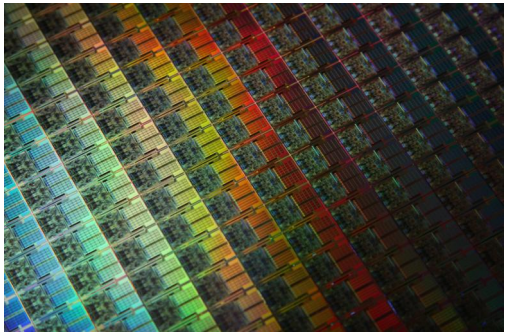


In safety terms systems should be designed for a SIL4 level , impaired by a fault but still fully functional in SIL3 mode.

OpenComRTOS ©

Formally developed and verified, OpenComRTOS is a network-centric, small but very powerful approach for transparent concurrent and distributed real-time embedded systems. Ideal for many-core SoCs.

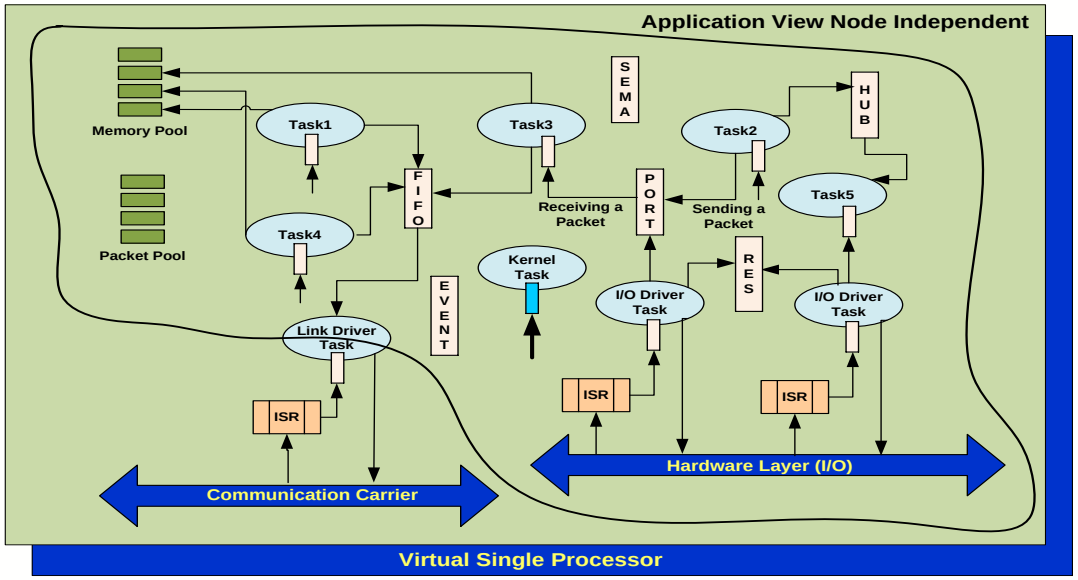
Design once, run everywhere has never been easier.

