Physical Programming: Beyond Mere Logic
(Invited Talk)

Bran Selic

Rational Software Inc., Canada
bselic@rational.com

Abstract. Plato believed in a "pure" reality, where ideas existed in their perfection into eternity. What we perceive as reality, he claimed, is merely a flawed shadow of this ideal world. Many mathematicians find this view appealing since it is precisely this universe of ideas that is the subject of their exploration and discovery. The computer, and more specifically, software, seem perfectly suited to this viewpoint. They allow us to create our own reality, one in which we can simply ignore the underlying physics, forget the tyranny of inertial mass, the drudgery of dealing with machinery that leaks or parts that do not quite fit. But, can we? Even in the ideal world with infinite resources, we have discovered that there are limits to computability. However, the situation with computers and software is much more dire than mere limits on what can be computed. As computers today play an indispensable part of our daily lives we find that more and more of the software in them needs to interact with the physical world. Unfortunately, the current generation of software technologies and practices are constructed around the old Platonic ideal. Standard wisdom in designing software encourages us to ignore the underlying technological platform Ð after all, it is likely to change in a few years anyway Ð and focus exclusively on the program "logic". However, when physical distribution enters the picture, we find that mundane things such as transmission delays or component failures may have a major impact on that logic. The need to deal with this kind of raw physical "stuff" out of which the software is constructed has been relegated to highly specialised areas, such as real-time programming. The result is that we are singularly unprepared for the coming new generation of Internet-based software. Even languages that were nominally designed for this environment, such as Java, are lacking in this regard. For example, it has no facility to formally express that a communication between two remote parts must be performed within a specified time interval. In this talk, we first justify the need to account for the physical aspects when doing software design. We then describe a conceptual framework that allows us to formally specify and reason about such aspects. In particular, it requires that we significantly expand the concept of *type* as currently defined in software theory.

References

1. B. Selic, "A generic framework for modeling resources with UML", *IEEE Computer*, June 2000.

H. Hussmann (Ed.): FASE 2001, LNCS 2029, p. 1, 2001.