Towards a Kernel Language for Heterogenous Computing

Dag Björklund Johan Lilius
dag.bjorklund@abo.fi johan.lilius@abo.fi
Turku Centre for Computer Science (TUCS)
Lemminkäisenkatu 14A, FIN-20520 Turku, Finland

What is characteristic of modern embedded systems like mobile phones, multimedia terminals, etc. is that their design requires several different description techniques: The radio-frequency part of a mobile phone is designed using analog techniques, the signal processing part can be described using synchronous data-flow, while the protocol stack uses an extended finite state machine based description model. This heterogeneity poses a challenge to embedded system design methodologies, and has resulted in a search for a System Level Design Language (SLDL) for describing both software and hardware.

We believe that to obtain a good SLDL one needs to first understand what the combination of models of computation means. To this end we are developing a kernel language in which it is possible to use different models of computation. The main contributions of this work are: (1) a common set of concepts that form the basis of the kernel language, (2) a formally defined operational semantics, which also makes it possible to verify designs using e.g. model-checking, (3) the explicit use of atomicity and, (4) the introduction of the notion of execution policy.

Several system description languages have been proposed, e.g. UML [4] and SystemC [3]. They make very different assumptions about the final implementation technology, e.g. UML is meant for modelling software, while SystemC addresses the hardware design domain. What is interesting is that many language constructs are common to these languages, but their semantics differ due to the computational model. E.g. parallelism in UML has a very different semantics from parallelism in SystemC.

It is this interaction between language constructs and models of computation that is the focus of our research. We have identified a number of concepts that are common, but are interpreted differently given a different computational model. The language has syntactic entities to represent these concepts and through the operational semantics we give each concept a exact mathematical meaning.

The basic concept in our language is the **state**. State is seldom explicit in programming languages like VHDL or ESTEREL [1] but is a common concept in many modeling languages like UML or Harel's Statecharts [2]. Several states may be active at the same time (**concurrency**). The activities in concurrent states should be run in parallel. However, the parallelism is interpreted differently depending on the **execution policy** in effect in the current scope. A novelty in our language is that we make **atomicity** explicit. Atomicity defines what the smallest observable state change is. The programmer is thus free to define the level of atomicity as he pleases. In synchronous languages like Esterel, atomicity encompasses the whole program, so that the internal workings of the program are not observable. This is often problematic because the programmer is forced to think of the whole system as a monolothic state machine. Other key language constructs are **interrupts**, **coroutines** and **communication**.

We have explicitly separated the execution policy from the semantics of the statements. It is our claim that we can model most models of computation using suitable combinations of atomicity, communication structure and execution policy. E.g. Harel's statechart semantics is obtained by using broadcasting as communication discipline, by using step semantics for interpreting concurrency, and by making the top state atomic. As an other example, we can view a UML collaboration diagram, where some nodes are refined using state machines, as a synchronous dataflow system, again by selecting a suitable execution policy and level of atomicity.

References

[1] G. Berry and G. Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[2] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions of Software Engineering and Methodology*, 5(4) Oct 1996.

[3] S. Y. Liao. Towards a new standard for system-level design. In *Proceedings of the Eighth International Workshop on Hardware/Software Codesign*, San Diego, CA, USA, May 2000.

[4] Object Management Group. *Unified Modeling Language Specification 1.3*, chapter 3.74.