explicitly supported, as distinct from active objects? Are multi-threaded objects required? There are almost as many answers as there are programming technologies, CASE tools, and researchers in the field.

The second aspect is reasoning by humans. People need to identify the nature of the components in such a way that they can reason about their collaborative behaviour while in the process of roughing out appropriate design solutions. For this purpose, distinguishing the following *concepts* is important (use any name you like for them): concurrent processes; passive components that implement procedure or data abstractions; collaborating teams of either or both concurrent processes and passive components. The need to distinguish these concepts is completely independent of whether or not they are all implemented as, say, Smalltalk objects, or ObjectTime actors. Furthermore, distinguishing the following concepts is also important: objects; instances of classes. This is because any of concurrent processes, passive components, or teams may be instances of classes.

Use case maps make these distinctions by having different notations for “processes”, “objects” and “teams” and by allowing any of these component types to be instances of “classes”. These components appear in use case maps in a manner that is free of commitment to details like interfaces, messages, intercomponent contracts, intercomponent protocols, or IPC. This enables concurrency to be introduced into high-level design at any convenient time, in a lightweight fashion. There is no need to defer processes as details ( there often is a tendency to defer them simply because IPC requires very detailed commitments). The bottom line is that high-level design can include performance issues.

## 3.0 Conclusions

Use case maps provide hope for getting help with issue 1, are capable of identifying places where issue 2 must be supported in the implementation, and provide high-level ways of thinking about issue 3.

## References


[1] R.J.A. Buhr, R.S. Casselman, *Use Case Maps for Object-Oriented Systems*, Prentice Hall, 1996 (available October 95), 302 pages .

[2] R.J.A. Buhr, *Use Case Maps: A New Model to Bridge the Gap Between Requirements and Detailed Design*, contribution to the OOPSLA 95 Use Case Workshop, Austin, TX, October, 95.

# 2.0 Workshop Issues

## 2.1 Dealing With Time Constraints In Object-oriented Real-time Systems

This is the least resolved of the three workshop issues and is the one for which use case maps offer the most promise. The problem is to estimate response times along important stimulus-response paths through the system during design, to guide choices among design alternatives and to identify problems (such as too much overhead caused along the paths by the use of object-oriented techniques, e.g., inheritance and dynamic creation of objects, for the required level of performance). Without use case maps, there is no context for making such estimates. This tends to force a cut-and-try approach to achieving satisfactory performance. Stimulus-response paths don't exist as a first-class concept in the detailed design models, and so get lost in the intricate details of inheritance, changing visibility, dynamic binding, and so forth, at the detailed level.

With reference to Figure 2, I foresee tools that (among other things—[2]) provide support for entering timing estimates through multiple different design models, such that times may accumulated along use case paths to give performance estimates.

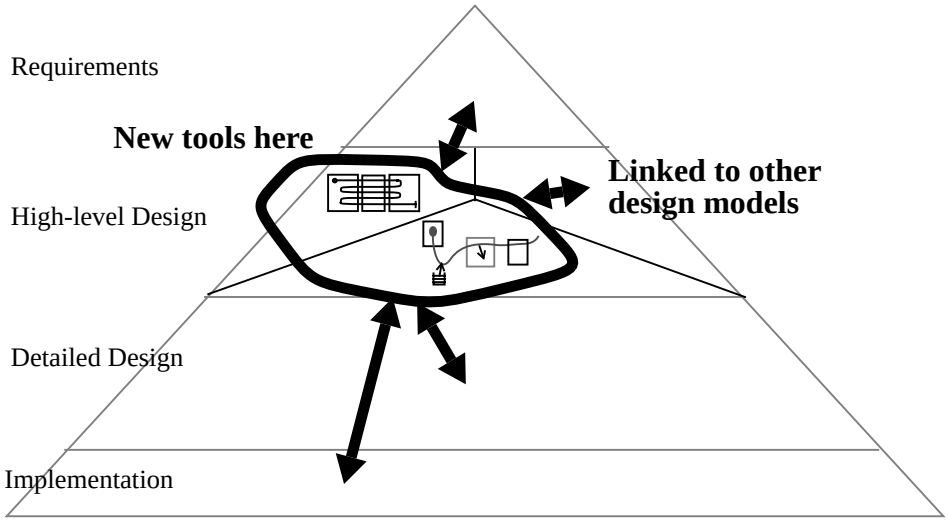


**Figure 2** Use case map tools linked to other models.

## 2.2 Reactive Behavior Models Versus Scheduled Behavior Models In Object-oriented Real-time Systems

Here, the term "reactive behaviour" indicates dispatch of processes on demand (due to the arrival of some event) and "scheduled behaviour" indicates dispatch of processes on some time schedule. Resolving this issue is part of the development of programming techniques that integrate object oriented and real time approaches. The implication is for use case maps, rather than the reverse (maps should distinguish processes that require different scheduling models).

## 2.3 Concurrent Behavior Models For Object-oriented Real-time Systems.

There are at least two aspects to this issue. One aspect is implementation: What concurrency models should be supported by detailed design or implementation tools? Some questions in relation to this aspect are as follows: Are actors sufficient? Do passive objects need to be

design concerns that we call *operation*, *manufacturing* and *assembly* (the names are intended to identify different design concerns, not to suggest anything about programming languages or code generation). Class relationship diagrams in the manufacturing domain describe “how it is constructed”. Use case maps cover both the operation and assembly domains to describe “how it works”. The operation domain is concerned with collaborative behaviour of teams of components. The assembly domain is concerned with how the teams are assembled at run time (i.e., how the members are created and made visible to each other). Use case maps cover both of these in an integrated manner at a high level of abstraction, as follows:

- The paths of use case maps provide both the routes for the progression of stimuli through a system of components (operation) and the loci for creating new components and moving them around to different places in the system at different times (assembly). Because the movement of components along paths is not different in kind from the progression of stimuli along paths, the model easily covers both operation and assembly in the same fixed map.
- The places to which components may move are called *slots*. Slots are boxes with dashed outlines that are analagous to positions in human organizations that exist independently of their occupancy or occupants.

