The need for expressing and describing architectures of embedded systems based products has been expressed for several decades. System developers in industries such as automation, automotive and telecom face functional growth, development and integration of systems encompassing multiple and heterogeneous components where time to market and the need to provide cost-efficient and high quality products are of paramount importance.

Architecture description languages provide a means for an architect to describe and communicate the fundamental organization of a system. The architecture description moreover provides a forum for eliciting, discussing and resolving conflicting requirements, and provides means for assessing alternative system designs. Given more detailed descriptions, system configuration and synthesis is also possible.

Given these needs, why is it that the industrial adoption has been rather slow? These and other questions were debated at the ICES workshop on April 12th which had the overall goal to discuss:

- What key formalisms, ADLs and visual languages, for design of embedded systems are there and what are the trends?
- What is the maturity (languages, tools) and industrial adoption of such formalisms?
- What are the industrial expectations on and experiences in adopting such tools?
- Which formalism best suits different types of systems and design tasks?
- What are the key outstanding research issues to pave way for larger scale industrial adoption.

A key fundamental challenge in developing embedded systems is that of managing their complexity while providing products of the desired quality, at the right time and at the right cost considering the whole life-cycle. As one important means to handle complexity, architecture description languages (ADLs) have emerged as a means to formally describe software and hardware architectures, providing a basis for analysis of system properties such as reliability and performance, and for synthesis (e.g. generating glue code). The dominating views provided by ADLs is that of describing the system structure, mainly with the notion of black box (SW/HW) components, following the lines of compositionality where the idea is that system properties can be derived from a configuration of components and their externally visible properties. This situation fits well for the purposes of a system integrator that specifies the system architecture and has to reason about system level properties without details of the internals of component implementations. It is clear that embedded systems expose many different types of structures. The term architecture is usually reserved for the fundamental organization of a system that determines essential non-functional properties.

At the same time several graphical formalisms have been proposed with the purpose to visualize, communicate and document advanced embedded systems. During the 70s and 80s, a large number of formalisms, such as structured analysis and design, and object oriented methodologies emerged. Examples of early ADLs and formalisms include MetaH, Hatley-Pirbhai modeling, CoDARTS and ROOM. Some convergence was achieved in the 90s with the emergence of the UML. Yet, the needs from different domains have pushed specialization.
As a result, a number of formalisms for embedded systems architecture description currently exist, including, for example:

- AADL - the Architecture and Analysis Description Language, an SAE standard
- UML2 and its profile derivatives SysML and MARTE
- Autosar and EAST-ADL in the automotive industry,
- Hardware description languages such as SystemC
- Meta-modeling approaches for constructing domain specific modeling languages (DSLs).

Obviously, each formalism is developed in a particular context, thereby emphasizing certain abstractions, relationships and properties. One thrust shared by these efforts is to combine the support for visualization with that of formalized descriptions that are amenable to analysis and synthesis. Another common thrust is that of combining a model-based with a component-based approach [Törngren et al, 2005]. Visualization, analysis and synthesis are all key to the engineering of advanced embedded systems based products, with the goals to facilitate communication among people, early decision making and design pautomation. However, while such formalisms have been researched, and prototype tools developed, there has been a slow, or only partial adoption in embedded systems industries. Moreover, the availability of several formalisms and emerging tools, together with the uncertain future evolution, makes the choice for a tentative developer non-trivial.

All these aspects provided motivation for the ICES workshop that took place as part of the Cyber-Physical systems week at KTH.
The workshop had a duration of one day and included two parts. In the first, invited speakers presented selected formalisms for architecture description and visualization, followed by a panel debate discussing the above mentioned topics. The second part of the workshop consisted of a hands-on session where the participants were able to try out various modeling formalisms and tools at the Mechatronics lab at KTH. Copies of the presentations are available to ICES members via Members Area of the homepage.

Summary from the presentations and discussion

The seminar presentations provided a good overview of the different formalism and also gave an indication of various harmonization attempts. A large number of research projects are currently using, extending, and developing tool support for the mentioned formalisms. The workshop provided a rather broad scope and selection of formalisms, from systems to hardware.

