TM-0606

Program Design Visualization System for Object-Oriented Programs

by
I. Ichikawa, S. Aikawa, M. Kamiko,
E. Ono and T. Mohri (Fujitsu)

October, 1988

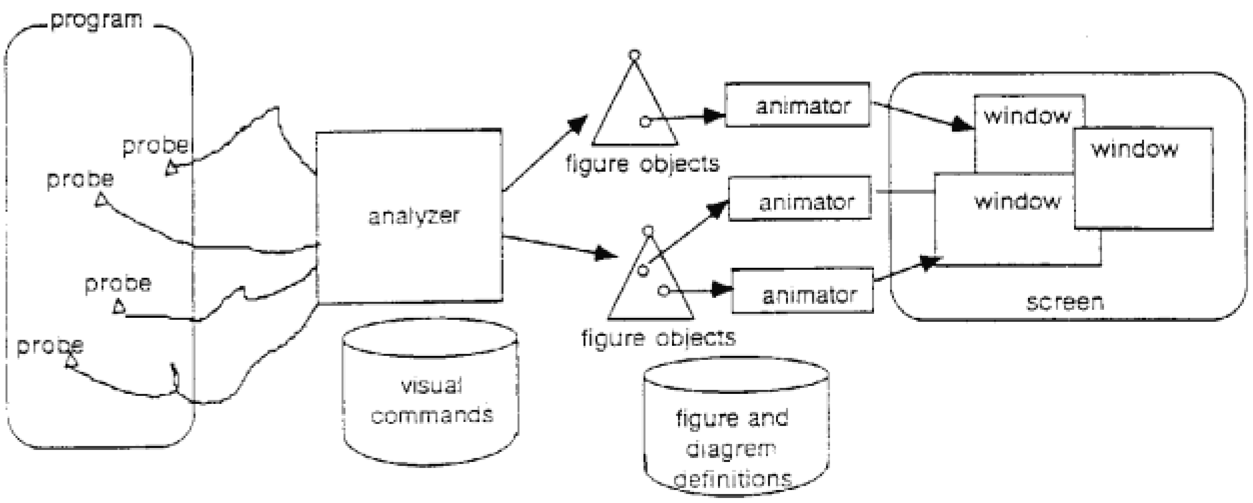

Figure 1: Program Design Visualization System

The analyzer analyzes the information with histories of events and with visualization commands, and the result of analysis commits to change the state of figure objects. The figure objects are loaded from the knowledge base of diagram and figure definitions, and they build structures of abstract figure objects. These structures are associated with the multidimensional views and individual diagrams of hierarchy. Figure objects create messages to the animators according to definitions of their shape and motion, and send them. Each of the animators manages a window of the screen and animates figures.

Some configurations of our system can be changed by the operations which invoked by user interaction. For example, structures of figure objects and their attributes are created by means of selecting diagrams from the definitions, preparation of the window for the diagrams, scrolling the window, zooming into the sub-diagram, and so on.

An example of the visualization on the prototype system

To investigate the realistic design of our program design visualization system, we made a partial prototype system on a PSI workstation. As a practical example, we tried visualizing behaviors of the KL1 software simulator [Ohara 86] on the prototype system. The KL1 (which is Flat Guarded Horn Clause, FGHC) software simulator is a simulator for basic models of the distributed KL1 processors. Figure 2 shows an example of the result of the visualization.

On the prototype system, we tryed to visualize the program using following three diagrams each of which are associated with three different views respectively. The last one is a sub-diagram of the second.

1) A basic model of a distributed computation.
   In Figure 2, a window labeled "Distributed Computation Model." In the window, we visualize how to execute goals and how to communicate among distributed processing elements (PE). In the diagram of that window, a processing element is represented by a black circle with a meter. The meter displays the number of goals queued in that PE. And communication between PEs are represented by an arrow, which flows along a channel between processors.
2) Whole configuration of the KL1 software simulator itself.
   A window labeled "KL1 Software Simulator." In the diagram of the window, there are nine processing elements (PE) and one network manager (MN), and all PEs are connected to MN with two message queues. In the window, we visualize how MN delivers messages to PEs.
3) The internal configuration of a processing element in the simulator.
   Windows labeled "PE#*i* ." In each window, there is a scheduler (sched) and a solver (solve), both of which are represented by a round rectangle, and there are queues which are represented by arrows. A goal is represented by a tiny square. In each window, we visualize how the scheduler and the solver process goals.

The system uses other windows in figure 2 for user interaction and the program execution environment.

