# Interactive Animation needs Components

C. Huizing and B. Barenbrug

Dept. of Mathematics and Computing Science, Eindhoven University of Technology
Den Dolech 2, 5600 MB Eindhoven
Tel: +31-40-247 4120
Email: keesh,bartb@win.tue.nl

31st August, 1997

**Abstract**

Interactive animation is a natural application area for component-based systems. Design time is an obvious part of the development process and many components originate naturally form the objects that we see on screen.
**Keywords:** Components, interactive animation, rigid body dynamics.

**Workshop Goals:** Learning; discovering goals for theoretical investigation; sharing ideas about autonomous objects.

## 1 Background

In conventional animation techniques an animation is simply a sequence of pictures, *frames*, that are displayed in quick succession. For interactive animation and simulation, this is not enough. It is useful to consider the scene as composed of objects that each have a more or less autonomous behaviour. For this reason, we developed an obect-oriented animation system, supported by a an object-oriented scripting language. This GDP system (Generalized Display Processor) and language LOOKS are implemented at our department and are used for numerous applications, varying from algorithm animations to simulation of mechanical systems.

One of the key features of this animation system is the simulation of rigid body dynamics. Objects move according to the laws of physics, taking into account relevant magnitudes such as mass, momentum, force. As such, rigid body objects form a salient example of components with both a well-defined interface and an autonomous behaviour.

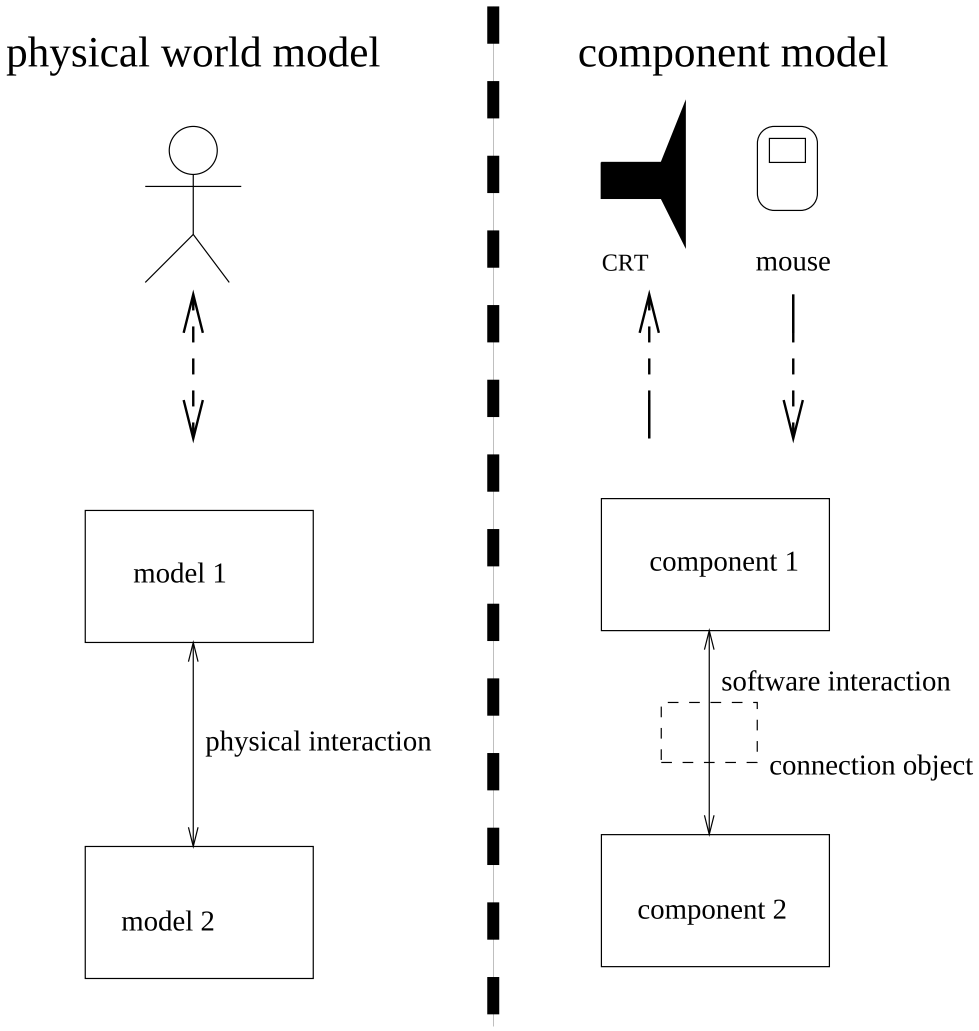
Thus, screen objects are relatively autonomous, displaying a behaviour of their own, though still subject to interactions. This makes the study of these objects as software components particularly interesting.

## 2 Position

Interactive animation is a natural application area for component-based systems. The reverse is also true: A component-based approach is vital for the further development of interactive animation systems. In this section, I will motivate these statements.

