

ECBS'04 - Brno

Workshop on Model-Based Development (27th. May 2004.)

Position Paper : Peter Wolstenholme

For a fuller account, see the paper "Closing the Gap Between Models and Code" which will be presented in session B1 of the main conference..

It would be highly desirable for software to be developed in a more abstract manner than is currently the case. Ideally, users would specify the requirements in sufficient detail for no discretion to be left to the "programmers" who implement the coding. If this could be done, then these programmers could be replaced by software.

In practical terms, at least one level of intermediaries is needed between the users and the generators of the final system, and this level is supplied by system designers, or software engineers, able to develop the specification into a form suitable for its later processing.

A real problem arises in the construction of a detailed implementation, as very many questions of detail normally arise while coding, and these are conventionally resolved by the programmers. This is a significant problem with UML specifications, which are inherently too loose to constitute formal models. Can this difficulty be avoided? In the area of embedded systems, a technique has been developed of preparing specifications of the behaviour of the system, producing models which are totally complete representations of that behaviour. The models are platform independent, and in that sense quite abstract. They are of course based on finite state machines.

A system may very well require to have its behaviour controlled by several finite state machines, each dealing with a small group of functions, and acting quasi-concurrently (within the limits of time resolution imposed by the host environment, and also by rules ensuring that the entire system is deterministic).

If an embedded system is designed, it is inevitable that unforeseen events will be encountered. A design of its behaviour based on finite state machines is very much less likely to crash or freeze, than a traditionally-coded design, in such circumstances.

The models created by using StateWORKS are necessarily so complete and totally accurate that they can be loaded into a target system and executed with no further transformation. Clearly, as they only apply to the system behaviour, so as to eliminate large amounts of "control flow" coding, much software will often need to be generated by other means for functions such as user interfaces, special numerical analysis of data, data-base structures etc. However, this second category of software is comparatively easy

to write and test, and can be very reliable. By arranging for it to be linked to the real-time data base of StateWORKS one discovers a ready-made framework for the entire project.

StateWORKS is not just an academic research concept: it has been applied to a wide variety of projects, and all the practical details around it have been worked out. The platform independent models of StateWORKS can be tested by the designer in the development environment, and will then show exactly the same behaviour when running in the target system. Furthermore, they permit ideas to be transferred to subsequent projects, even when the hardware characteristics of the system have changed.

I suggest that it is misguided to seek methods of generating code, such as C++ source code, from models, because such code is subject to ill-advised modification at late stages of a project, and that can be very dangerous for reliability.

There is of course considerable reluctance on the part of software practitioners to develop in terms of models rather than of code, and it will take a major effort to bring them to accept to work at a higher level of abstraction. (I have seen this before: 40 years ago I knew colleagues skilled in programming control-system software in octal without comments; they were reluctant to move to a symbolic assembler, and ridiculed the idea of high level languages!)

Whether or not StateWORKS itself is ever widely deployed as a software-generation tool, I put it to you that the concepts it embodies, or very similar ones, are crucial if the aim of generating reliable software from models is to be achieved.