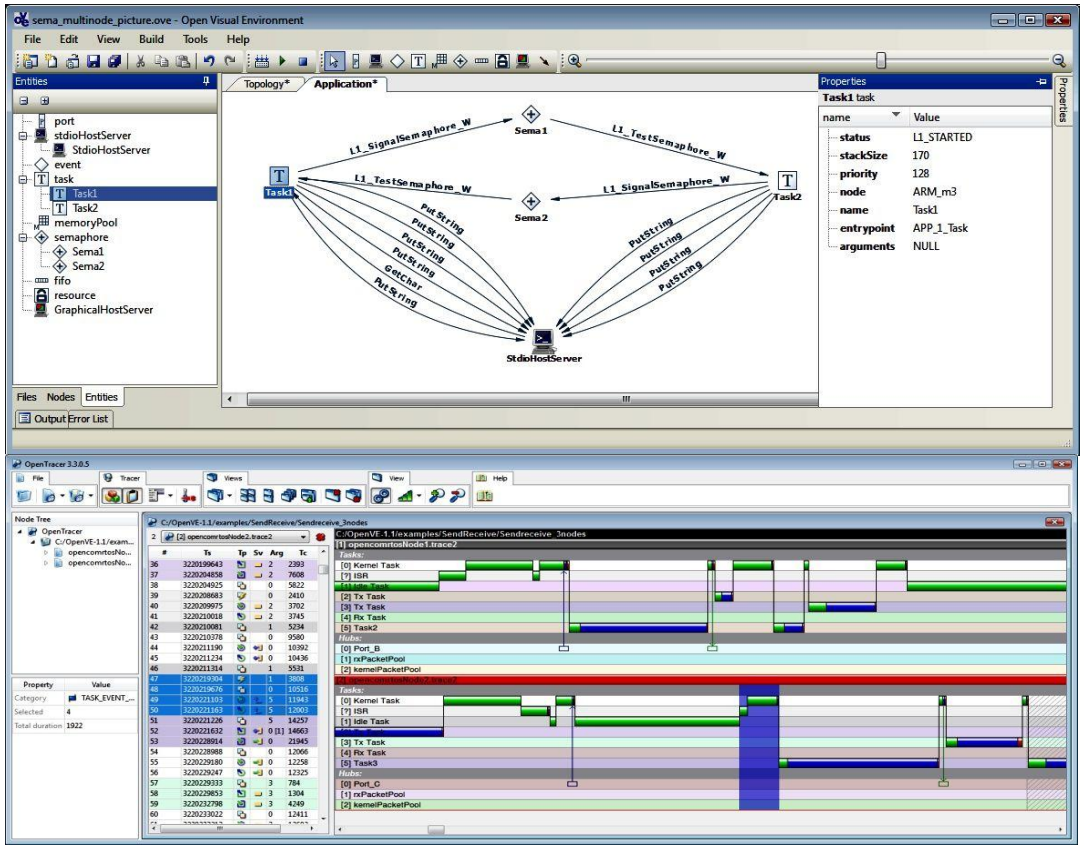


OpenTracer ©

Altreonic's OpenTracer is a powerful graphical tool for analyzing, verifying and profiling real-time embedded software.

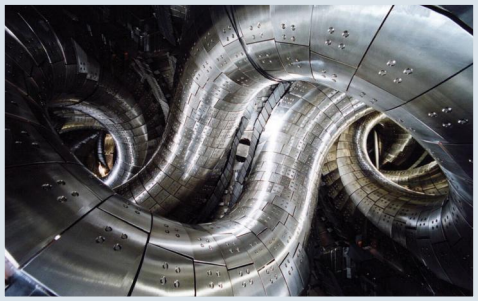
Integrated with Altreonic's OpenComRTOS Designer, OpenTracer is like an oscilloscope for the embedded engineer.





OpenComRTOS Designer ©

Alreonic's visual modeling environment. Combined with powerful code generators and simulation capability on Windows or Linux PCs, it allows the engineer to focus on the essence of his application and reduces considerably the time needed to develop embedded software.



Virtual Machine in 3 KB

Dynamic code in embedded distributed systems doesn't need to be big. Using a formalized approach we developed the capability to execute any binary code on any processor. Still the code is encapsulated as a real-time task and only requires a few KBytes of extra memory. This is what real embedded systems need.

Safety engineering standards

The first domain where engineering was applied is in the safety domain. Naturally, lives are at stake and failures can be catastrophic and expensive. Standards like ISO-26262, IEC61508, DO-178C have pioneered and make way for a systematic approach to systems engineering.

Why a formalized methodology is paying off

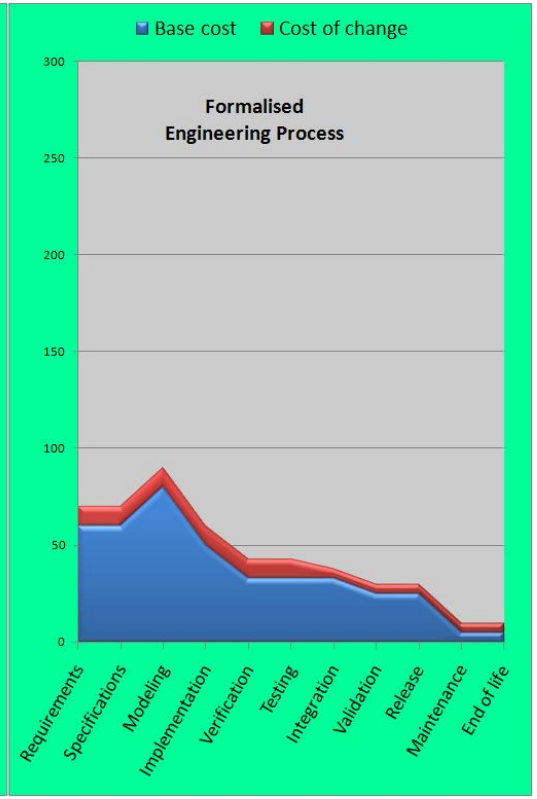
Traditional bottom-up development can sometimes gives a quick first result. This can be a good approach for quick prototyping, but it carries a high risk to use this approach for a production version. Incomplete and contradictory requirements will creep up in the architecture, resulting in issues during testing or worse in production. This shows up as a high risk in run-away costs as the cost of changes can skyrocket. A formalized approach on the other hand, will shift the shift the work upfront where making a change is often just a matter of thinking things through and making the changes in the specification or computer models. A such a high reliability driven design can easily have a lower lifecycle cost.

Engineering services

Altreonic puts is experience at work by providing its customers training, supporting tools and services. Especially for new projects, the transition to a formalized methodology can be daunting. However, once the concepts are understood, the complexity disappears to make room for a new insight.

Safety system properties

Safety can not be bolted onto a system like an afterthought. It is not provided by any of the subsystems, be it hardware or software. Safety is the result of a well thought out system architecture and can only be reached by following a systematic and formalized process.



Service domains

- Training
- Formal model checking and software verification
- Embedded software development
- Customer specific adaptations of Altreonic's software tools.

Enabling technology

Altreonic's approach shines in the context of novel and daring applications where scalability, distributed operation, high reliability and real-time are dominant requirements.

OpenComRTOS is an enabling technology and thanks to its small footprint it combines performance and low power.

Application domains

- Robotics
- Autonomous systems
- Machine control
- Parallel DSP systems
- Many-core platforms
- Fault Tolerant systems
- Measuring and sensing

Unexpected benefits of going formal

"Altreonic has first hand experience that its methodology delivers. While we have a 20 years experience in real-time embedded systems and in particular in developing and selling the highly successful Virtuoso RTOS (acquired by Wind River in 2001), it was still a mind opening experience when we developed a new RTOS from scratch, whereby a systematic methodology was followed. Moreover, we found that using formal methods was not that hard and could be done with a small but dedicated team.

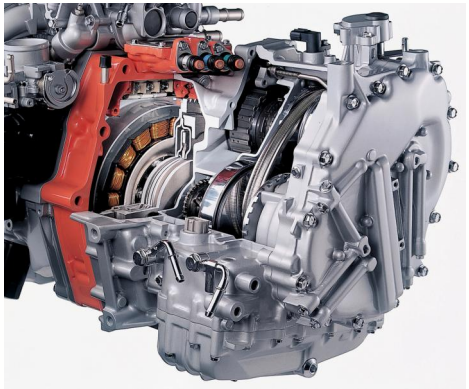
The abstraction provided by the formal tools allowed us to completely rethink the architecture without getting lost in the details of the implementation. The result was astonishing. The code size for example is 5 to 10 times less than the equivalent functionality of a traditionally designed RTOS. We can easily fit all functionality and more in 5 to 10 Kbytes per node. In addition, OpenComRTOS is much more portable, scalable and safer. It can also transparently support heterogeneous, many-core or networked target systems, which also enables a new way of looking at fault tolerance.

Sometimes, people tell us that small code size is not that important anymore as memory is cheap. They seem to forget that memory speed hasn't kept up with processor technology. Less code and less data also means less power consumption and higher performance. The transparent parallel/distributed operation also makes it easier to use multiple cores at lower frequency, further reducing power consumption.

