



Subject: Open License Society

**Research Company for
Systematic Systems Engineering and Development
With Trustworthy Embedded Components**

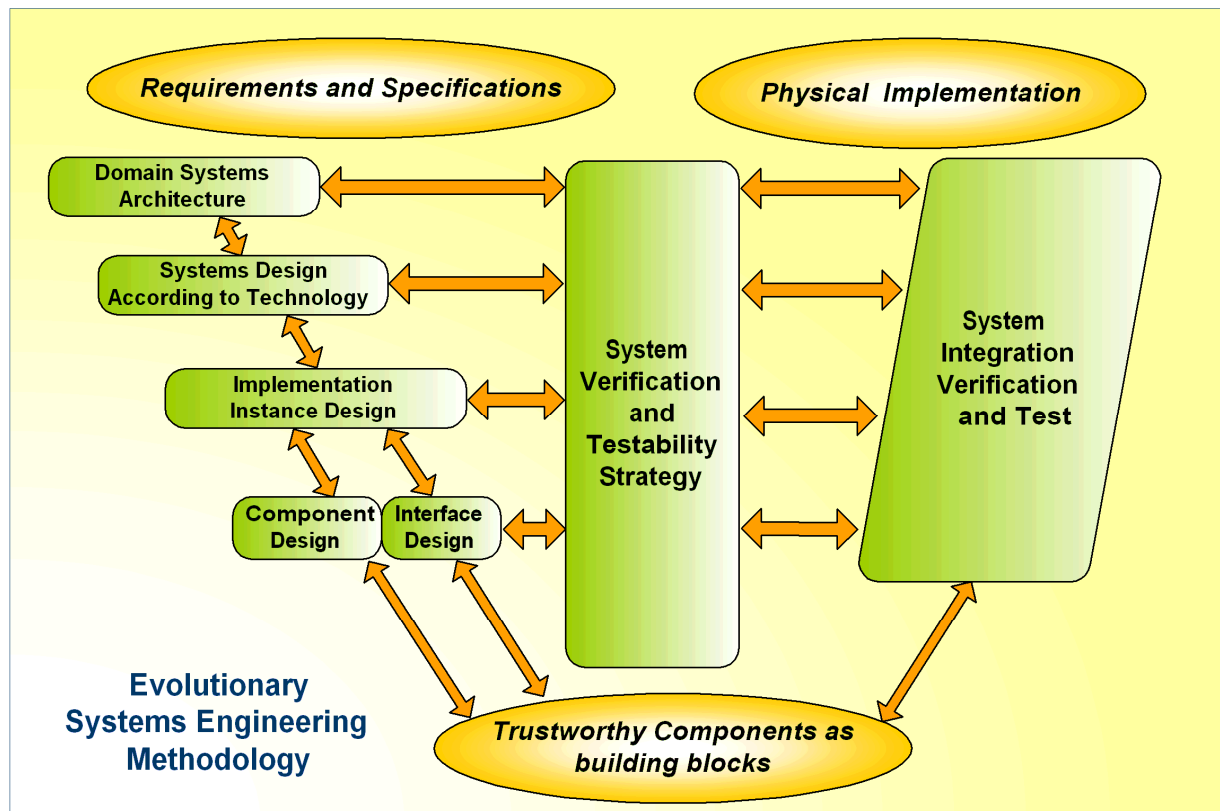
Positioning White Paper

Version 1.4

Created: 15th January 2004

Updated: 14th January 2008

Author: Eric Verhulst, Director



1/14/2008

Introduction

Developing a system is perhaps one of the most challenging tasks. While often associated with engineering, it actually covers a very wide domain. A system might be a small chip that is composed of multiple modules when looked upon from the inside; it can as well be a worldwide transportation system as it can be a social and political organization. Typically developing a system is done by experienced people and requires the input of all the stakeholders such as policy makers, management, users and yes, also the development engineers. As this is such a wide domain, the Open License Society will first focus on the domain its founders know best: embedded systems.

Embedded Systems can be defined as systems or devices that are capable of accepting data from sensors, after conversion processing the signals in a digital form (often in real-time) and generating appropriate responses. Such embedded systems contain a processing engine with a minimum amount of computing power. They are increasingly becoming part of our living environment, even if we might aware that there is a processor inside. A great multitude of embedded devices is being applied in very different shapes and applications.

While the above definition can equally be applied to general-purpose desktop or server computers, embedded systems are typically characterized by a specific functionality in a specific domain, even if personal computers are being used as part of an embedded system. In addition, embedded systems are often designed with strict boundary conditions of power consumption, size, cost price and reliability. While the first characteristics are often a matter of selecting the appropriate technology components, applying the technology and making sure the embedded system will meet all functional and reliability requirements is a matter of correct system design. Part of the challenge for the embedded systems developer is that each embedded domain has its own history, the diversity is wide and there is a lack of a unified engineering approach between the domains, even if from a functional point of view the similarities can be very high.

The Open License Society as a research institute is aiming at developing and promoting a coherent unified system design methodology that is accessible, reliable and cost efficient. As a research institute the social aim is to function as a center of excellence and to promote the developed methodology to a wider industry to increase their productivity and competitiveness resulting in embedded systems of higher quality and reliability. As a research institute, the Open License Society aims to cover the whole domain from the cognitive processes that govern the engineering methodology to the generation of reference platform designs. The aim is to develop an integrated methodology and supporting tool chain that covers the path from first idea to implementation, albeit in an incremental way.

In order to achieve a wide adoption an affordable 'open licensing' model is introduced for the core tool chain and associated reference platforms, complemented with activities of training and general education. The adoption of the methodology in a systematic way allows creating a large momentum as third parties can pro-actively develop complementary tools and reference platforms, with the institute focusing on the methodology it self and the core tool chain.

