The Chiron-2 Petri Net: A Multi-Level Block Experiment Using Chiron-1

Term Project Report
ICS 227 - Winter 1994
by

Neno Medvidovic and Jonathan E. Shaw

March 3, 1994

1 Issues

The main topic area of our project was to explore the use of a multi-level block architecture in the implementation of a user interface. The focus was on determining the viability of the proposed Chiron-2 event-based architecture. Each layer of blocks in the architecture would be responsible for a certain level of abstraction. When passing events between the levels, lower level blocks would have knowledge about the blocks above them, but their details would be hidden from the higher levels.

Other issues we explored include the separation of concerns between levels, the ability of programmers to interchange components, and the ease with which components can be added, modified, or maintained. Finally, we considered the ramifications of strictly separating the actual presentation of the components of an interface from their layout.

2 Investigation

In order to investigate the above issues, we decided to modify an existing Chiron-1 application in order to simulate what could be done in the proposed Chiron-2 architecture. The application we worked with was the Petri Net Artist, written by Kari Forester. This artist used graphical objects to represent changes in the state of a Petri net, the application's abstract data type.

The Petri Net Artist provides access to Petri net functions, such as the creation of places, transitions, and arcs, as well as presentation-specific concerns such as selecting and moving places and transitions. Like many Chiron-1 artists, the Petri Net Artist is fairly monolithic and flat. Certain ideas were decided upon in the beginning and "burnt" into the code. For example, the notion that all places would be circles and all arcs would be polylines was decided at the outset and directly incorporated into the artist. The result was a single, large, hard-to-manage artist that was difficult to understand in places and not easily extensible. This is not necessarily the fault of the artist's designer, but rather a result of the Chiron-1 architecture. Chiron-1 excels at separating user interface concerns from the application, but there is only one artist layer between the application and the Chiron server (see Figure 1). Multiple artists can be added, each responsible for a specific aspect of the display, but they all reside at the same level of abstraction and are likely to possess similar problems. This brings us to our experiment on multi-level blocks.

To our knowledge, this is the first Chiron artist designed to use sub-artists. The single level Petri Net Artist was broken into two layers of artists (see Figure 2). The higher level layer closest to the application is responsible for maintaining the layout of the Petri net, while the lower level artists are responsible for the presentation of the Petri net components. The Layout artist controls the placement of various objects on the canvas. It distinguishes between places, transitions, and arcs but has no knowledge of how those objects will appear on the screen (see Figure 3). The Presentation component artists have some knowledge of what is above them – namely, the canvas where they should be drawn and the specific locations on that canvas. Both of these conventions are consistent with the proposed way of handling events in Chiron-2: layers know what is above them, but there is no downward visibility.

The original Petri Net Artist hard-coded the design decision that all places are circles and that the number of tokens within a place is represented by various bitmaps that follow these circles around on the canvas. Beyond five tokens, the bitmaps are replaced by a text string stating the number of tokens. We could have preserved these conventions and still demonstrated our new multi-level block architecture, but we decided that a fundamental architecture change warranted a significant change in the how the semantics of the Petri net are presented.

Our presentation displays places as polygons, where the number of sides is equal to the number of tokens within the place (e.g., a triangle is a place with three tokens, while a pentagon contains five). The special cases of zero, one, and two tokens are represented by a circle, a dot (filled circle), and a thin rectangle respectively. As the transitions fire, the shapes of the places themselves, and not just the bitmaps or text associated with a place, change as tokens are added or subtracted. This was not possible with the original artist's hard-coded use of the ADL_circle class, but was made relatively simple in our Layout artist by using the abstract ADL_application class.

The specific events being sent downward from the Layout artist in the simulated Chiron-2 Petri net are Draw and Move for places, transitions, and arcs, as well as Erase for places. Drawing is fairly straightforward. After one of the Add object buttons on the canvas is pressed and a canvas location is specified with the mouse, the Layout artist sends a Draw event downwards. All of the lower Presentation component artists that deal with the particular object will hear this event, but the Layout artist has no knowledge of that or of who, if anyone, will respond. Each component in the Presentation layer is associated with a specific object representation and will only respond to a specific event. For example, the Draw_Place event has several pieces of information attached to it, including the Load – the number of tokens in the particular place. The value of Load will determine which component artist responds (the square artist only draws when it hears of a Draw_Place event with a Load equal to four).