The lesson to remember is that rigorous engineering doesn't need to be more expensive. It does not only provides more trustworthy products, it can also make them more cost-efficient and less taxing on the environment. Good engineering has always been about resource management."

Open Licensing

Customers want to reduce their risks and be able to verify and certify their products. Beyond the Open Source model, an Open License not only delivers source code, but also the formal models, design documents, test suites and the right to use and modify, even to resell. This is innovation as well.



KISS revisited

"Keep it Simple but Smart, which means that a complex solution is a problem not well understood. On the other hand finding a simple solution can require quite a lot of hard thinking."

Code Size Matters

Did you know that processors have become about 1 million times more powerful since the first one was created? This is thanks to Moore's Law telling us that the semiconductor industry would double performance every 18 months because they could make the features smaller and smaller.

But did that make the applications like your laptop 1 million times as faster or more powerful?

No, it didn't.

Why? Because memory technology couldn't keep up. Software became bigger every day. This means that the CPU is kept waiting for the memory before it can put it in its cache. This also means that that more energy is needed.

Hence, code size still matters. Less gives more performance and requires less energy.

Ultra Low Power Processing

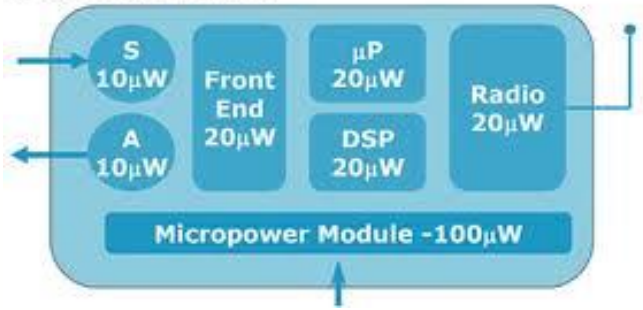
Wearable or even bio-implanted electronics are increasingly assisting our human sensor system.

At the heart is often a small chip that senses the environment, amplifies the signal and feeds it to one or more small on-chip processors.

The more processing performance available, the smarter we can make these devices. But, the limiting factors are today no longer size, but power requirements and heat dissipation.

Small code matters and can make the difference between having a laboratory demonstrator and a real product on the market.

Figure 1
Schematic representation of the major components of a WSN



Green Software

Does small code size only matter for ultra low power devices? Of course not.

Some server clusters that are the centre of our ubiquitous internet require a small electric power plat to operat en have their own cooling system.

A lot of the heat comes from the ever growing software size. And while it enables faster development of more reuse, the overhead can be a factor of 100 to a 1000.

The key is the architecture. If we could develop OpemComRTOS from scratch and using a formalized approach make it 10 times smaller than a hand-coded earlier version, what does that mean for mainstream software?

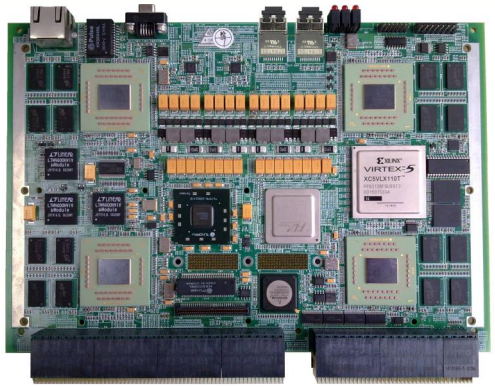
Again code size matters.

Code Size Matters

If the processor are getting too fast for the memory, how can we avoid the performance bottleneck?

Algorithms are often concurrent by nature. But they execute on sequential processors. Therefore it helps to decompose the software in smaller concurrent units, each being smaller. Less instructions means less often going to memory and less processing cycles. The result is less code giving more performance.

As the side-effect, one can then distribute the application over multiple processing cores. As each can run slower than as single processor, there is less mismatch with the memory speed and less power will be required.

