Playing with Verification, Planning and Aspects: Unusual Methods for Running Scenario-Based Programs

(Abstract of FLoC Keynote Talk)

David Harel

The Weizmann Institute of Science
dharel@weizmann.ac.il

The talk first describes briefly the inter-object, scenario-based approach to programming that I've been working on with colleagues and students for the last eight years. It starts with the 1998 advent of the language of *live sequence charts*, or LSCs, jointly with Werner Damm. LSCs extend message sequence charts, or sequence diagrams with modalities, and thus can express possible, mandatory, forbidden and fragmented scenarios of behavior. Following this, together with my ex-PhD student Rami Marelly, we extended the language quite significantly, adding time, symbolic instances, forbidden elements and more. We also developed a convenient method for programming LSCs directly from a GUI, called *play-in*. The highlight of the work with Marelly, however, is *play-out*, a method for executing LSC specifications, and it is play-out that serves to turn the entire approach into a means for actually programming a system, and not just one for eliciting requirements. The entire approach we then implemented in a tool called the *Play-Engine*.

There is something very declarative about LSCs, and something akin to the execution mechanisms of constraint programming and logic programming in the play-out method, but for various reasons it is more subtle and therefore was considerably difficult to work out. Still, the basic play-out mechanism deals with the nondeterminism inherent in the LSC language in a naive way, just like the way most software development tools that execute models deal with racing conditions: it simply chooses one of the possible next things to do and does it. Of course, this may lead to violations of the constraints present in the LSC specification. Had another path been taken this could perhaps have been avoided.

The present talk discusses three more sophisticated ways to run LSCs, or, more generally, to execute scenario based models and programs. Interestingly, and somewhat unusually, the three methods use ideas from three quite separate fields of computer science: verification, AI and programming methods.

The first method, *smart play-out*, which was developed with ex-PhD student Hillel Kugler, translates the problem of finding a full non-violating superstep (i.e., a sequence of actions that the system takes in response to an external event) into a verification problem, and then employs model-checking to solve it. The resulting superstep is then promptly executed in a way that is transparent to

T. Ball and R.B. Jones (Eds.): CAV 2006, LNCS 4144, pp. 3–4, 2006.

the user. We thus, surprisingly, use hard-core verification not to prove properties of programs or check consistency, etc., but to run programs.

The second method, *planned play-out*, under development with MSc student Itai Segall, uses AI-style planning algorithms (we use one called Graphplan) to do essentially the same. The advantage over smart play-out is in the fact that we can find more than one possible superstep, and we have set-up a sort of user-guided exploration mechanism to allow the user to navigate among possibilities during execution.

While both these methods follow the original play-out mechanism in being an interpreter approach to execution, the third method is a compilation one. With PhD student Shahar Maoz, we exploit the similarities between aspect-oriented programming and the inter-object nature of LSCs, and have worked out a scheme for compiling LSCs directly into AspectJ. We use what we call scenario aspects to coordinate the simultaneous monitoring and direct execution of the LSCs.

All three methods still require lots of work. None work yet on the full extended LSC language, with time and symbolic instances being the main features that cause difficulties. There is also a lot of research still to be done in refining and strengthening the methods to scale up to large systems, and of course the jury is still not in on which of these will serve to be the best, and on whether there are other ideas for executing inter-object scenario-based programs. However, given our own excitement about the general approach, and the feedback we have been receiving, the topic seems to be deserving of the efforts needed.