Move events are slightly more complicated and will be discussed in greater detail below when we look at the events which are sent upwards by the Presentation layer. The downward Move event simply contains the object to be moved and the X,Y coordinates of its new canvas location. The Erase event is only used for places. When a transition is fired and the number of tokens in a place changes, the current shape of the place is erased and a new one is drawn at the same location.

The events sent up are Set events for places, transitions, and arcs and Ship events for moving, selecting, and adjusting them. Set events are a way for a Presentation artist to inform the Layout artist that an object has been drawn on the screen. When the Layout artist receives this event, it adds the object to its appropriate association table. It should be noted here that the Draw events discussed above pass down the newly created ADT object so that the Set event can return it. The low-level Draw handlers do not use the ADT object, and its sole purpose for being in the event is to let the Layout artist know which Petri net place should be inserted in the table when it handles a Set event at a later time. This type of data passing can be seen as a weakness of our design and will be discussed further in the **Results** section at the end of this report.

Finally, the Ship events are client events used by the Presentation layer to pass along the appropriate server events to the Layout artist. Currently, the Layout artist passes these events right back down to the Presentation layer in the form of a Move event discussed above. However, this does not have to be the case. In the future, additional constraints could be added to the Layout that would affect how objects are moved. For example, the Layout artist could disallow any place to be moved on top of another place. Constraints such as this one could be added at the higher level, requiring some computations to be performed to decide whether to send a Move event back down or not. At the same time, it is important to note that the change would only affect the Layout artist. Because of the separation of concerns, the Presentation component artists would not be aware of any such change.

3 Results

We feel that the aspects of the proposed Chiron-2 multi-level block architecture we studied have been successfully demonstrated by this project. The new Petri Net Artist can serve as a partial verification of the ideas behind Chiron-2. The layers in our architecture formed an abstraction hierarchy and events were passed between them with no downward visibility. The artists designed under the new architecture resulted in several benefits when compared to the original artist. The new architecture provides for a higher degree of modularity. Individual component artists can be added, interchanged, or substituted for new ones without affecting the other Presentation artists or the Layout artist. The appearance of the Petri net can be changed in any desired way with no additional modifications required at the Layout artist level. Instead, the code would be added to the Presentation level where programming is easy. Each component artist is small and simple and they are all entirely independent of each other.

Our approach was not without flaws, however. As mentioned earlier, we had to send some data in the downward Draw events that was not needed by the lower level artists, just so that we could get the data back in the upward Set events. This was a byproduct of starting with an existing system that was designed in a certain manner. We were forced to break up some single operations across the levels, so we needed to pass some additional data along with the events in order to preserve program's integrity . We would like to point out that this is not necessarily a disadvantage of the Chiron-2 approach in general, but may only be a flaw of this particular design. However, this is an issue that should be explored in the future.

Another possible failing of our approach was that we used a single Chiron-1 dispatcher to simulate the two levels of event and transaction managers in Chiron-2. The reason behind this is Chiron-1's restriction to a single client-event type. We explored the possibility of altering Chiron-1 to allow us to build a second dispatcher, but decided that this would be beyond the scope of our project after consulting the Chiron technical support staff.

While we did try to separate the concerns of window display and the Petri net itself, an area for future work would be to move all of the presentation concerns down to sub-artists. Currently, the base frame and push buttons are handled at the same abstraction layer as the Petri net Layout on the canvas. Button press events go straight to the Layout artist from the Chiron server, without first passing through a sub-artist. For the purpose of this project, however, we did not feel it was necessary to pass down the base frame in an event to a sub-artist so that it would know where to place the button.

A final point of interest is that the architecture shown in Figure 2 was not our originally planned Chiron-2 simulation. In our proposed initial architecture, the Presentation layer did not receive events from the server. We planned on setting the display objects' behaviors for events such as Move at the Layout level only. Layout was already keeping track of the locations of the abstract objects and we found out that a specific ADL class instance, such as a circle or a square, could be moved as an ADL_application if typecast beforehand. However, in Chiron-1 it is not possible to set a behavior on an abstract class such as ADL_application. This forced us to create the current architecture that actually looks more like the proposed Chiron-2 architecture than our original idea. Therefore, not only did this project indicate that it would be possible to implement Chiron-2 the way it was presented in class, but when we tried to steer away from that architecture, we were unable to do it!