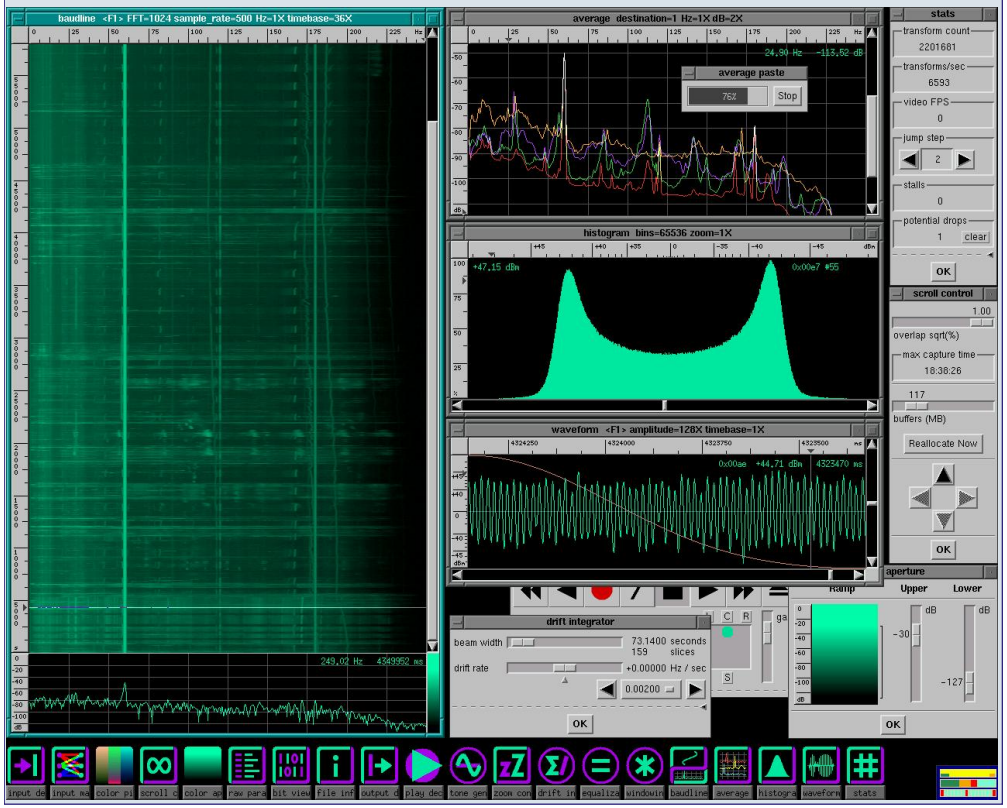


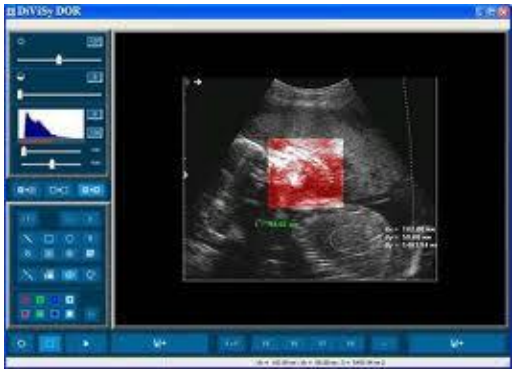
StarFish© Very High Speed Processing

Embedded systems often have to process real-time data coming from the environment. The amount of data can be massive either by its nature, either by the fact that a large number of channels are sampled. Also to extract meaningful information (e.g. object recognition) very complex and processing intensive algorithms are needed, often necessitating the use of parallel processing hardware.

This is the domain of embedded supercomputing. This domain is often even more constrained by power and size restricting because the embedded computer is based in a difficult environment.

OpenComRTOS was designed with such boundary conditions in mind.





