SOFTWARE-HARDWARE COMPLEXES: TOWARDS FLEXIBLE BORDERS

Position Statement

Franz J. Rammig
Heinz Nixdorf Institute
University of Paderborn
Paderborn, Germany
franz@upb.de

For a long time it was relative clear what the terms hardware and software are standing for. Hardware was just the platform for the execution of software. This idea of a platform later was adapted by system software as well. From this point of view the idea of platform-driven design did emerge. The previously used term "Meet in the Middle", created by Hugo de Man describes this idea very clearly. It is a mixed bottom-up and top-down design method, meeting in the middle. Given some design target concurrently two design activities take place. One (bottom-up) tries to develop a fixed platform that is adapted to the specific needs of the application. "Adapted" means in most cases that a broader class of platforms is just customized towards the specific needs. At the same time an initial abstract solution of the problem is stepwise refined until it can be directly executed on the customized platform.

This approach today is widely accepted, at least in academic environments. There are even formal methods available to support such an approach like formal refinement calculi like the B-method or various process algebraic techniques like π-calculus.

Modelling such complexes can follow traditional approaches. OO techniques, e.g. UML did prove to be well suited to support such platform-oriented design methods. Big effort has been spent to study UML in the context of system engineering including hardware specific aspects. During a design cycle which follows the MDA principles hardware parts can be refined to more hardware oriented modelling languages like System-C or System-Verilog. Looking at any kind of platform just as a service providing entity unifies this design process. What has to be designed for the "Meet in

the Middle" approach is just a proper application programming interface (API). Whether such an API is given by a set of primitives (as in the case of operating systems, see Posix for example) or an instruction set is a minor distinction from the point of view modelling principles. The term "Transaction Level Simulation (TLS)" used in modern hardware design makes this unification of basic principles even more evident. Simple instructions are abstracted to more complex services provided by the hardware.

So it seems that we did achieve a certain agreement how to model mixed hardware software complexes. The situation changes, however, as soon as we allow dynamic reconfiguration at run-time.

If so we damage the cornerstone of platform-oriented design. There is no longer a clear distinction what is platform and what is application. Assume a micro-programmable processor where the micro-program can be altered during run-time. UCI's NISC (No Instruction Set Computer) approach can be seen as an attempt into this direction. Any kind of dynamically reconfigurable hardware (FPGAs or similar devices) may serve as another example. Again, the set of services offered may be altered at run-time.

A platform in such a scenario still is a virtual entity where applications can be deployed on. But now it becomes highly dynamic what set of services is offered, where such services are located, in which way these services are provided and implemented. Even the distinction between platform and applications becomes dynamic. In essence this means that now decisions that usually are made at design-time are deferred to run-time, i.e. have to be made by the system itself and automatically.

An approach to model such situations may be based on the assumption that both, the application software and the hierarchy of virtual platforms will be organized in a strict component-based way. This means that we assume a fabric of components with well-defined interfaces in between. Both, the application programs and the execution platform follow a reflective paradigm. They collect knowledge about their own state and these ones of objects they are connected with.

An application tries to require as little fixed private resources as possible (it virtually may have to pay for any kind of resources). Therefore it is interested in getting rid of as many components as possible by handing over them to the platform and just paying for using the services.

A platform is interested to collect as many components as possible as then it may charge usage fees to applications. On the other hand it has to pay for its private usage of resources as well. So a benefit exists for a platform only if it owns components that it can offer to a variety of applications. This model means that components that can be shared functionally between many applications should have a tendency to be handed over to the platform while

dedicated components will remain part of the respective applications. This makes evident that in this model a platform is rarely stable. It will adapt itself dynamically to requests by dynamically changing applications just by inclusion and exclusion of services. It will do the same internally, i.e. internally it becomes an application that treats lower levels of services as its platform.

This completely dynamic model of hardware-software complexes is fundamentally different from traditional approaches. Novel means of modelling, analysis, and synthesis will be requested to handle such systems in an adequate manner. It is the strong belief of the author that this kind of "dynamization" will happen, it already started to happen. So the question is not whether we have to deal with such situations but how to. The research community is requested to investigate potential approaches.