An important, and obvious, class of components are the objects on the screen. These maintain there own state, containing information such as shape, position, velocity, mass, and visual appearance. They may contain a script that dictates their behaviour, such as a motor driving a machine, but there may be also passive objects, only driven by the laws of nature, such as inertia, gravity and impulse exchanges. This leads to a natural decomposition of the system, namely into objects that our physical model of the world suggests (see figure).

Here, we use the following notion of component. A component corresponds to an object in an object-oriented model. It contains reusable code and a (persistent) state. In simple cases, the only thing that is reused is in fact the code and hence using such a component will not differ much from instantiating an object of a class. For example, a plain wheel is simply an instantiation of the class Wheel. In other cases, however, components contain data that models the graphical appearance, for instance a spoke wheel, which could be a plain wheel, decorated with spokes as an image or a texture. Espacially for more sophisticated components, such as a complete bicycle, it is awkward to design a different class for every version of a bicycle and it is easier to ditribute components as instantiated and initialized objects of a common class. Furthermore, as opposed to a mere class, a component has a design time interface. This will be discussed in the next section.




### 2.1 Design time

One of the important differences between a software component and a (software) object is that a component should have a design time interface and an object should not necessarily have one. Between the development and the actual use, a design phase is foreseen, in which components can be customised and connected to each other. Not in all applications it is clear what should be done in this phase and who should do it.

In the process of building animations, this is much clearer. The software developer builds the system to perform the animations. The end user views the animations and interacts with it. In between is the designer of the animation. He likes to think about his animations in terms of the objects that can be seen and their relationship. When the software developer provides him these objects as software components, he can work on the level of his own specialism.

At design time, the designer performs two tasks:

1. Choosing and customising the components, i.e., the visible objects of the scene. This involves

selecting their size, shape parameters, colour and other visible attributes, initial position, etc.

2. Making connections between the objects. These connections can be seen as joints, describing how two objects can move relative to each other.

As an example, consider the construction of a bicycle. This involves the components frame, handlebar, pedals, and two wheels. The parts are connected by so-called pin-joints and, for the connection between pedals and rear wheel, an orientation constraint that forces the wheel and pedals to revolve synchronously. The wheels are connected to the surface with a special wheel connection.

### 2.2 Interface

All connections between the objects are implemented as constraints that are maintained by the animation system by applying appropriate forces on the objects. This leads to a very plausible and effective simulation.

A consequence is that the interface of the components is simple. From the point of view of the dynamics, the only thing that can be done to an object, is to apply a force, a torque, or a change of impulse to it. This is represented by a method call that takes two arguments: the magnitude of the force, torque or impulse and the point of application.

## 3 Challenges and problems

Although the scene is decomposed naturally into components, some global objects remain. One of them is the constraint handler, which maintains a list of all constraints and observes deviations from the specified constraints (an object moving out of its joint, etc.). Then a numerical solver is called to compute forces that correct this deviation.

It is difficult to decompose this process into separate objects, since, potentially, arbitrary interactions exist. E.g., when two objects collide, interactions between their parts have to be computed. This is initiated by the collision detector, which creates a new constraint object when it signals a collision. This contraint object contains references to the colliding objects and represents the physical interaction between them. Again, the collision detector is a typical global object. It is a topic of active research to decompose the activity of the collision detector, especially in order to improve performance.

## 4 Ongoing research

An system for interactive animation is implemented at our department. This software is provided with Java wrappers to make writing scripts easier and to facilitate interfacing with other software. Ultimately, everything should be written in Java and the screen objects and connections should be available as Java Beans.

A next step is to develop a dedicated construction tool to build scenes out of these components. This leads to research issues such as how to build a low threshold user interface to specify complex and time dependent behaviour. Furthermore, many optimisations that are made by designers that know the mechanism of the animation system are difficult to generate automatically. In real-time animation these optimisations are often crucial for the performance.

As mentioned before, constraint correction is in principle a global task that is difficult to distribute over components. In this area reseacrh is carried out.

## References


[BarOv95] C.W.A.M. van Overveld and B. Barenbrug. All you need is force: a constraint based approach for rigid body dynamics in computer animation. *In* D. Terzopoulos and D. Thalmann (eds.), *Computer Animation and Simulation '95*, Springer, New York, 1995.

[Barenbrug96] Bart Barenbrug. *Designing a library for constraint driven dynamics in the GDP*. Final report postgraduate programme Software Technology, Eindhoven University of Technology, 1996.

[JavaBeans] Java Beans Documentation. Sun Microsystems, Mountain View, USA, 1997. `http://splash.javasoft.com/beans/spec.html`.

[Peeters] E.A.J. Peeters. *Design of an Object-Oriented, Interactive Animation System*. Ph.D. Thesis, Eindhoven University of Technology, 1995.


## 5 Biography

**Kees Huizing** is teacher and research fellow at the Department of Mathematics and Computing Science of Eindhoven University of Technology. He does research at graphical animation systems and object oriented design of autonomous objects. He received a Ph.D. in theoretical computer science in 1991.

**Bart Barenbrug** is Ph.D. student at the Stan Ackermans Institute. He is designing a rigid body dynamics subsystem at the Department of Mathematics and Computing Science of the Eindhoven University of Technology. He completed the post-masters Software Technology program in Eindhoven in 1996.