According to market research, close to 50 % of all embedded projects that are started can be classified as a failure, don't reach their specifications and/or are delivered long after the planned date. For a small region like the Benelux this means that if the success rate can be made to increase from 50 to 60 %, there is a direct economic impact in the order of 5 billion euro.

The following text briefly describes the current mission of the Open License Society. Subject to change.

Background and history

The Open License Society was founded by Eric Verhulst and Annie Dejonghe with support of Lancelot Research NV, their management company active in technology assessment, market research and management services. Prior to Lancelot Research, the founders have created the Eonic group of companies in 1989. Eonic was initially specialized in parallel processing and developed the distributed Virtuoso Real Time Operating System (RTOS). Virtuoso was a unique RTOS as it featured so-called 'distributed semantics'. This allowed for fully transparent parallel programming, independently of object mapping and target network topology. This was achieved by implementing a pragmatic superset of the CSP (Communicating Sequential Processes) computing model developed by C.A.R. Hoare. This model was combined with a distributed scheduler and communication layer. Given the portability and scalability of the Virtuoso programming model, it evolved into a middleware layer for developing embedded programs, even on heterogeneous targets. Subsequently, Eonic applied the Virtuoso philosophy to hardware design and developed the CSPA (Communicating Signal Processing Architecture), hereby shifting the emphasis from the processing nodes to a reconfigurable communication backbone.

Over the years, this history has provided an enormous wealth of experience with embedded systems design. Exposure to customers, porting Virtuoso to different platforms, R&D projects covering from fault-tolerance to satellite design as well as experience with third party software have given an important insight in the problems encountered when developing embedded products. Often these problems start in a very early stage of a project. E.g. in many cases customers had a hard time clearly defining their own requirements and more than once, especially in smaller companies who cannot necessarily afford the tools and training, it was found that engineers would develop code without much of a real engineering methodology and process. In addition serious weaknesses were found in the offerings of the embedded market. E.g. there are some 150 commercial RTOS on the market, few have any support for embedded distributed processing while more and more embedded devices consist of multiple processors while each embedded device is becoming interconnected with another one. At the same time, commercial software has proven to be expensive without necessarily offering high quality and certainly few offer certification. In addition there is the constant risk of perenity for the customer as happened with Virtuoso after Wind River Systems Inc. acquired the technology and subsequently discontinued the product. This was a major problem for customers with long-term projects. The emergence of Open Source software was partly a reaction against such problems emanating from the commercial vendors, but it has shown to have its own problems. E.g. often the Open Source software is only maintainable by real 'hackers', whereas the concepts and real-time behavior leave a lot to be desired. Moreover, even if the software is freely available with source code, the lack of documentation and the difficulty to understand the lower level parts have shown that the project development cost with Open Source is often as high as with commercial software. Moreover, just like most COTS software cannot be certified because there is no real design documentation, the same applies to Open Source software where the only documentation often is the source code. Supported by a market research program sponsored by ESA, Lancelot Research further developed the Open License model initiated at Eonic. The market research confirmed the need for such an Open License model. This being said, such observations do not ignore that commercial and open source software is and can be a good enough choice when it fits the application. At the Open License Society we want to focus on the development of software components (but that includes 'soft' hardware IP) as 'Trustworthy Embedded Components'. To handle the growing complexity, standardization and reuse of components and architectures are a must. But this can only work if such system components are fully characterized and certified to work according their specifications. Hence in the Open Licensing model, the licensee not only gets source code but also the formal analysis and validation documents.

The accumulated experiences as well as the development of the Open Licensing model are the basis for the charter of the Open License Society research institute. First communications on the initiative have already resulted in potential partnerships in Sweden, Germany, U.K and Italy, possibly creating a new European momentum. While the initial objective is to focus on embedded systems, it is clear that a sound methodology has an impact on many more areas in the socio-economic domain. So over time the Open License Society is to become a Systems Engineering Company. It actually happened in January 2008 with the first release of OpenComRTOS.

Engineering as a cognitive process

The concept of an embedded system is of course not new. Since mankind's early days, humans have developed devices to enhance their environment in view of improving their lives. The major change has come with the advent of semiconductor technology. This has allowed adding an increasing amount of 'intelligence' and programmability to embedded devices, hereby greatly adding to their complexity and abstract content.

From the development point of view the change has been one from 'craftmanship' to 'engineering'. Whereas the individual creativity skills remain important the development process itself is increasingly becoming abstract, moreover as the intangible 'software' content is becoming the dominant part. The main result today is that embedded devices can be extremely complex and are being composed of a very large number of 'entities' that all must work together to achieve a wide range of not necessarily fully compatible requirements. These requirements can range from very low cost to fault-tolerance, the latter becoming an increasingly necessary quality attribute.

This evolution is proven to be a major challenge for the industry, moreover as the competitive pressure is high and increasingly global. Studies have shown that about 25 % of the projects never achieve a result, whereas less than 50 % delivers within acceptable limits of performance and cost. This has to be compared with e.g. the construction and aerospace industry that has a much better track record. The first industry succeeds because there is an enormous history and track record that resulted in a predictable engineering process. The second succeeds because the stakes are high and the aerospace industry had to develop a rigorous methodology to be able to deliver, even if the costs were high. Hence, few bridges collapse when being constructed and flying has become one of the safest means of transportation over a long distance.

What the Open License Society wants to achieve is that by using a sound and coherent - which means predictable - (engineering) process for developing embedded products, this becomes a rigorous and natural engineering process rather than the craftsmanship it too often still is. Given that the abstract content of the embedded systems is becoming dominant, this is a hard challenge.

