ON THE DESIGN OF REACTIVE SYSTEMS

R. KURKI-SUONIO and T. KANKAANPÄÄ

Computer Systems Laboratory, Tampere University of Technology,
Box 527, SF-33101 Tampere, Finland

Dedicated to Peter Naur on the occasion of his 60th birthday

Abstract.

The notion of joint actions provides a framework in which the granularity of atomic actions can be refined in the design of concurrent systems. An example of a telephone exchange is elaborated to demonstrate the feasibility of this approach for reactive systems and to illustrate transformations that are justifiable in such a process. Particular problems arise when a refinement would allow new interleavings of semantically relevant events. The meaning of a reactive computation is specified in a way that makes this possible.


Keywords: formal specification, design methods, concurrency, reactive systems, joint actions, refinement of atomicity, statecharts, telephone exchange.

1. Introduction.

As expounded by Peter Naur [22], formalization is no replacement for informal insight in the development of computer programs. One area where formalisms and mathematical theories abound is concurrency. In spite of the richness of these theories, none of them seems satisfactory for making the associated problems manageable in practical design situations.

As opposed to systems that implement given input-output relations, the term reactive has been adopted [24] for systems that implement interactive behaviors with their environments. Embedded systems typically belong to this category. Concurrency is usually involved in such systems, but there are significant differences between reactive and non-reactive concurrency: a reactive system cannot be specified in isolation of its environment, a computation in a reactive system is typically nonterminating, and concurrent reactions to different
stimuli cannot be handled merely as a sequence of input-output computations. Although there is inherent complexity in most concurrent systems, it is by no means necessary for a design process to lead to designs that are difficult to analyze. In principle, the situation should not be different from sequential programming [11] in the sense that the design process could ensure the essential correctness properties of the result. A reasonable design also provides abstractions that correspond to the informal insight of the designer and allow one to envisage the resulting program as an implementation of these.

The problems in designing manageable systems are not the same as those in analyzing arbitrary ones. In concurrency, theories and formalisms mostly deal with the modelling and analysis of arbitrary systems, rather than with the design of well-structured ones. In order to impose a better structure, better abstractions of concurrency are needed. The granularity of atomic actions is a crucial factor in the sense that the characteristic problems of concurrency are associated with its degree. This suggests stepwise refinement where it is the atomicity that is refined. Such an approach allows interpreting a computation in the resulting system as an implementation of an abstract computation that has a coarse level of granularity. For a recent discussion on atomicity the reader is referred to [19].

The purpose of this paper is to demonstrate the feasibility of systematically refining atomicity in the design of reactive systems. Conceptually our approach is based on the notion of joint actions, that will be briefly introduced in the next section, and on temporal logic [21,24], which is the natural vehicle for reasoning about them. For a more detailed treatment of this background the reader is referred to [9]. The rest of the paper is devoted to a discussion of a non-trivial example that illustrates the approach, a simple telephone exchange. It will be demonstrated how this system can be specified by joint actions that prescribe its interactive behavior on a coarse level of atomicity. The example will also illustrate what it means for reactive systems that a refined action system is an implementation of the specification.

Concerning the treatment of the example it is to be noted that dealing with real time is outside the scope of this paper. Real-time requirements are therefore omitted from the specification. Although this does not preclude events that are triggered by timeout, the associated real-time constraints are not expressed.

2. Joint action systems.

A joint action system operates on a collection of objects with local states. For a given joint action system the collection of objects is fixed, i.e. no objects are dynamically created or destroyed during operation.

The units of execution are called joint actions, or just actions. Syntactically we shall use the following format: