

6. IMPLEMENTACIÓN

Se implementó el sistema propuesto en el caso de estudio. Las figuras 6.1, 6.2 y 6.3 presentan la interfaz gráfica del sistema en el computador de escritorio y en realidad aumentada. La implementación del caso permite definir cómo se integra el modelo a un sistema de RA.

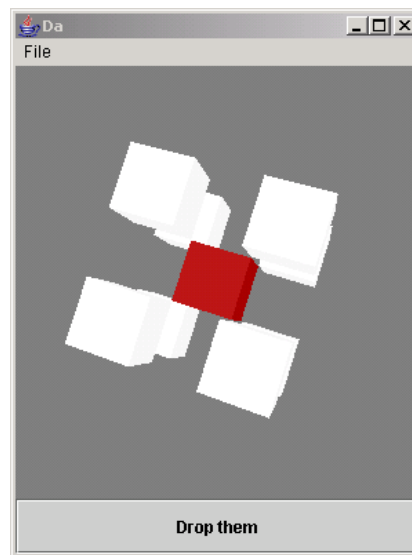


Figura 6.1: Los objetos en el computador de escritorio.

La implementación permitirá, como se presentará en la parte 7, evaluar las ventajas del uso de este modelo frente a los métodos de implementación tradicionales. Esta evaluación será útil para probar el concepto propuesto. A continuación se presentará la implementación del modelo y la arquitectura del sistema y como encaja el modelo dentro de ésta.

6.1. Representación del modelo

Se implementó una librería de grafos dirigidos en el lenguaje Java para representar el modelo. La librería permite independencia de la aplicación haciendo posible que sus elementos sean extendidos mediante herencia e implementación de interfaces. La librería

6.1 Representación del modelo

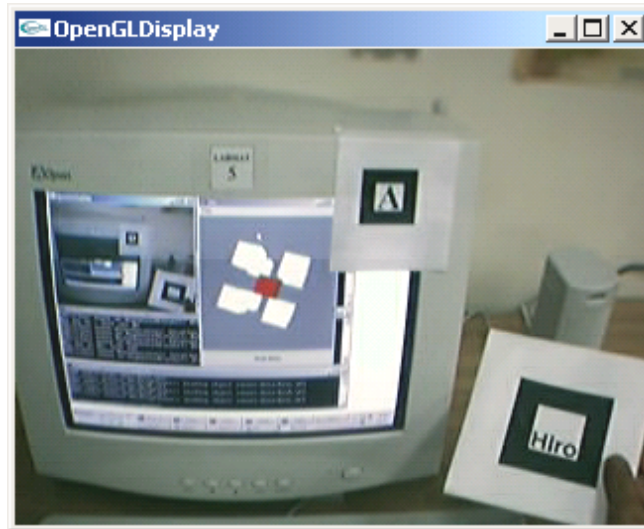


Figura 6.2: El usuario antes de tomar los objetos.

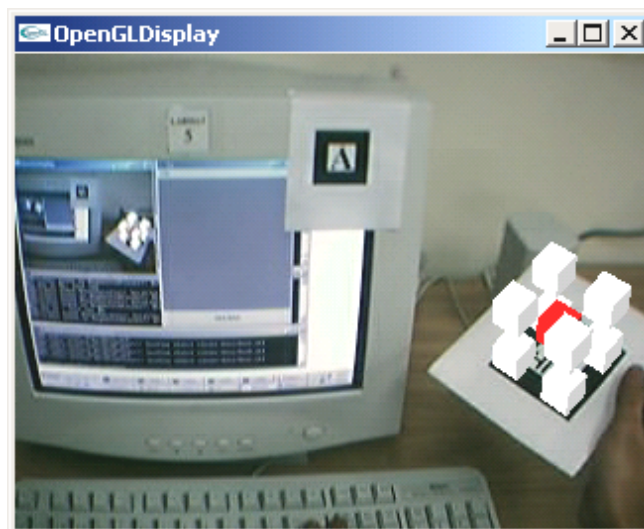


Figura 6.3: El usuario toma los objetos.

6.1 Representación del modelo

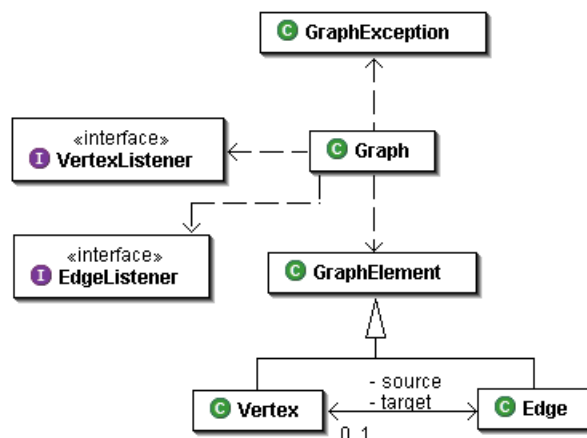


Figura 6.4: Librería de grafos.

contiene la funcionalidad básica de construcción de entidades, relaciones y atributos como se propuso en 5.2. El diagrama de objetos de la figura 6.4 enseña las clases principales de la librería. La clase *Graph* es la clase principal. Una instancia de *Graph* contiene un grupo de vértices (nodos¹) y aristas formando un grafo dirigido. En la figura 6.4 las líneas punteadas señalan dependencias de *Graph* con las demás clases. *VertexListener* y *EdgeListener* son interfaces que deben implementarse para ser subscriptores de un objeto *Graph*², que se encarga de notificar ante cambios en los vértices o aristas. Los vértices y aristas son representados por los objetos *Vertex* y *Edge*. Los miembros comunes entre estos se definieron en la superclase *GraphElement*. La excepción de grafos *GraphException* es lanzada en caso de una operación inválida sobre el grafo.

Conocer los métodos miembro de *Graph* permitirá entender la forma en que se mantiene la información del modelo. La figura 6.5 detalla los métodos de las clases principales. *Graph* implementa las cinco operaciones principales como se definió en 5.3. Además, adiciona funciones para consultar el estado del grafo, obtener elementos y adicionar subscriptores.

Los elementos del grafos tienen en común atributos, identificación y datos de usuario, que es un objeto opcional que puede ser requerido por la aplicación para almacenar información adicional. Los atributos se definen como un par nombre-valor almacenados

¹Se nombraron vértices en vez de nodos por evitar conflictos de nombre con otras librerías de Java.

²Véase el patrón del observador o publicador-subscriptor de Gamma et al. (1995)

6.1 Representación del modelo

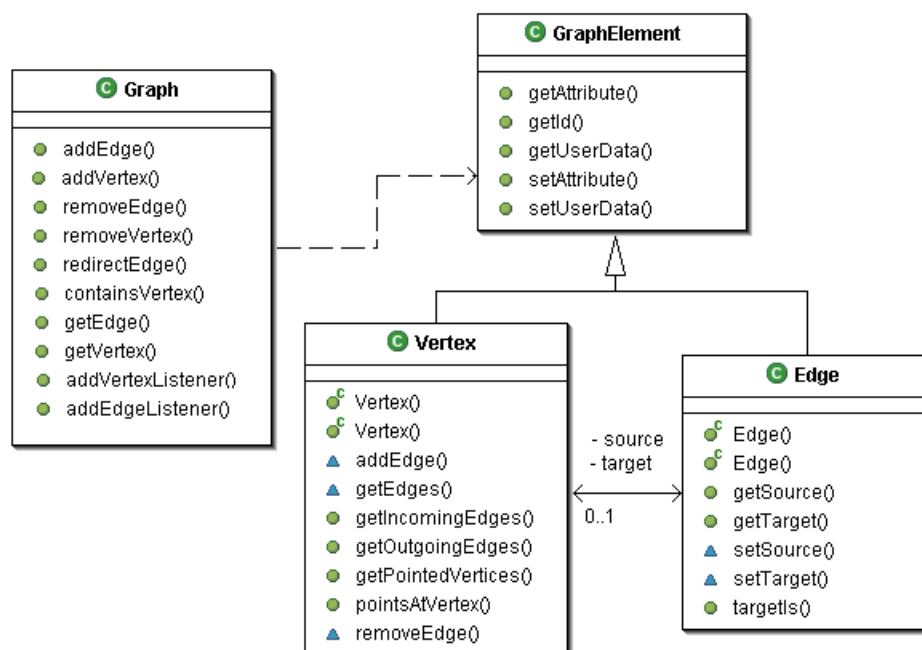


Figura 6.5: Clases principales de la librería de grafos.

ambos como objetos *String* de Java. El uso de cadenas permite simplicidad y extensibilidad en el diseño.

La clase *Vertex* representa a un vértice o nodo. Este incluye métodos públicos dentro de la librería que conservan la integridad del grafo (Ej. *addEdge()*) y métodos públicos de consulta (Ej. *getIncomingEdges()*). La clase *Edge* representa una relación. Un objeto *Edge* representa un nodo de relación mas no un enlace en sí. Los enlaces no requieren de representación debido a la descomposición presentada en la sección 5.2. Los valores semánticos de la relación son almacenados por una atributo especial en *Edge*.

En la implementación para la aplicación del caso de estudio fue requerido que el grafo fuera representado en XML. Puede pasarse del grafo XML al grafo en objetos y viceversa pues son equivalentes. El modelo descrito en formato XML nos permitirá comprender la forma en que se estructuran los tributos y relaciones. La figura 6.6 presenta el código XML. Se usaron elementos *node* para representar vértices³ y *edge* para representar

³Se usó ahora la palabra *node* por compatibilidad con lenguajes de descripción de grafos basados en XML.

6.1 Representación del modelo

```
<?xml version="1.0" ?>
<graph>
  <node id="null"
    class="null"/>
  <node id="user1"
    class="user" name="user1"/>
  <edge id="u0"
    class="uses" source="user1" target="Da" time="0.0"/>
  <edge id="u1"
    class="has" source="user1" target="Ma"/>
  <node id="Da"
    class="device" type="desktop"/>
  <node id="Db"
    class="device" type="desktop"/>
  <node id="Ma"
    class="device" type="mobile"/>
  <node id="Oa"
    class="object" type="virtual" data="boxa.wrl"/>
  <edge id="Oa0"
    class="ported" source="Oa" target="Da"/>
  <node id="Ob"
    class="object" type="virtual" data="boxb.wrl"/>
  <edge id="Ob0"
    class="ported" source="Ob" target="Da"/>
  <node id="Oc"
    class="object" type="virtual" data="boxc.wrl"/>
  <edge id="Oc0"
    class="ported" source="Oc" target="Da"/>
</graph>
```

Figura 6.6: Código en XML.

6.2 Arquitectura del sistema

relaciones. Un atributo de identificación *id* es necesario para cada vértice y relación. La clase o tipo de relación se almacena como *class*. Los enlaces se representan como los atributos *source* y *target* que apuntarán a los identificadores de los nodos. Los demás atributos son de igual forma atributos XML.

Por la simplicidad del modelo a implementar no se definieron esquemas. En caso de requerirse, el lenguaje *Scheme* de XML podría usarse para restringir los elementos ⁴. No obstante, las restricciones a las relaciones no podrían definirse en *Scheme* pues están expresadas como valores semánticos de los atributos y no sintácticos. Para restringir las relaciones debería incluirse lógica en la clase *Graph*.

6.2. Arquitectura del sistema

La aplicación propuesta en el caso de estudio se construyó como un sistema distribuido compuesto por tres sub-aplicaciones conectadas por un *middleware* común. Para su construcción se partió de Parallel Worlds Framework (Ganapathy et al., 2003) re-implementando el módulo de Realidad Aumentada, agregando soporte a una interfaz de usuario para escritorio y nuevos métodos de ubicación. El grafo en forma de *Scene Graph* original se usó a manera de jerarquía plana donde los vértices y aristas son todos hijos de la raíz del árbol.

Como se presentó en 5.1, la aplicación permite a un usuario tomar un objeto virtual de un equipo de escritorio a la realidad y luego tomarlo de regreso al mismo equipo de escritorio u otro. La ubicación del objeto, es decir qué lo contiene, es el escritorio o el computador móvil. Se requiere entonces:

- Presentar al usuario el objeto virtual en el computador de escritorio.
- Presentar al usuario el objeto virtual en la realidad.
- El sistema debe saber en que equipo se encuentra el usuario.

⁴El lenguaje de definición de esquemas XML permite describir y restringir el contenido de un documento XML (Vlist, 2002).