The first challenge is that we need to understand is how the thinking process works when engineers have to develop their systems. There are more and more indications from fundamental neuroscience research that the human mind itself is a bit like a very complex but fuzzy computer (for the specialist reader we are aware that this is probable a very simplistic statement). Humans shape their environment, but the environment shapes the human mind as well. It's a fuzzy process because the human mind has to be flexible to survive because strict logic doesn't give all the answers needed. That's partly because in real live all elements are not always present to make the right decisions in time. Hence often the human mind will apply template matching rather than rational reasoning. The price to pay for being able to make such quick decisions is that the outcome can be proven wrong afterwards even if right in most of the cases.

Hence as engineering an embedded system that is composed of a lot of abstract entities needs to become as predictable as possible, we need to understand how this mental process works. 'Know thyself' is since the Greeks the key to wisdom and knowledge. In our case knowing how the human mind works is a key to a predictable engineering process. And as we know that it is not always as rational as it should be, moving towards a process of 'formalized' thinking will certainly improve the odds that the outcome will be better.

Note that there is an equally difficult related challenge: understanding what are the real user requirements. This can be very complex as we are at a stage were embedded devices are developed that create a need after they have been adopted. Similarly, also all the stakeholders do not always formulate their needs on a purely rational basis.

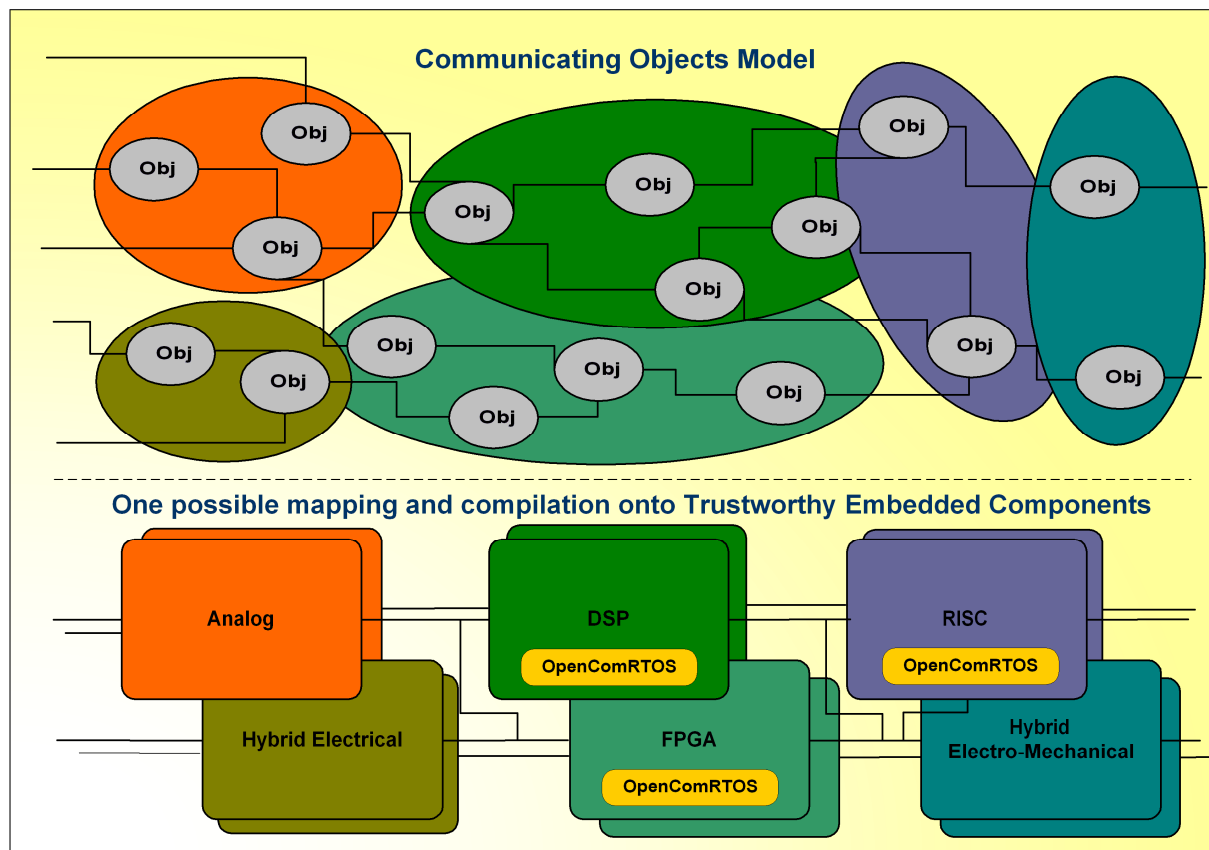
It will not be the main task of the research institute to research how the human mind works, but it will be an important activity to stay aware of the latest insights and to apply them to the development methodology. This includes providing this insight to those who follow the trainings and seminars and use the tool suite in order to develop better embedded systems.

The initial development approach: Interacting Entities

Without laying claim to have the final answer, we believe that a coherent approach must start from the user's point of view when developing embedded systems. This starts by defining what he expects the system to do for him ('the mission') and what are his expectations ('the requirements'). The task of the development engineer is to translate the mission and requirements into specifications and to select the right technology to implement these specifications into real systems. However, there are other requirements as well. These can range from the social level (e.g. "shall not produce harmful side-effects when used") to economic ones ("shall be competitively priced and be more advanced than the competition"). Even when starting from an existing design, is it imperative to be able to define the requirements and specifications of the derived system.