The keynote of the workshop, Bran Selic (well known for developing the ROOM methodology and for heading the UML2 standardization) advocated an approach based on a component paradigm, model-based engineering and standards. Bran pointed out the deficiencies in the OMG Model-Driven Architecture (MDA) approach which emphasizes platform independent models. Exploring the mapping and gap between functions/behaviors and execution/communication platforms is a central issue for embedded systems because it determines essential properties such as performance, dependability and cost. This issue thus needs to be explicitly dealt with; a statement well supported by research findings in the areas of embedded systems and electronics design automation. He also pointed out the importance of syntax in relation to how users are able to understand and find various representations preferable compared to others.

After the keynote, 8 presentations provided insights into the AADL, Autosar, EAST-ADL2, Hardware description languages, Domain specific modeling languages, the Rubus component model, Modelica and industrial requirements for architecture descriptions.

The presented formalisms represent different concerns, traditions and applications domains. For example, the AADL (originally developed as MetaH by Honeywell) has its roots in safety related embedded control systems development and is today mainly considered in the aerospace domain. Autosar and EAST-ADL2 (which both have their roots in the EAST-EAA project started in 2000) are developed for automotive embedded systems where the EAST-ADL2 adds requirements, environment, variability and higher-level abstractions to the Autosar descriptions of hardware and software components. The Rubus component model, which is used in safety related embedded control systems, has similarities in flavour with the AADL, but is mainly seen as a simpler and predictable version of Autosar for the vehicle industry. Hardware description languages are evolving towards higher levels of abstraction with the goal to provide abstract descriptions which can be used as a basis for hardware/software codesign. MetaEdit is a tool for prototyping, definition and industrial use of domain-specific languages. Modelica is language for modeling physical systems but is now being extended to provide better support for embedded systems.

Are all of the covered formalisms architecture description languages? All of the formalisms can be seen as ADLs in the sense that they are providing system descriptions, albeit at different levels of abstraction and emphasizing different properties. This raised the issue of what the needs for such formalisms are. A key issue for an architect is that of visualizing and communicating the fundamental organisation of a system. Another related topic is that of formalizing the architecture description sufficiently well, such that analysis of the system properties can be performed. The desire to formalize has to be balanced or somehow integrated with the desire to quickly sketch and visualize the system. Most of the current approaches tend to emphasize the formalization. As a result of this, architects tend to stick with tools such as paper and pen, powerpoint or visio. This trade-off is valid.
for all levels of architecting. A key aspect in using models is that of making them executable, enabling simulation. This aspect was also emphasized as important for architecture descriptions.

Are modeling languages any different from programming languages? While there clearly are many similarities in terms of syntax, semantics, compilers etc., programming languages provide limited abstractions when it comes to expressing several of the qualities and attributes of interest for embedded systems [Schmidt, 2006]. Programming languages could be seen as modeling languages that are focused on providing support for detailed systems design in terms of constructing programs. A program written in C or Java, for example, represents a model since the program is an abstraction of the actual behavior. The actual execution will yield its timing behavior and accuracy of computations depending on the hardware platform (and also depend on the compiler and linked libraries). It was however stated that many of the experiences and knowledge from the programming field were neglected by researchers working in model-driven engineering (compare with model-transformations vs. compilers).

**Summary of further issues raised and discussed during the workshop**

**With several existing formalisms, why is there a lack of adoption?**

There are probably several factors that explain the relatively weak adoption including the following:

- The use of architectural descriptions necessitates that a clear need, roles and resources have been determined in the organizations and this not always the case!
- The lack of of mature tools and standards. ADL tools have in many cases been developed as prototypes. While there are a very large number of UML tools, there has been a lack in understanding how to use the UML language for architecting. UML provides a large number of diagrams and both the individual diagrams as well as their relations can be (and have been) interpreted in different ways by tool developers as well as users. The UML standards have emerged without thorough testing.
- The lack of standardized software/hardware platforms upon which useful architectural abstractions can be built and with respect to which architectural description could be used as a basis for configuration and synthesis.

While behavior descriptions (such as Matlab/Simulink and Scade) and related design flows are quite mature for several embedded systems domains, structural descriptions à la architecture description languages have been lagging behind. The automotive AUTOSAR represents a new effort and departure from this state in providing a middleware standard and by pushing tool vendors to support AUTOSAR (thus removing the 3rd obstacle above). The success of AUTOSAR however still remains to be proven.

It is clear that there are adopters of the above mentioned formalisms. It does however take time and huge efforts to establish new development processes and technologies!

