ENHANCING A REAL-TIME DISTRIBUTED COMPUTING COMPONENT MODEL THROUGH CROSS-FERTILIZATION

Position Statement

K. H. (Kane) Kim
DREAM Lab., EECS Dept.
University of California, Irvine, CA, 92697-2625 USA
khkim@uci.edu

The need for overall optimization of software-hardware complexes has been there throughout the history of computer applications. However, the need for significant improvement in the techniques for achieving it has become very acute as the growth of the embedded computing application field has been in an accelerating mode since mid-1990's. As a natural consequence of it, the desire to have unified modeling approaches that are effectively applicable to both hardware systems and software systems is as strong as ever.

Often software design activities and hardware design activities proceed largely independently without joint analysis and optimization, i.e., without disturbing each other, until the integration phase is reached. In such situations, software designers can benefit considerably from the availability of reliable models of hardware under development at the early stage. Similarly, hardware designers can benefit from the availability of early abstract but reliable models of software under development. Both designers are bound to wonder about the essential differences between hardware system modeling techniques and software system modeling techniques and about the aspects common to both types of modeling techniques.

This author, who is a researcher dealing with techniques for software system modeling, feels that the following complimentary relationship exists between the recent work on software system modeling and that on hardware system modeling.

- Software designs used to involve producing multiple layers of abstraction. In general, the number of layers defined during software

(Kane) Kim, K.H, 2007, in IFIP International Federation for Information Processing, Volume 231, Embedded System Design: Topics, Techniques and Trends, eds. A. Rettberg, Zanella, M., Dömer, R., Gerstlauer, A., Rammig, F., (Boston: Springer), pp. 427–430.

design has been significantly larger than that defined during hardware design. Therefore, in developing software modeling techniques, the ability for handling multiple layers of abstraction has been emphasized.

- On the other hand, the software modeling research has involved timing specifications at coarse levels in comparison to the hardware modeling research. Therefore, it seems worthwhile for software modeling researchers to attempt to learn from the techniques developed for timing specifications as parts of hardware models and use that knowledge to enhance software modeling techniques.
- In addition, the software modeling research has involved concurrency exploitation and specifications at coarse levels in comparison to the hardware modeling research. Again, it seems worthwhile for software modeling researchers to attempt to learn from the techniques developed for concurrency specifications as parts of hardware models and use that knowledge to enhance software-modeling techniques without damaging the strong ability to deal with multiple layers of abstraction.

One of the major challenges that researchers in software system modeling have faced is to establish techniques effective in modeling of real-time networked computing software. In particular, the modeling techniques that help the developers in *safety check*, i.e., analyzing software designs to check about the possibility of violating action timing requirements inherent in the given applications, have been wanted. Research in this important area has been advancing rather slowly.

The ease of such safety analysis depends heavily on the structure of the real-time networked computing software. Yet, the art of structuring real-time networked computing software that eases such safety check while enabling efficient and flexible design of high-performance software has remained immature. Here several potentially conflicting desiderata exist. First of all, the structuring approach must enable maximal exploitation of concurrency since otherwise, it will lead to designs of low-performance software of which inability to meet certain stringent timing requirements in certain applications is quite obvious. Yet, careless exploitation of concurrency leads to designs that are hard to analyze. Secondly, timing specification must be done in terms that can be supported by the execution engine consisting of hardware, operating system kernel, and middleware. Without defining and realizing new-generation execution engines, the timing specifications are bound to be in low-level terms and it is hard to check whether such specifications lead to efficient safe overall designs or not.

This author and his collaborators have been enhancing the structuring scheme called the *Time-triggered Message-triggered Object* (TMO) scheme as well as associated execution engine models and software engineering

tools in the past 15 years (Kim 1997, Kim 2000, Kim et al. 2005). TMO is a programming model of real-time distributed computing software components. The TMO scheme is intended to enable maximal exploitation of concurrency while maintaining a high degree of analyzability of the real-time distributed computing software. At present the TMO research community is seriously interested in learning from the hardware modeling research community about additional potential mechanisms for concurrency and timing specification.

Attempts have been made from the beginning to incorporate into TMO a practically sufficient set of mechanisms for expressing timing constraints without violating the fundamental modular structuring principle underlying the object / component structuring schemes. So far, all reported experiments and experiences seem to indicate that the set of mechanisms is practically sufficient but it is too early to conclude as such.

The essence of the modeling power of the TMO scheme is as follows (details in Kim 1997, Kim 2000, Kim et al. 2005):

- *Active components*, i.e., components which have hearts or internal energy sources and thus may be of hardware type or software + physical_ or virtual_hardware type, can be modeled as TMOs because of the availability of the *time-triggered method*, also called the *spontaneous method* (SpM) mechanism in TMO.
- Interconnections among components can be modeled by logical multicast channels called *Real-time Multicast and Memory-replication Channels* (RMMCs) or *service requests* which are sent from TMOs to TMOs and may be of one-way communication type, two-way non-blocking communication type, or two-way blocking communication type.
- All real time references in TMO are references to *global time* that is commonly accessible from all distributed computing sites and may be maintained in a decentralized fashion. Action timings of components can thus be specified in terms of global time. Clock-driven actions and periodic actions can be specified as parts of the time-triggered method specifications in TMO.
- Signal delays incurred over interconnection links can be represented by use of *official release times* (ORTs) of messages (RMMC messages or service requests) or combinations of production time-stamps and ORTs.

Again, what can be learned from hardware modeling techniques to further enhance the TMO scheme toward supporting more efficient types of networked real-time embedded computing systems is a question that we plan to address more intensively in coming years.

REFERENCES

Kim, K.H., 1997, Object structures for real-time systems and simulators, *IEEE Computer*, **30**(8): 62 – 70.

Kim, K.H., 2000, APIs for real-time distributed object programming, *IEEE Computer*, **33**(6): 72 – 80.

Kim, K.H., Li, Y., Liu, S., Kim, M.H., Kim, D.H., 2005, RMMC programming model and support execution engine in the TMO programming scheme, *in Proc. of the 8th IEEE Int'l Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, pp. 34–43, Seattle, USA.