Hence at the outset the system is a 'black box' and the engineering process is one of making trade-off decisions to achieve the result in a way that meets all the expectations in an acceptable way. This means as good as needed. A natural way to tackle the development is to define the functional requirements and to decompose the black box into a number of smaller 'sub'-systems to reduce the complexity that has to be handled at a given time. Often these will match the functionalities although some functionalities emerge at higher levels as a result of the combined operation of the defined subsystems.

Important is to define how these subsystems interact. Such interaction can best be modeled by defining synchronization and communicating schemes. It has the major benefit that it restricts the interaction between the subsystems to a well-defined communication protocol, isolating the internal behavior from the rest of the system, unless explicitly exported. Note that we define 'communication' in a very broad sense and the essence is the interaction it carries between the objects. Another view is to regard these interactions as contracts between entities. When both sides respect the contract, they can become matching entities of a larger system.



This was precisely the original idea behind the formal CSP developed by C.A.R. Hoare and implemented in a pragmatic way in the Virtuoso RTOS and the CSPA architectural model. Although similar concepts are around with different names, we will call this approach 'Interacting Entities' whereby an Entity can be any subsystem

module that itself can be composed of different entities in a hierarchical way. Note that this was initially labeled “Communicating Obkejts”, but we now believe that “Interacting Entities” is a better generic term for our purpose. A difference with other approaches is that the interaction pattern comes first as it allows to define the system’s behavior without the need to know the internals of the Entities, even if this assumes some a priori know-how about the internals. Hence the interactions are the key, just like they are the key to understand how the different entities can through interaction become as system.

There are several benefits to this approach. First of all, it allows postponing the implementation details of entities to a later stage. Using the appropriate interfaces, it even allows to develop fully heterogeneous systems with parts being analog, software, electronic, digital, mechanical, hydraulic, etc. It also provides means to isolate functionality in an orthogonal way from the interaction and the real-time behavior. And it allows developing logically equivalent execution models in software, albeit not necessarily with the same timely behavior. This is logical as the system is at the outermost level described as a set of functional requirements with specifications taking into account the boundary conditions like e.g. size, power consumption, real-time behavior and the characteristics of the environment in which it has to function. The benefit is that one can have at all stages a logically equivalent executable model, independently of the selected implementations.

Below a certain level of decomposition, it often no longer will make sense to decompose further. At that moment, domain specific descriptions are needed. In software, this can still be a set of virtual objects modeled as communicating processes, or a state machine. In hardware this can be the description of an analog filter or e.g. a piece of logic with a well-defined (and then often rigid) behavior).

To summarize, we see the following methodology for embedded systems development:

- Description of the ‘mission’ of the embedded system in terms of a user, with the possibility that the user is another system.
- Description of the user ‘expectations’, to be transformed into the requirements for the system to be developed
- Description of the boundary conditions such as size, power consumption, cost price, temporal behavior, user behavior, environmental effects, etc.
- Translation of the requirements and boundary conditions into system measurable specifications

The design phase starts by defining the entities that compose the system, their dependencies and the communication protocol or interactions between the objects. This decomposition is hierarchical and is done at the highest level of abstraction possible.

When the implementation level of an entity is reached, domain specific implementation tools and languages are needed.

Concurrently with the development, a test strategy needs to be developed to certify and to be able to verify that the system meets its specifications at all time.

Hence, one can distinguish three main components in the OpenDesign core tool chain:

OpenSpecs:

- A tool for collection the user requirements, boundary conditions and translating these into specifications
- A tool to decompose the system into entities in a hierarchical way
- A tool to document and verify the inter-entity interactions
- A tool to document the object’s behavior at the implementation level.

OpenObjects:

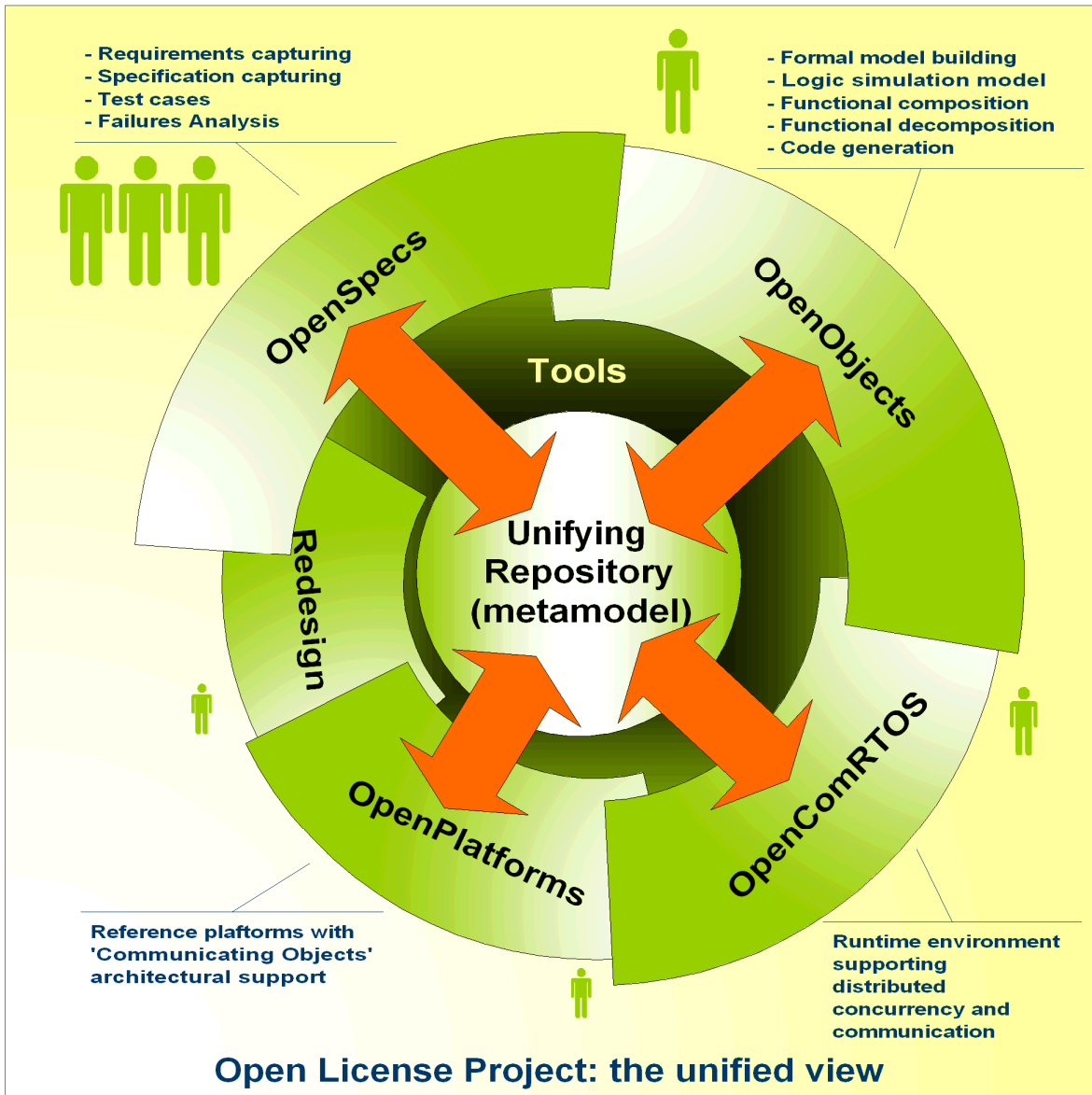
- A tool to translate the OpenSpecs statements into an executable, eventually graphical representation
- A tool to build and analyze a formal model of the system or of specific properties of the system
- A tool to generate executable simulation models of the embedded system
- A tool to generate the production code of the software parts of the embedded system.

OpenComRTOS:

- A runtime environment with following characteristics:
- Small and efficient network-centric RTOS
- Scalable message passing protocol
- Formally analyzed and validated, hence Certifiable
- Runtime monitoring

- A functional simulation and cross-development environment for systems developed according to the 'Interacting Entities' paradigm

The tool chain will be capable of keeping a consistent state between all levels even when changes are made at a lower level. Note that this is not only an important requirement, an emerging quality from a consistent model at all levels. In the absence of intra-level consistency translation steps are needed between the levels and this is often not possible for semantic reasons. Note: some of these core tools already exist in a beta-level or early development stage.



The longer term goal: high reliability, fault tolerance and certification

The increasing use and presence of 'smart' embedded devices in our daily environment has an important consequence for the requirements put to these devices. Not only are we increasingly becoming dependent on the availability and presence of these devices, when they operate, they must operate in a predictable manner. Often this means that they must be reliable, which means very seldom fail, and fault tolerant, which means

that they continue to operate even in the presence of faults. The absence of these qualities can have serious consequences ranging from a high economic cost to life threatening.

The challenge the embedded industry faces is that these qualities must be achieved in the presence of demanding boundary conditions:

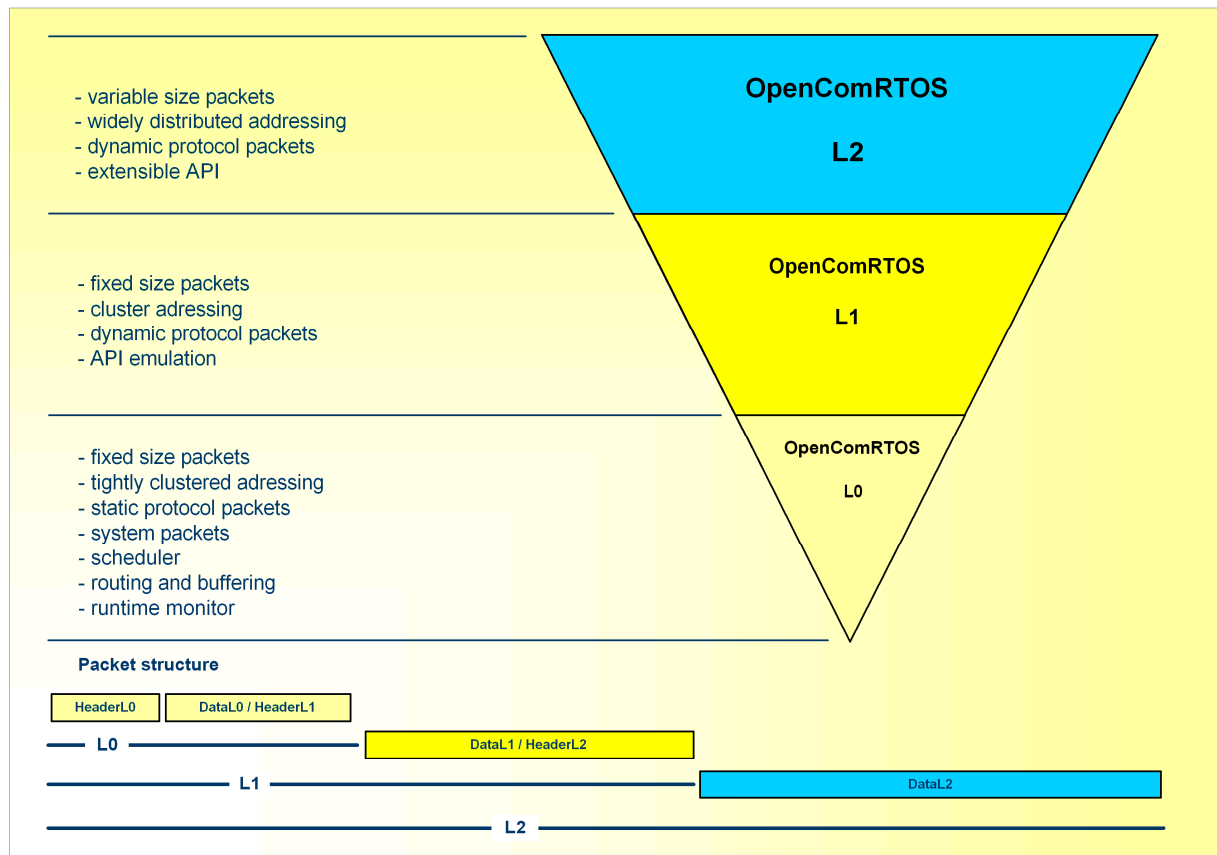
- Cost-efficiency, as well as for the development as for the manufacturing costs
- Shorter development processes
- Increasing rate of change of the technology
- Increasing complexity and size of the embedded systems.

