Components in Embedded Software

Technical LEGO in LEGO land

Ir. P.N. Wouters MTD

Composing a system from reusable subsystems with a well-defined behaviour is a common design principle in most engineering disciplines. PCB's are built from standard electronic components, houses are built from architectural units, and even mathematical models are composed from sets of coherent equations. Likewise, software is often built by assembling pieces of software.
However, reusability has always been a weak point of software engineering. In view of this, the introduction of Component Based Development and component technologies in the last few years is promising. Especially in the field of information systems, these technologies are successfully adopted. How do these trends relate to current embedded software practice? What is new about them, and what can be expected from the future?

Reusability

In order to significantly reduce the throughput time of software projects, for many years the software engineering community is trying to increase the degree of reuse. Although object orientation has been a big step in the right direction, it has not had the effect that many people expected from it. In my opinion there are three major reasons:

Lack of standardisation in system architecture

The almost unbounded variety of system architectures makes it very difficult to built components that can be reused within another system. Without standardisation in component interfaces, general mechanisms, parallelism, etc., a substantial amount of effort is required to port a component from one system to another.

Lack of software engineering skills to built reusable components

In order to be reusable, component interfaces should be simple, generic, and abstract. In practice, component interfaces are often directed towards the specific use of a component. This leads to complex and user-specific interfaces, and lack of encapsulation.

Lack of organisational commitment to invest in developing reusable software

In a lot of organisations, time-to-market is prevalent in software development. When product development is not accompanied by a strong drive to adhere to a component based architecture, reuse is out of the question.

These arguments hold for all kinds of software development projects. In embedded software, the problems are even larger. Because of the large number of constraints on embedded software, reusability is hard to accomplish. Timing aspects, hardware dependencies, and resource constraints make it almost impossible to develop off-the-shelf reusable software. When, however, the concept of weak coupling and strong binding is applied accurately, traditional software development methods may still lead to a fair amount of reuse.

Component-Based Development

The latest solution to the problem of reusability is Component Based Development (CBD) by means of component technologies. A component may be defined as "a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties" (Szyperski). There is a fundamental difference between Component-Based Development (CBD) and component technologies. CBD mainly describes a philosophy: systems can be built by assembling well-defined components that provide a clean interface to their functionality. It is just like LEGO: once you have a set of standard components, a variety of systems can be built with it.

A component technology may be seen as a framework that enables CBD. In general, component technologies consist of an architecture and a set of application programming interfaces (API) within which one can define software components that can be combined to form an application. There is a basic set of services that is provided by most component technologies, including

Event handling
: Components may communicate with other components and the system itself by means of messages.

Persistence
: Components may need to store non-volatile information.

Development support
: components may be customized to be used within development tools (introspection).

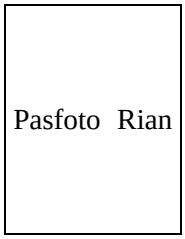
In general, CBD is expected to increase the amount of reuse. Although several component technologies are competing to become a De facto standard, components are to a certain extend compatible between those technologies. Furthermore, standardisation makes components marketable. The number of components on the market will rapidly grow. Time-to-market can significantly be reduced when 3rd party components can be integrated into customized applications.

On the other hand, building reusable components is not an easy job. A high level of abstraction is required and transforming a custom component into a reusable one is not a straightforward process. Moreover, if organisations are not willing to invest in producing reusable software, a component-based system might still end up in a set of communicating components that cannot be reused because of their strong dependencies.

The question arises whether CBD will solve the problem of reusability in embedded software. Going back to our LEGO metaphor, embedded software is the technical LEGO of software engineering. This implies that a lot of components and component interfaces in embedded software are substantially more complex than those in information systems. Moreover, because the variety of components is much larger, such a component can be reused in a limited number of systems. Therefore, it can be expected that CBD will only increase the amount of reuse if a meaningful set of components can be defined, and investments are done to adhere to this component architecture.

Future directions

In the next few years, the maturity level of CBD and its enabling technologies will grow. component based system development will become easier, as more and more components will be commercially available. However, for embedded software it will take quite some time before high-quality components will be available. Moreover, development of reusable components is not guaranteed when CBD is applied. Whether components can be reused or not is still determined by the development process and the skills of the developers.



Rian Wouters is working as a Senior Technical Designer at Alert Automation Services b.v., a Software House specialized in training, detachment, and consultancy. Before joining AAS, he completed the OOTI programme at Eindhoven University of Technology.