The result is that we need only two models for high-level design, whether a system is structurally static or structurally dynamic. This enables us to come to grips with the performance implications of the pervasive structural dynamics that often exists in object-oriented programs. This is in strong contrast to the detailed-design level, where multiple models may be needed and descriptions may become very complex in structurally dynamic cases.

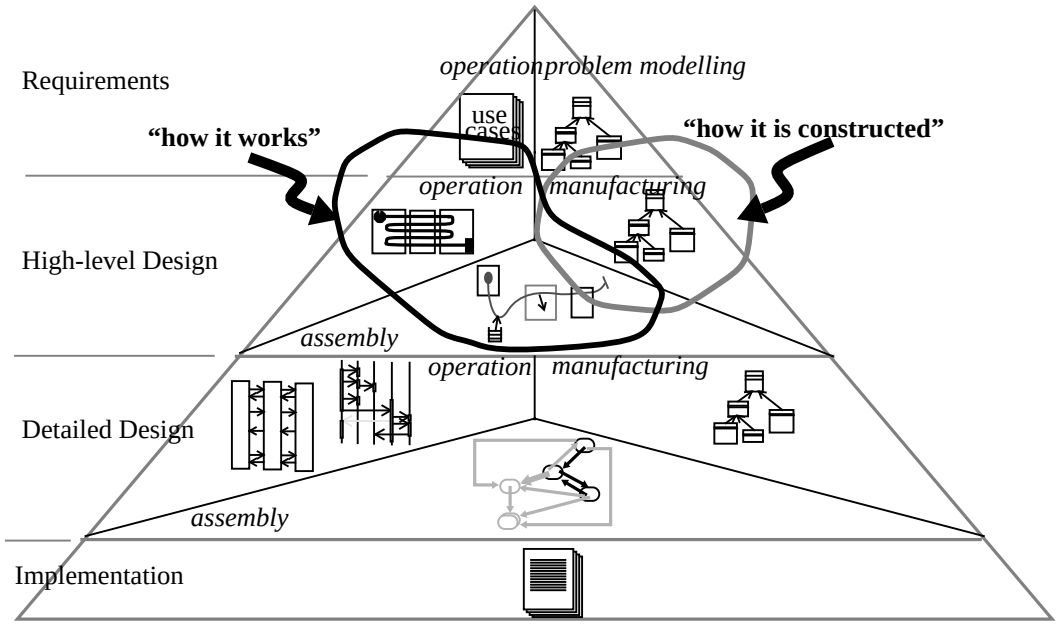


**Figure 1** A context for use case maps.

# Issues in Bringing Object Orientation and Real Time Together

**R.J.A. Buhr**
**Department of Systems and Computer Engineering**
**Carleton University, Ottawa, Canada**
**buhr@sce.carleton.ca**

## Abstract

**The areas of object orientation and real time are difficult to bring together, in part because established programming techniques in the two areas have a different focus. Real time techniques, such as prioritized processes (dispatched on demand or on a time schedule), focus on "how it works". Object-oriented techniques, such as classes and inheritance, focus on "how it is constructed". Although ways of integrating the programming techniques are emerging, there is still a lack of *design* models that can bring together all the factors in a way that will avoid the use of expensive cut-and-try methods to achieve required performance. This paper gives an overview of the problem and indicates how a new design model called *use case maps* can help.**

# 1.0 The Problem and a Solution

To find appropriate design models that can bring together all the factors, we need to stand back from the details and look for commonalities. A commonality is that both object-oriented programs and real time systems are "systems" at run time, that is, sets of collaborating components. We need a view of systems that gives equal importance to models of "how it works" and "how it is constructed". The key is to make the the two kinds of models first class, meaning each must be expressed in its own terms, not dependent on details of the other. Class hierarchies of object-oriented programs are first-class models of "how it is constructed". However, popular models of "how it works", like interaction sequence diagrams (a.k.a. message sequence charts), are not first-class models, because they depend on details (e.g., methods and messages) of other first-class models (e.g., class hierarchies).

Figure 1 positions a new model called *uses case maps* [1] as an appropriate high-level design partner for class relationship diagrams (see [2] for a quick overview of why we need yet another design model, the highlights of the use case map model, and a vision for the future development of the model). Use case maps stand well back from the details to get an overall view of behaviour patterns in cause-effect terms. Their paths trace the progression of causes and effects from points where stimuli occur, through the components of a system, to points where responses are felt.

The bottom line from the perspective of this workshop is that the paths in use case maps provide a common denominator for adding up diverse effects on performance. This is because the paths pinpoint where performance effects will be felt. For example, pipelines of processes along paths visibly identify where overheads will occur along paths due to process switching and IPC. As another example, objects and slots (see below) along paths visibly identify where overheads will occur along paths due to such things as traversal of inheritance relationships, and creation and destruction of objects.

All of this is possible because of the unique way in which use case maps bring together "how it works" and "how it is constructed". These two aspects are related to three domains of separate

# Issues in Bringing Object Orientation and Real Time Together

**R.J.A. Buhr,**

**SCE 95-**

***Contribution to the OOPSLA 95 Real Time Workshop***