**Unification vs. diversification?**

An interesting debate arose regarding how to view the current diversification of formalisms. Two contrasting viewpoints were expressed:

1. The need for unification!
2. Embracing the diversification as a source of innovation!

It is interesting to note that the number of formalisms tend to fluctuate (just like other things in the human society). Before the emergence of the UML there were a huge number of formalisms which
where then merged. However, over the recent years, many domain specific approaches have been emerging (see further the section on domain specific modeling languages).

Bran Selic pointed out that the reason that the UML came about in the 90s was a pressure from users for whom the situation was becoming difficult with such a multitude of various OO formalisms with associated tools. This made tool investments as well as training of users difficult to plan and risky. Since the standardization of the UML, including UML2, the situation has diverged again, with domain specific modeling languages, created as UML2-profiles, or using other meta-modeling approaches. In the model-transformation community, the diversity of formalisms is often seen as less of a problem. Given the definitions of the formalisms (meta-models), concept mappings, and model-transformations techniques, it is possible to convert models in one formalism into models in other formalisms. This standpoint is supported by the fact that developers would like to use their preferred languages and tools. On the other hand, architecture descriptions are at the core of system development and should be used and understood by a wide variety of stakeholders. There is then clearly some value if unification is possible to enhance understanding, training and tooling.

This discussion indicates that the means for integration are important to standardize, but also that certain special viewpoints and the formalisms they employ are central to agree on. The importance of a common language for communication, among engineers, and among tools was emphasized for architecture descriptions, with the need to anchor files, formats and meaning.

Standards could be important for both approaches. Good standards provide stable definitions for tools and technology development. Providing mature standards at the right level is a challenge. Attempts for data exchange standards, such as XMI/DI and STEP, clearly illustrate this, and seem to imply that standards should instead target other levels when domains and technologies are still evolving.

Creating domain-specific modeling languages - which approach is better?
Following the OMG standardization approach, domain-specific modeling languages (DSLs) are created using the UML and profiling. This provides the possibility to use UML tools that support profiles and leaves the legacy of UML in the DSL (this may be beneficial or not depending on the users and the DSL model semantics). Creating DSLs using meta-modeling tools may instead use standards for model transformations and for meta-meta-modeling, and possibly a greater degree of freedom in creating user desired views. The strength of DSLs in general are that they can be tailored for a particular project and product, with quick implementation and full control; this in contrast with years for standardization and compromising.

How does one go about choosing the right language?
This question was asked by a participant of the workshop. The panel gave the following answers:

~ Which questions do you want to ask using the model?
~ What are the domain constraints and aspects you want to deal with?
~ Is the formalism/tool being taught, are there people that know it?
~ What is the interoperability of the technology?

What kind of research should be addressed?
This point yielded an interesting discussion and the following proposals for what research is needed.

Independent evaluation of technology is lacking. New formalisms and tools are often claimed to provide certain advantages in terms of development time and quality. Do the technologies add value to the organization? How effective are certain notations compared to others? There are few evaluations on these topics and while they are difficult, they need to be addressed!
There is a lack of a sound theoretical foundation for model-driven engineering. While there are standard text books for programming languages/compilers and for certain aspects of model-based development, there is a shortage of a holistic approach for embedded systems. Parts of such a foundation should include core concepts, abstractions, modeling languages and transformations in the context of embedded systems. Such a foundation should include technology, process and people perspectives. A remaining challenge is that of providing tools that are intuitive, support increased creativity, yet provide sufficient formality when needed - and these challenges of course go beyond embedded systems.

All researchers present were encouraged to contribute to a new book on this topic! "Model based engineering: Here are the five ways, theory, and ways of doing it!"

Finally, the panel suggested means for strengthening exchange of ideas and experiences among academia and researchers. This is fully in line with ICES goals!

Work on a white paper is underway…

... Watch this space!
says Martin Törngren, ICES Director

ICES Research Students share their thoughts at the ADL Workshop

Authored by Martin Törngren together with contributions from the Speakers and Panelists:
Bran Selic, Sebastien Gerard, Mamoun Filali, Jakob Axelsson, Juha-Pekka Tolvanen, Henrik Lönn, Sandro Penolazzi, Alberto Sangiovanni Vincentelli, Mattias Rehnmann, Kurt-Lennart Landbäck and Peter Fritzson.

References and further reading:


