

# Real Time: Further Misconceptions (or Half-Truths)

Reino Kurki-Suonio, Tampere University of Technology

**Since specification and design of real-time systems may involve both nontimed and timed levels of abstraction, the compatibility of these levels seems important.**

In his notable article,<sup>1</sup> Stankovic analyzed some common misconceptions about real-time computing. His analysis addressed the very notion of real-time computing and touched upon the applicability of concepts that have proven useful in nonreal-time modeling of reactive systems, such as interleaving models, non-determinism, and fairness. From his viewpoint, such concepts are no longer appropriate when real time is a concern, and a new set of abstractions must be devised.

Here, I address these issues from a somewhat different angle, from the viewpoint of theories and formal modeling. By analyzing some common sources of misunderstanding, I want to show that some common statements about real-time modeling are unjustified, or express only half-truths.

A potential source for misunderstanding is the restricted practical interpretation of theoretical notions. With limited experience in using theories of computation, we can easily mistake a theoretical construct for the reality of a program.

As pointed out by Stankovic, the special role of time in real-time systems easily leads to incompatibilities between nonreal-time and real-time models. There are approaches where the transition from nontimed to timed models is smooth, however, and where the above theoretical notions are also meaningful in the presence of metric time. Since specification and design of real-time systems may involve both nontimed and timed levels of abstraction, the compatibility of these levels seems important.

## Models and reality

Computing is a young science. It is an applied science, expected to respond rapidly to new needs that arise in practice. Our concepts, definitions, and reasoning not only originate in the reality that computing practice addresses, but are also intimately tied to it. Therefore, it is often difficult to know whether one is speaking of reality or models thereof.

To be more specific, when discussing a program, do we mean the real phenomena that arise from its execution, or some formal model of them? Are we discussing the properties of a program in isolation, or do we include what happens in its environment? In using any theory, such distinctions are important, and failing to make them is bound to cause misunderstanding. The distinctions are especially important in sit-

An earlier version of this article was presented at the Fifth Euromicro Workshop on Real-Time Systems in Oulu, Finland, in June 1993.



uations where informal reasoning is insufficient, either for complexity reasons or because of the application's criticality.

Every theory is an abstraction that omits some aspects of reality. Which aspects can be omitted depends on what the theory is intended for. In computing, this is not always understood, and we frequently see the misguided attitude that the closer a theory is to reality, the greater its usefulness. In concurrent and distributed computing, such arguments may be used to defend new formalisms without considering sufficiently whether the described properties are significant in the specification and design of software. When comparing theories, we should keep in mind the well-known maxim attributed to Albert Einstein: "A theory should be as simple as possible, but not any simpler."

In the analysis of potential misunderstandings, we start with the very notion of real-time systems.

**"Real-time systems are a well-defined class of systems."** By questioning this assumption, we challenge the definitions that underlie any textbook or course on real-time systems. What we mean is that all criteria for distinguishing "real time" from "nonreal time" in the real world are artificial and depend on what we decide to consider "real time."

We can, however, distinguish between theories and models that deal with real-time properties and those that do not. With this distinction, "real time" is essentially an attribute of theories and models, and there certainly exist situations in which nonreal-time models abstract away those properties that we find the most important to reason about.

Just as there are different kinds of theories in physics, like Newtonian mechanics, the theory of relativity, quantum mechanics, and so on, there are different theories of computation. These include *transformational theories*,<sup>2</sup> wherein computations terminate and transform input to output; *reactive theories*,<sup>3</sup> wherein computations are possibly nonterminating sequences of interactions; and *real-time theories*, in which computations still lack a commonly accepted formal characterization. With the so-called synchrony hy-

**Nondeterminism:** A situation with some freedom for the outcome of an operation, or for the selection of the next operation to be executed. Unlike probabilistic situations, no information is assumed to be available about how the actual choice is made. Some programming languages, like Ada, have nondeterministic constructs that reflect uncertainties in concurrency and scheduling.

**Interleaving:** The theoretical assumption that there exists a global ordering in which all operations — including concurrent processes — are executed.

pothesis — that is, the assumption that a system is always infinitely fast in comparison to its environment — reactive behaviors reduce to sequences of transformational steps.<sup>4</sup> Correspondingly, real-time theories can be obtained by appending time to nonreal-time theories (see, for example, de Bakker et al.<sup>5</sup>) or by taking an inherently real-time approach.<sup>6</sup>

A physicist does not ask whether a particular reality is Newtonian, relativistic, or quantum mechanical but instead selects the theory that is most appropriate for the level of inspection at hand. Why then do we ask for the definition of real-time systems, as if there were definite criteria for such definitions in reality? And why are we satisfied with definitions like the one given in the *Oxford Dictionary of Computing*:

*Real-Time System.* Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to the same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.

In spite of their deficiencies, informal definitions like these help us understand what we use real-time models for. The above definition, for instance, provides

intuition about why real-time properties cannot be abstracted away in some situations. Looking at it more carefully, however, we cannot think of any system that would be excluded by such a definition. Real time is unavoidably significant for every (real) system. However, the crucial question is not about reality itself, but whether we should not reason about a system using a theory in which real time has been abstracted away.

**"Real time requires determinism."**

This statement is often defended with arguments about predictability. On the same basis, programming languages like Ada are criticized for their nondeterministic constructs.

Nondeterminism is a property of models, not of reality. Reality may or may not be nondeterministic, but as far as programs are concerned, we can assume reality to be deterministic but incompletely known. If we accept this philosophical assumption, nondeterminism cannot possibly be implemented and belongs to the world of theories and models. It may be imitated by probabilities and pseudorandom numbers, but this misses the point.

What then is the use of nondeterminism? Basically, it reflects the lack, or the deliberate hiding, of information, and its usefulness can be justified as follows.

First, a reactive system continually interacts with its environment. Since a system cannot enforce restrictions on its environment, any reasonable model of the environment must be nondeterministic. If a transformational theory is satisfactory (that is, a model wherein each stimulus is individually mapped into an appropriate response), a deterministic model may be sufficient. If a truly reactive model is needed (that is, the joint behaviors of the system and its environment are described), there is no way for the model to avoid nondeterminism.

Second, nondeterminism in a model may reflect potential concurrency in a distributed implementation or any sources of uncertainty in timing. In this case, an attempt to avoid nondeterminism would only increase the complexity of the model.

Third, models are not only used for single implementations of programs, but



also for specifications that are supposed to allow different kinds of implementations. Nondeterminism is then essential for the implementation freedom that a specification should leave.

Of course, nondeterminism should not allow computations that do not meet the intended real-time requirements. Therefore, real time may decrease the nondeterminism that would otherwise be acceptable.

Nondeterminism is especially useful in models used for incremental construction of specifications. An unspecified system is then understood as one with completely nondeterministic behavior, and the refinement steps gradually decrease this nondeterminism.<sup>7</sup>

**“Interleaving models are insufficient for real-time concurrency.”** Concurrency is often modeled by nondeterminism, requiring the execution of events to always be *interleaved* into some sequential order. A more realistic approach is so-called *true concurrency*, which imposes only a partial temporal ordering between the events in a computation.

There has been much debate about the two principles. True concurrency is intuitively appealing and is more truthful to reality. As I pointed out above, however, this criterion is not the one to judge a theory by.

A commonplace argument holds that we should discard interleaving because it is too simple and unrealistic for real-time concurrency. But this position is usually based on an unnecessarily restricted interpretation of interleaving models. The existence of several interleaving approaches to real time is evidence of this.<sup>5</sup>

Informally, we can understand an interleaving model as one that provides interleaved views on whatever the “real computations” are. For any two “real” events, an “interleaving observer” always sees them happen in some mutual order. If there is a causal-temporal relationship between them, this order is the same for all observers; otherwise, different observers may disagree. For a distributed system such disagreements are quite natural in practice. To deal with real time in an interleaving model we can associate

## Zeno's paradox according to Aristotle

Zeno's arguments about motion, which cause so much trouble to those who try to answer them, are four in number. The first asserts the non-existence of motion on the ground that that which is in locomotion must arrive at the half-way stage before it arrives at the goal. . . .

The second is the so-called Achilles, and it amounts to this, that in a race the quickest runner can never overtake the slowest, since the pursuer must first reach the point whence the pursued started, so that the slower must always hold a lead. . . . The result of the argument is that the slower is not overtaken; but it proceeds along the same lines as the bisection-argument (for in both a division of the space in a certain way leads to the result that the goal is not reached, though the Achilles goes further in that it affirms that even the runner most famed for his speed must fail in his pursuit of the slowest), so that the solution too must be the same. And the claim that that which holds a lead is never overtaken is false: it is not overtaken while it holds a lead; but it is overtaken nevertheless if it is granted that it traverses the *finite* [emphasis added] distance.

—Aristotle, *Physics*, 239b10-29

each event with a time stamp indicating the moment (or interval) in which it occurs. The real-time order of events is then indicated by these time stamps. For events that are logically independent and need no common resources, this order need not be the same as their interleaving order.

So far there is no consensus about the relative merits and drawbacks of interleaving and true concurrency. Interleaving models seem to have an advantage in simplicity of reasoning, and attempts to demonstrate that they are “simpler than what is possible” have not been convincing. This is not, however, a proof of their superiority.

## Logical properties and real time

Properties of (finite and infinite) behaviors are classified into *safety* and *liveness* properties,<sup>8</sup> which can be characterized as follows:

A property is a *safety property* if its violation can always be demonstrated by a finite initial part of the violating behavior.

A property is a *liveness property* if any finite behavior can be extended to one that satisfies the property.

Any logical property of behaviors is a combination of a safety property and a liveness property.

Of particular interest among the liveness properties of a model are *fairness properties*. Fairness always relates to some event or action that is possible in the model. Let  $e$  be such an event, and let  $g$  be some condition under which  $e$  could take place. (Usually  $g$  is taken as the enabling condition of  $e$ .) We can then define (strong) fairness as the antonym of unfairness:

A nonterminating behavior is *unfair* with respect to event  $e$  under condition  $g$  if  $g$  holds infinitely often in the behavior but, after a finite initial part,  $e$  no longer takes place in it.

In other words, if  $g$  is continually true, or always eventually becomes true, fairness (with respect to  $e$  under  $g$ ) guarantees that  $e$  will always eventually take place.

In constructive models of computation, fairness assumptions are a natural way to enforce liveness properties. The simplest liveness property implicitly assumed in several formalisms is *fundamental liveness*. Informally, this means that “if something can happen, then something must happen.” Obviously, this is fairness, with  $e$  standing for an arbitrary event and  $g$  for the condition that the execution has not terminated.



## Logical properties of temporal behaviors

**Safety:** Informally, a safety property is one that has the form "something bad never happens." By themselves, safety properties do not require a program to do anything. Like partial correctness requirements in sequential programs, they can only prevent a program from doing the wrong things.

**Liveness:** Informally, a liveness property is one that has the form "something good eventually happens." A simple example is the termination of a sequential computation. No matter how long a computation sequence is, we can always imagine it being extended to one that ends in the final state.

**Fairness:** Fairness properties are theoretical restrictions on nondeterminism, by which the continual omission of some operations can be forbidden. Enforcing only "eventual execution" of an operation, a fairness property allows any finite bias for the nondeterministic choice of operations.

In operational terms, we could imagine a general fairness scheduler that eventually schedules  $e$  for execution if  $g$  is repeatedly observed to be true. However, this is not generally implementable. Consider a fair semaphore, for instance, that allows each process to enter its critical region eventually. Any real scheduler that implements this fairness property must be based on some restricted policy (like first-come, first-serve queuing, for instance) that excludes some fair behaviors. Therefore, fairness is an abstraction that is useful as such only in models.

The definition of fairness usually relates to interleaving semantics. Further evidence about its "nonreal" character is then given by the fact that for a distributed system, different "interleaving observers" may disagree on whether a given behavior is fair or unfair. This does not, however, prevent us from considering fairness a useful notion for a theory.

We can see that adding a fairness assumption to a model does not add any safety properties to it. In other words, fairness assumptions affect only what happens "in the infinity." This leads to the following generalization:

A liveness property is a *fairness-like property* for a given model if it does not add any safety properties to it.

This characteristic of fairness is known as "feasibility" or "machine closure."

We might ask what significance the above notions have for real-time systems,

if any. In particular, the question applies to fairness properties, which may seem useless in the presence of real time. To analyze this situation, we must examine several interrelated statements about the role of such properties in real-time models.

**"Real-time properties are safety properties."** This statement might seem to follow from the definition of safety properties. For instance, the requirement that an event take place by a deadline cannot be satisfied by any extension of a behavior once the due moment of time has passed. From this perspective, a liveness property may approximate a real-time property only as long as metric time is not available.

As an example, consider the liveness property that event  $a$  is always eventually followed by event  $b$ . A corresponding real-time property would be the bounded response property that event  $a$  is always followed by  $b$  within a given time bound  $d$ . If this is violated in a behavior, there is a first occurrence of  $a$  for which this happens. If this  $a$  occurs at time  $t$ , then the violation can be observed from any prefix of the behavior where time passes  $t + d$ . Therefore, bounded response properties would seem to be safety properties.

The soft spot in this reasoning is whether time ever passes  $t + d$ . For real computations, we can reasonably take this for granted:

*Real-Time Liveness.* In each real-time be-

havior, time will eventually grow beyond any finite bound.

Of course, the question is not whether this is a reasonable liveness property for real computations, but whether we can assume the modeling formalism and logic to guarantee it.


A behavior that does not satisfy real-time liveness (RTL) is called a *Zeno behavior*. The term originates in the famous paradoxes of the Greek philosopher Zeno of Elea. One of these claims that Achilles can never catch a tortoise because once Achilles has covered the distance, the tortoise will have advanced some distance away from its previous position, and these steps can be repeated ad infinitum.

The role of Zeno behaviors in real-time theories can be compared to nonterminating computations in the theory of algorithms. Although it would be easy to exclude nontermination by a restricted programming formalism, this is seldom useful. Instead, termination is left as a property to be proved. Similarly, the possibility for Zeno behaviors can be prevented, but doing so seems to induce essential restrictions on the theory and the specification formalism.

As analyzed by Abadi and Lamport,<sup>9</sup> a real-time model is feasible in this respect if RTL is a fairness-like property for it. In this case, obeying the safety properties of the model can never lead to a "time block," wherein time can no longer proceed unboundedly. Obviously, only Zeno behaviors can arise from such a time block.

Under this feasibility assumption, the essential requirements for bounded response properties can indeed be given as safety properties, and the fairness-like RTL assumption then takes care of the rest. For proof techniques, ordinary real-time properties can be proved like safety properties, provided that the RTL property holds. In the terminology of Henzinger,<sup>10</sup> such properties are called *relative safety properties* with respect to RTL.

**"In real-time models, actions are triggered by time."** This statement reflects an operational view that requires some force to put actions into execution. In traditional theories of programs, this force is hidden in the implicit liveness assumption.



tion that the execution always proceeds until the program is finished. In nonreal-time theories of reactive systems, such a force can be provided by explicit liveness requirements, and fairness assumptions can be understood as an operational facility to exercise that force. In the absence of fairness assumptions, execution in an operational model could then halt arbitrarily at any point.

For real-time models, the above statement can be interpreted to mean that RTL is the only possible liveness assumption. After all, RTL is the only execution force that we can assume of the reality.

Abadi and Lamport<sup>9</sup> have shown that this is a reasonable approach in formalizing the facilities used in real-time programming. Fairness assumptions of nontimed levels of abstraction then turn into properties that are logically implied by RTL and the safety properties of the program. This also means that fairness assumptions can be considered nontimed abstractions of some real-time requirements.

Although this approach seems reasonable for real-time models of concrete programs, we may ask whether actions in operational specifications should always be triggered by real time. In fact, there is use also for more abstract real-time models with liveness properties other than those enforced by RTL.

Even in the presence of real time, we can still hold the view that fairness is the only facility to force actions into execution.<sup>7</sup> Obviously, the statement about time triggering all actions is then no longer true, and fairness assumptions are no longer just abstractions of real-time requirements. Instead of the intuitively natural assumption that the passing of time is the primary cause for anything to happen, the alternative philosophical principle is that time passes only because something happens.

**"Real-time deadlines cannot be imposed by fairness."** This statement reflects the truth that fairness with respect to an event cannot by itself impose a deadline for the event. Considering fairness the fundamental facility for enforcing liveness properties, one can also defend the opposite view that "no deadlines can be imposed without fairness."

Let us see how deadlines can be imposed in real-time models. Expressing a deadline requirement explicitly means that some liveness property is required to hold. However, in the absence of guaranteed non-Zenoness, this does not necessarily give what one would expect. For instance, together with the given safety requirements, this might make the specification identically false.

A common alternative is to use more constructive notions of real-time modeling. These do not, however, make the desired properties explicit. Priorities, for instance, impose deadlines only when used

---

**Instead of assuming  
that passing time  
causes things to  
happen, the alternative  
is to assume that time  
passes only because  
something happens.**

---

appropriately and with care. Timers (for expressing when actions must be executed) are a bit more explicit, but these too achieve the desired effects only with proper use. For example, if a chain of actions is to be executed by a deadline, it is the designer's responsibility to see that the actions in the chain will properly enable each other at due times.

As we have seen, fairness can be considered another facility for enforcing liveness properties operationally. Therefore, we can effectively use fairness to impose real-time deadlines by designing the model so that "competing" actions will be disabled in situations where the deadlines would otherwise be missed.

**"In real-time models, time prohibits some logical computations."** In shifting from nontimed to timed models, nondeterminism usually decreases; that is, some of the logically possible behaviors are excluded by the timings introduced. This follows, for instance, from the *maximal parallelism assumption* that no resources

are kept idle when there is something they can be used for. Obviously, this may exclude computations that would otherwise be possible, and this is what we can expect to happen in the design of a concrete implementation.

However, in models of real-time systems, this phenomenon is not necessary: Timings can be imposed so that no logical computations are excluded.<sup>7</sup> The provable properties are, of course, weaker, but since they are more stable under timing changes, this approach may be recommendable for a practical design methodology.

There certainly are situations in which the desired real-time properties cannot hold unless some logically possible computations are excluded from the model. How frequently this happens, and in which kinds of situations, would be important to know. Perhaps it is only these models that we should truly call real-time models.

**A**lthough reactivity is an attribute usually associated with real-time systems, the basic concepts in the well-developed theories of reactive systems are often considered irrelevant or totally inapplicable once real time is taken into consideration. In short, there is still much misunderstanding in discussions of real-time systems, and, as I point out, a smooth transition from nonreal-time to real-time modeling is possible.

We must return to the fundamental question of what we mean by real-time systems. In other words: What kinds of theories and models are needed to reason about those properties that we find essential in "real-time systems"? Saying that current theories of reactive systems are basically sufficient, since they extend naturally to real-time properties, would only be testimony of our having confused theory and reality.

In addition to discrete events, which are characteristic of reactive models, real-time systems often deal with continuous changes in their environments. This has led to research on so-called *hybrid systems*, where behaviors also include continuous state variables.<sup>11</sup> In principle, dealing with continuous variables does not require any new concepts beyond



those already available in theories of reactive systems. The important question is, however, not whether such an approach is possible, but whether it is useful for the purposes we need it for. We should ask, for instance, whether the essential properties of adaptive control systems can be conveniently described and reasoned about in these theories. ■

## References

1. J.A. Stankovic, "Misconceptions about Real-Time Computing: A Serious Problem for Next-Generation Systems," *Computer*, Vol. 21, No. 10, Oct. 1988, pp. 10-19.
2. E.W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.
3. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, New York, 1991.
4. D. Harel, "Statecharts: A Visual Approach to Complex Systems," *Science of Computer Programming*, Vol. 8, No. 3, June 1987, pp. 231-274.
5. *Real-Time: Theory in Practice*, J.W. de Bakker et al., eds., Springer-Verlag, Berlin and Heidelberg, LNCS 600, 1992.
6. Z. Chaochen, C.A.R. Hoare, and A.P. Ravn, "A Calculus of Durations," *Information Processing Letters*, Vol. 40, No. 5, Dec. 1991, pp. 269-276.
7. R. Kurki-Suonio, "Stepwise Design of Real-Time Systems," *IEEE Trans. Software Eng.*, Vol. 19, No. 1, Jan. 1993, pp. 56-69.
8. B. Alpern and F.B. Schneider, "Defining Liveness," *Information Processing Letters*, Vol. 21, No. 4, Oct. 1985, pp. 181-185.
9. M. Abadi and L. Lamport, "An Old-Fashioned Recipe for Real Time," in *Real-Time: Theory in Practice*, J.W. de Bakker et al., eds., Springer-Verlag, Berlin and Heidelberg, LNCS 600, 1992, pp. 1-27.
10. T.A. Henzinger, "Sooner is Safer than Later," *Information Processing Letters*, Vol. 43, No. 3, Sept. 1992, pp. 135-141.
11. *Selected Papers on Hybrid Systems*, L. Grossman et al., eds., Springer-Verlag, Berlin and Heidelberg, LNCS 736, 1993.

**Reino Kurki-Suonio** is a professor of computer science and engineering at Tampere University of Technology in Finland. With earlier research on formal grammars, parsing methods, and programming languages, his current research interests are the specification and design of distributed and reactive systems.

He received his PhD degree from the University of Helsinki in 1964, has held visiting positions at Carnegie Mellon and Stanford Universities, and is a member of the Finnish Academy of Sciences and Letters and of the Finnish Academy of Technology.

Readers can contact Kurki-Suonio at Tampere University of Technology, Software Systems Laboratory, PO Box 553, SF-33101, Tampere, Finland; e-mail rks@cs.tut.fi.

## Call for Book and Software Authors

You can enhance your professional prestige and earn substantial royalties by authoring a book or a software package. With over 350 titles in print, Artech House is a leading publisher of books and software for professional engineers and managers. We are seeking to publish new books on computer engineering and information systems management with an emphasis on computer communications and networking, high-performance computing, computer applications, object-oriented systems, VLSI design and test, expert systems, and software engineering.

We are currently seeking potential authors among engineers and managers who feel they can make a contribution to the literature in their areas of expertise. If you have published technical papers, conducted professional seminars or solved important real-world problems, you are an excellent candidate for authorship.

We invite you to submit your manuscript or software proposal for review. For a complete publications catalog and Author's Questionnaire please contact:

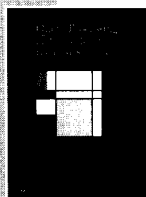
Mark Walsh, Acquisitions Editor  
685 Canton St., Norwood, MA 02062  
1-800-225-9977  
aqartech@world.std.com

**Artech House Publishers**



## QUERY PROCESSING IN PARALLEL RELATIONAL DATABASE SYSTEMS

edited by Hongjun Lu, Beng Chin Ooi,  
and Kian Lee Tan



Provides a detailed exploration of the latest developments in the design and implementation of high-performance parallel relational database systems. The first two chapters provide a concise overview of parallel database systems and the major technical issues involved in areas other than query processing.

Subsequent chapters delve into strategies to parallelize join processing in different system architectures, techniques employed to balance the load of skewed data, proposed query optimization strategies for parallel database systems, and innovative solutions to parallel database query processing and optimization problems.

392 pages, May 1994, Hardcover. ISBN 0-8186-5452-X.  
Catalog # 5452 — \$60.00 Members \$48.00

**IEEE COMPUTER SOCIETY PRESS**

▼ Call toll-free: 1-800-CS-BOOKS ▼

▼ Fax: (714) 821-4641 ▼