The industry and the academia have not stood still in taking on this challenge. Indeed, very rigorous formal methods were developed (CSP itself was a formal language) and have been successfully applied. The problem with formal methods often has been that they require experts to use them and that they hit a barrier when the system becomes too complex. The global development process itself became the subject of a formalized approach resulting in e.g. the ISO-9000 norms, the CMM levels of process maturity, etc. The industry has also created its own body with the OMG group hereby defining standards like CORBA, UML, XML and others. While often deemed expensive to use, such approaches have nevertheless given us reasonably reliable worldwide transport services. Without such an approach, the aerospace industry would not have been capable of delivering.

The Open License Society believes that such approaches need to become affordable and widespread for the whole of the embedded industry. Partly to increase the success rate, partly to improve the quality of the embedded systems, partly to become more productive while being able to keep up with the rate of change.

As was often seen, reliability and fault-tolerance are often in jeopardy because small system details have been overlooked, or worse have been given lower priority and were added late in the development process. Quality is a feature that needs to be designed in. It cannot be added afterwards because it has an impact on the global architecture and covers more than one domain. Likewise, its enemy is complexity. Complexity grows with the advance of technology, but is often a sign of the fact that the problem was not well understood. This essentially closes the circle. As development is an activity carried out by humans, it will help to understand the problem to solve if we understand how the cognitive engineering process works. This comes down to bringing perception and logic reasoning in harmony. This will help in defining the development methodology as a well-mastered and repeatable process. Therefore one must be able to define the requirements and specifications in a structured way. Dealing with the complexity means increasing the abstraction level. The latter implies the use of trustworthy components and leads to the 'Interacting Entities' methodology. Another implication is that the tool chain supporting the methodology must be bootstrapped. Each tool itself must be developed according to the methodology and this means that it is certifiable. This is in particular true for the runtime environment. Although the underlying goal will be 'correct by design', one has to be pragmatic and part of the proof will still come from extensive but carefully designed Monte-Carlo type testing that will provide the ultimate reliability and fault-tolerance confidence.

In summary, we believe that 'trusted embedded components' must be designed from the start to deliver three qualities: reliability (or correctness), safety and security. The key to achieve these is to use formal methodologies as much as possible when developing them. It should be noted that formal methods are not just seen as verification tools, but rather as modeling and architectural design tools supporting the formalized engineering process in a rigorous way.



Supporting tools

The Interacting Entities tool chain will initially be targeted towards traditional embedded developments. This means that the system is composed of some enclosure and internally one or more electronic boards. The enclosure provides mechanical support, input/output capability with the external world of the embedded system and the 'look and feel' for the human user (if any). The electronic boards are the heart of the system, however it is assumed that at this stage the boards are designed using off-the-shelf available components with standard CPUs and FPGA logic providing flexibility and reprogrammability. On the standard CPUs, we will most likely use an RTOS. For more complex systems, the FPGA will provide pre-and post processing functions between the input and output sections and the general purpose CPUs. The communication layer and part of the RTOS can be implemented in the FPGA. This clarification of an embedded system makes it clear that the design of the new chips itself is not supported by the tool chain, although the target system can be a prototype for it.

While the core tool chain will be made available under the open licensing terms from the Open License Society, we anticipate that most of the supporting tools will be co-developed with third parties. Such tools fall into three categories:

System development support

- Diagramming support
- Documentation support
- Formal specification, validation and testing support

Code generation support

- To import designs from other environments
- To export designs to back-end runtime environments
- Device driver skeleton generation

- (Formal) code analyzers and optimizers
- Real-time support for the RTOS
- Scheduling analysis
 - Scheduling generation
 - Runtime monitors and visualization tools

In particular, we need to be aware of other developments in the embedded market and if meaningful, seek to integrate to foster the transition to the Interacting Entities methodology (or vice-versa). Such developments are, e.g.:

- SystemC
- UML and UML/RT
- SDL and SDL-RT
- SysML
- Matlab/Simulink
- CORBA (in particular microCorba)
- Component based development
- Model Based Architectures
- ...

Reference platforms

Given that an increasing amount of embedded systems is defined by software it becomes natural not only to seek a standard development tool chain, but also to seek to use standard reference platforms for embedded systems. This trend is increasingly becoming applicable to hardware given that FPGA are increasingly part of a design. Of course, this approach needs to be offset against the need for low cost when developing very high volume applications, the latter often necessitating a customizing and optimizing design step.

It is envisioned that the Open License model can be applied to reference platforms as well. Hence the Open license Society will encourage the development of application specific reference platforms that in conjunction with the tool chain and third party application specific libraries result in a fast development track in that specific market segment, even if the next step is a high volume cost-reducing redesign step.

From the methodology point of view there is no contradiction as the reference platform allows an easy development of the application, while it continues to serve as development platform for e.g. upgrades and new developments.

Although not exclusively, it is believed the concept of 'Interacting Entities' need to be ported to the platform architecture as well. This is in line with the evolution towards 'switch fabrics'. Early adoption can be found in the FPGA where LVDS links are now becoming standard interfaces, emerging standards like RapidIO, StarFabrics, PCI-X and many others. One of the first organizations to adopt such an interconnection standard was ESA with SpaceWire. This evolution signals the change towards separating the functional behavior from the communication behavior and is in line with the concepts that will be promoted by the Open License Society methodology.

At least one partner has been identified who is willing to contribute his design to the reference platforms.

Beyond 'Interacting Entities'

By adopting the Interacting Entities or Interaction Oriented development paradigm, one can gain several benefits that reach beyond a reuse of existing technology. First of all, there are now several formal languages and methods that use similar concepts. CSP was one of the front-runners, but meanwhile developments have taken place to extend the original concepts. E.g. uCRL adds datastructures and buffered communication while Pi-calculus adds mobility of processes. This is important because it means we have the formal background and tools allowing us to validate mathematically systems composed of a very large number of interacting objects. Any other approach will be less complete or simply 'ad-hoc'.

A second benefit of the approach is the total encapsulation as all interaction is through a well defined interface, hereby hiding of any internal operations and datastructures. This also makes distributed and heterogeneous systems much more natural to achieve, provided the system level layer supports natively the

'interaction' functions and the handling of 'entities'. Hence we get portability as well as support for adding new technology components in an incremental way.

A third but very important benefit from the technology as well as economic point of view is scalability. As the Interacting Entities approach is a logical one, systems designed this way can be made available without major changes in different ranges of performance, such as pure computational speed, power consumption or service level.

The reason for this is that there is no real limit on the 'grain-size' of an Entity. The whole system itself can be an entity in a much larger system while a small software entity can be implemented with just a few lines of code. The proposed methodology with a central repository will work well, but only well if all system development activities and tools share the same semantics. Building a development system with ad-hoc tools would require an enormous and continuous effort in translation tools that in practice would not only absorb a serious amount of development resources, it is even questionable it can work at all because the semantic gap between ad-hoc tools and methods is too wide. Hence major competitive advantages as well as a higher degree of reliability and trust will be achieved if the used components are also developed with the concept of 'Interacting Entities' in mind.

What this means is that we also get the right means to handle the looming wall of complexity. On the one hand we raise the level of abstraction while shifting the effort to the analysis and model building phase. In the case of software we can even hope to reduce the error-prone coding phase to automatic code generation. On the other hand the 'entities' approach provides an opportunity to radical simplification if consistently applied in all tools and levels of the methodology. E.g. at the programming level we should only use concurrent programming models, at least at the highest level. Inside the concurrent objects existing programming styles can be used, but if we have support for fine-grain concurrency more can be gained because the grain-size of the component entities is smaller. At the processor level the same model can be applied. Rather than trying to make the myriad of processors fit into the model, it is proposed that a new but single type of architecture is developed that supports lightweight concurrency and communication in the hardware. The result will be an architecture that is simple and elegant like the now defunct transputer. Critics might say that a single architecture will not be sufficient for all applications. Our reply is that if fine-grain concurrency is supported in software as well as the hardware we have a much better instrument to meet different application needs because we get scalability. In addition, different silicon technologies can be used to make trade-offs between performance and power-consumption, while for very high performance co-processing units can be added. Using automatic silicon and compiler generators it is now possible to generate CPU cores that remain compatible at the level of a high level language but with specific extensions to address different needs. One can even further reduce power consumption by exploiting parallelism on the same chip as this allows putting multiple CPU cores at lower frequency on the same die. The concurrent programming model makes this rather easy to achieve. This being said, this also implies that the inter-processor communication becomes standardized as part of the architecture and instruction set as well as at the physical level when used to connect multiple processor chips and I/O devices. Note that ESA has already adopted a similar approach by adopting the SpaceWire interconnect as the standard 'link'-bus in all their systems. Perhaps not surprising, the SpaceWire link was derived from the transputer architecture.

Supporting activities

The Open License Society principal aim is to develop and promote a coherent unified system design methodology that is accessible, reliable and cost efficient. As a research institute the social aim is to promote the developed methodology to a wider industry to increase their productivity and competitiveness resulting in embedded systems of higher quality and reliability. This implies that the Open License Society is pro-active in promoting the methodology in public presentations, trainings and seminars. At the same time in-house workshops are needed not only to introduce the methodology to a wider public, but also to get the feedback from users to improve the methodology and tool chain. The importance of analyzing the development process as a cognitive process is recognized

In the same context, the Open License Society should engage in executing projects with the developed methodology. The goal here is not so much to develop this in a full blown activity, although this can be the mission of a spin-off or partner company, but to acquire real experience with real developments to discover the weaknesses and limits of the proposed methodology.

The Open License business model

The Open License business model is a result a many years of experience in the embedded market whereby it was discovered that neither the Commercial licensing scheme, neither the Open Source licensing scheme works to full satisfaction. Commercial licenses are often expensive and their use can require royalties, but they are not necessarily a guarantee for high quality. Open Source license software (and e.g. IP cores) is free, but often comes with little documentation. By providing source code Open Source products seem to solve the issues of the commercial offerings, including perenity. In practice both commercial and open source offerings can be very useful and adequate for a specific development, but both do not address the real problem of predictable and cost-efficient system development.