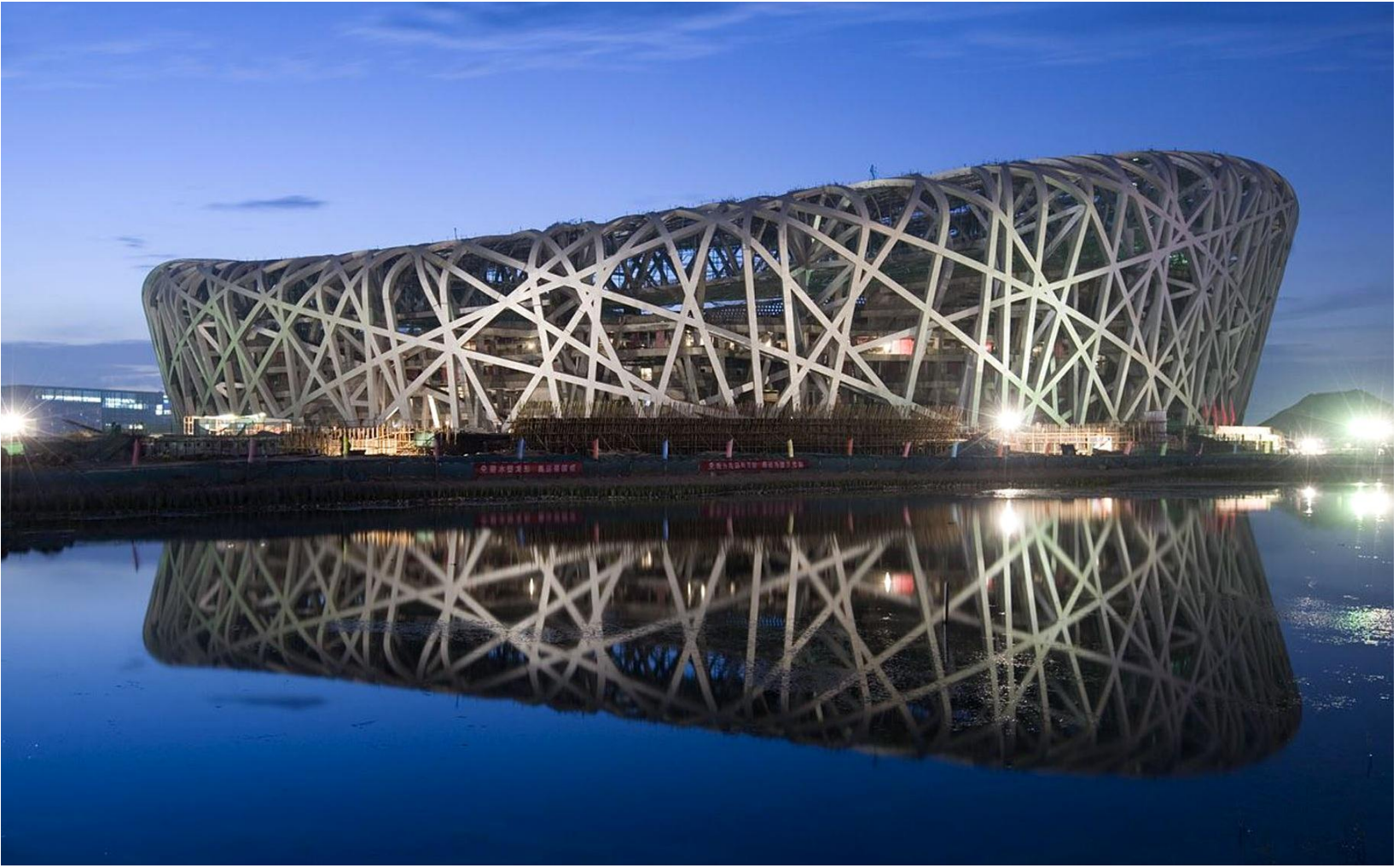
Parallel Software

If the processor are getting too fast for the memory, how can we avoid the performance bottleneck?

Software is essentially modelling of systems. Most systems are composed of concurrent sub-systems that interact. Hence, concurrent software is more natural that the large sequential programs we find today.

In GoedelWorks the user will map his specifications to separate entities. Mapping them to the concurrent tasks of OpenComRTOS Designer is therefore natural.

The code is easier to maintain, and easier to parallelise, hence providing more performance.



If it doesn't work, it was art.

If it does, it was good engineering

Contact:

Altreonic NV

Gemeentestraat 61A b1

B3210 Linden—Belgium

Tel.:+32 16 202059

info.request @ altreonic.com

From Deep Space to Deep Sea

Altreonic

Trustworthy Forever

www. Altreonic.com