In a proper tempo of the animation, it shows some actions of the program as if they happened simultaneouslly, and a user of the system easily understand some concurrent behaviors.

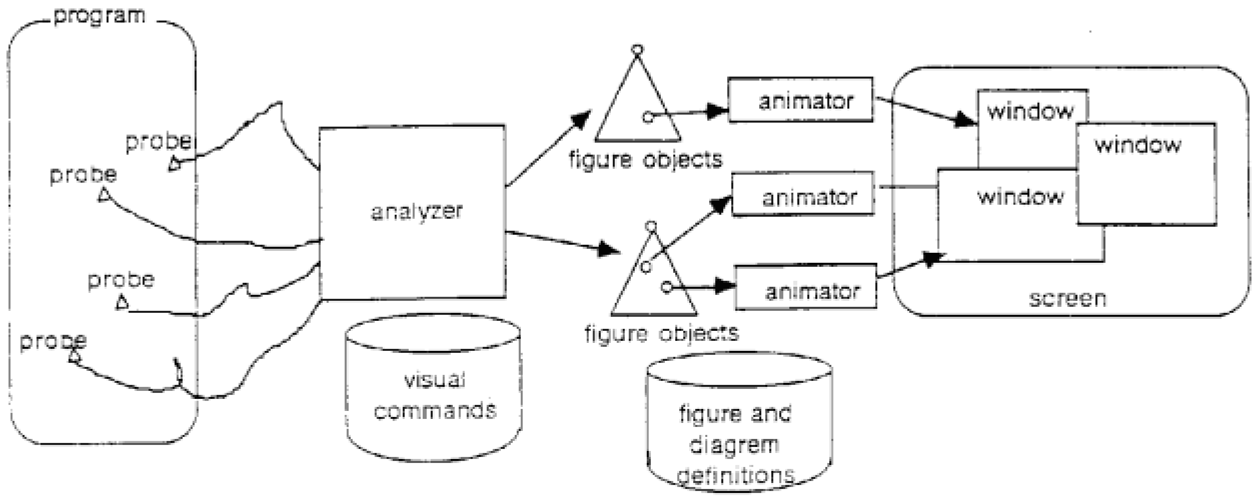

Figure 1: Program Design Visualization System

The analyzer analyzes the information with histories of events and with visualization commands, and the result of analysis commits to change the state of figure objects. The figure objects are loaded from the knowledge base of diagram and figure definitions, and they build structures of abstract figure objects. These structures are associated with the multidimensional views and individual diagrams of hierarchy. Figure objects create messages to the animators according to definitions of their shape and motion, and send them. Each of the animators manages a window of the screen and animates figures.

Some configurations of our system can be changed by the operations which invoked by user interaction. For example, structures of figure objects and their attributes are created by means of selecting diagrams from the definitions, preparation of the window for the diagrams, scrolling the window, zooming into the sub-diagram, and so on.

An example of the visualization on the prototype system

To investigate the realistic design of our program design visualization system, we made a partial prototype system on a PSI workstation. As a practical example, we tried visualizing behaviors of the KL1 software simulator [Ohara 86] on the prototype system. The KL1 (which is Flat Guarded Horn Clause, FGHC) software simulator is a simulator for basic models of the distributed KL1 processors. Figure 2 shows an example of the result of the visualization.

On the prototype system, we tryed to visualize the program using following three diagrams each of which are associated with three different views respectively. The last one is a sub-diagram of the second.

1) A basic model of a distributed computation.
In Figure 2, a window labeled "Distributed Computation Model." In the window, we visualize how to execute goals and how to communicate among distributed processing elements (PE). In the diagram of that window, a processing element is represented by a black circle with a meter. The meter displays the number of goals queued in that PE. And communication between PEs are represented by an arrow, which flows along a channel between processors.

2) Whole configuration of the KL1 software simulator itself.
A window labeled "KL1 Software Simulator." In the diagram of the window, there are nine processing elements (PE) and one network manager (MN), and all PEs are connected to MN with two message queues. In the window, we visualize how MN delivers messages to PEs.

3) The internal configuration of a processing element in the simulator.
Windows labeled "PE#*i* ." In each window, there is a scheduler (sched) and a solver (solve), both of which are represented by a round rectangle, and there are queues which are represented by arrows. A goal is represented by a tiny square. In each window, we visualize how the scheduler and the solver process goals.

The system uses other windows in figure 2 for user interaction and the program execution environment.

In a proper tempo of the animation, it shows some actions of the program as if they happened simultaneouslly, and a user of the system easily understand some concurrent behaviors.