Actually, many conditions are not fulfilled:

- Access to all development sources
- Documentation of requirements
- Documentation of specifications
- Documentation of implementation
- Formal specification and validation (certification)
- Source code for long term maintainability

In addition the current market is highly fragmented, putting the economic viability of both models at risk.

Most often the developer must select a set of tools from different sources. Most of the time these tools are not easily integrated into a seamless tool chain. A major reason for this is not so much the practical side of things (e.g. exchanging data) but the difference in 'semantics'. Often each tool will have a set of underlying assumptions about its application domain resulting in early optimizations. A typical example is e.g. matlab/simulink. While widely used, the tool was a simulation tool at the onset for developing algorithms on a PC. For this it was very successful. But a result, the central paradigm is a main control loop combined with a dataflow type representation. As the internal data structures are hidden it is very difficult exercise in reverse engineering to use the tool for acceptable multi-tasking code generation, even if conceptually the blocks can easily be mapped onto RTOS processes or tasks.

Hence, two lessons are important from these observations:

- A critical mass of a comprehensively seamless integrated set of tools is needed to achieve a critical mass for a coherent system design methodology
- A middle ground is needed between the commercial licensing model and the Open Source licensing model.

The result of these observations is the Open Licensing model. Hereby the licensed item is made accessible at an affordable fee to a large community. This allows reaching a critical mass and providing access capability to the large community of small and medium sized companies, who often opt for Open Source because they can't afford the more expensive commercial licenses. Under the licensing terms the licensee can freely use the licensed product with access to the source code. Support and upgrades are part of the licensing fee for a fixed period of time, after which the licensee can renew his subscription. No restrictions are placed on the development of application level or derived products for as long as the licensee submits the changes to the originally licensed product to the licensor, who, as legal owner, remains the only one in control of the source code and the subsequent releases. Hence, licensees are not allowed to sublicense the products but are allowed to develop application level products (in 'binary' form). Should they develop supporting tools, then the licensor has the first choice in re-licensing these to a larger community (on a non-exclusive basis) in exchange for sharing the licensing fees with the licensor. This means that e.g. the reference platforms can be licensed with all design data in order to re-use the design as part of an application specific development, but that changes to the reference design itself remain the property of the licensor even if the licensee can freely use these in application level products.

In order to create the momentum with the design methodology, the Open License Society as a research institute only intends to develop the core tool chain, inviting third parties to add complementary tools as well as reference designs. To be accepted as an Open License product, the product itself must be developed or documented in a way that is consistent with the Open License development methodology. While existing designs can be incorporated by adding the right documentation and design data (according to the methodology), new developments, including the ones done by the Open License Society itself, should be

strictly done using the system development methodology it promotes. Third party Open License products are non-exclusively licensed by the Open License Society in return for a share of licensing fees with the third party.

Sponsors

While the Open License Society aims at improving the embedded systems development process in a global way, the economic benefits will only be reaped by achieving a wide adoption. As such the methodology and the associated tool chain acts as a catalyses and generates most of its economic added value indirectly. This is partly due to the limited volume potential of embedded development tools, itself partly a consequence of the functional and historical fragmentation of the embedded market. This explains why there are about 150 RTOS on the market and with perhaps an even greater number of in-house developed ones. Moreover, most of these tools, but that also applies to the components available to embedded developers, are not compatible with each other, resulting in developers spending a large effort on achieving integration. Hence the need to build in critical mass from the beginning.

The critical mass is achieved at four levels.

Firstly, the methodology is to be developed by a research institute with an aim to achieve widespread adoption and thereby developing the core tool chain as a coherent whole. The coherency is partly achieved by limiting the scope to the core tool chain. Part of the success will come from applying occam's razor, whereby the main question is not what to take with, but what to leave out. While standards have certainly contributed to efficiency in the development process, often they ultimately fail to achieve critical mass partly because they are developed by large committees whose members might have contradictory desires and as result many standards are overly complex.

Secondly, the Open License model encourages a widespread adoption and by acting as a catalyzer, it encourages third parties to contribute complementary tools, reference platforms, etc. In addition, the model aims at providing a quality tool chain available on a long-term basis. This is of particular interest to the small and medium sized companies that taken together represent perhaps the majority in the market, who cannot afford to develop such a methodology and tool chain internally. The result is the creation of a community while the industry as a whole benefits.

Thirdly, the Open License model introduces a new concept of 'sponsors'. Many players in the embedded market generate their added value by focusing narrowly on a specific segment in the value-chain. E.g. semiconductor companies develop and produce chips, whereas embedded consumer product developers provide pure 'functionality' to end-users. Embedded systems development tools are for such niche players important, but not their core activity. Such market players can become 'sponsors' of the Open License Society. They are invited to contribute to the financing of the research and development of the methodology and tool chain. In exchange they get a number of licenses that they can freely pass on at the normal price and are invited to participate in the quarterly technology review meetings so that they can help steering the methodology and tool chain.

Fourthly, the Open License Society recognizes the importance of local presence. Hence, it will seek to create a network of centers of excellence across Europe and later on in other continents.

International network

It is envisioned to create a network of 'Open License Society affiliates. There are several reasons that encourage this. First of all, language and cultural habits are to be included in the whole process. Hence they influence how a given methodology is adopted and implemented. In addition, promoting a methodology requires training, support and being able to execute local projects. The Open License Society has the intention to create an international network of associated 'local centers of excellence' to further enhance the critical mass and adoption of the methodology.