6.2 Arquitectura del sistema

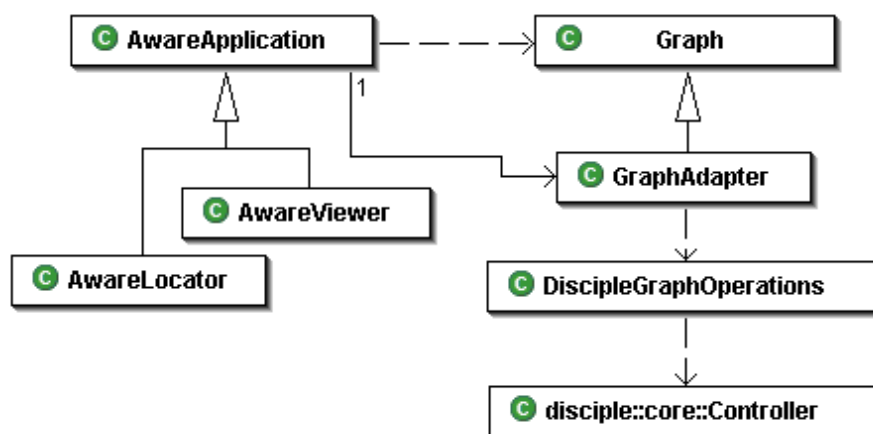


Figura 6.7: Arquitectura general del sistema.

En la figura 6.7 puede verse la arquitectura general del sistema. Se definió la clase *AwareApplication* para encerrar la funcionalidad común de las denominadas “sub-aplicaciones”. La función más importante de *AwareApplication* es el acceso al modelo, eso se hace con un objeto *GraphAdapter* que hereda de *Graph*. *GraphAdapter* oculta los detalles de implementación inherentes al *middleware* de PWF, DISCIPLÉ (Marsic, 1999). *DiscipleGraphOperations* tiene la tarea de convertir las operaciones sobre el grafo en operaciones de manipulación del SG de DISCIPLÉ y viceversa. Una vez convertidas en operaciones del SG, el *middleware* se encarga de la distribución y sincronización de las operaciones en la red. Las operaciones en el sentido opuesto son notificadas al *GraphAdapter* que usa los métodos de publicador-subscriptor del objeto grafo.

Pueden extenderse dos tipos de sub-aplicaciones, una aplicación de presentación y una de localización. Una aplicación de presentación (*AwareViewer*) presenta en realidad aumentada ejecutándose en el computador portátil o en una ventana en el computador de escritorio. La aplicación de localización (*AwareLocator*) por su parte controla el sistema que ubicación del usuario. También se ejecuta en el computador portátil. Se presenta ahora cada una de éstas.

6.2.1. Visores Se implementó el visor heredando de *AwareApplication*. La presentación se hace independiente con la interfaz *Viewer*, que se implementa de acuerdo a sí es un visor de RA o de escritorio. Esto puede observarse en la figura 6.8. Aug-

6.2 Arquitectura del sistema

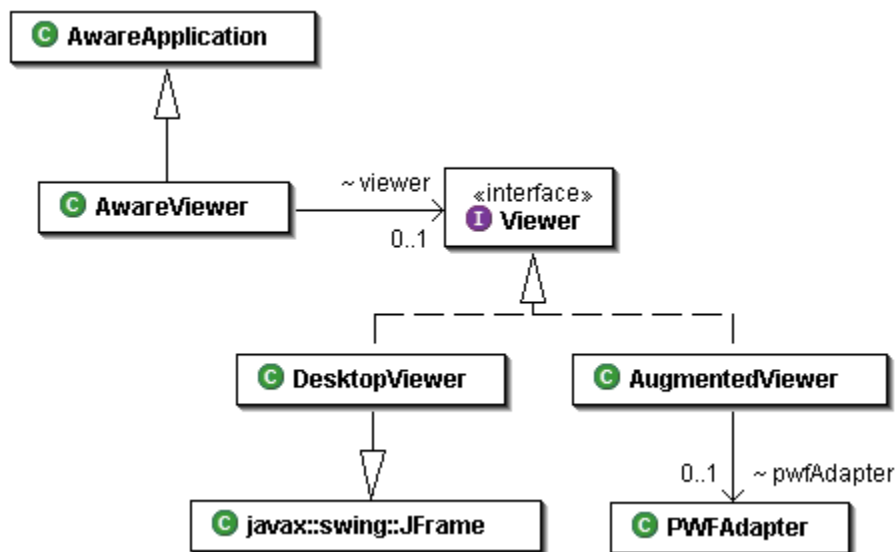


Figura 6.8: Arquitectura de los visores.

mentedViewer hace uso de un adaptador que esconde el funcionamiento del representador gráfico de RA. Este se encuentra implementado en código nativo para mayor rendimiento⁵. *DesktopViewer* extiende una ventana gráfica de las librerías de Java. La presentación 3D se hace con la librería Java3D (Sowizral et al., 2000).

El uso del modelo hizo de la construcción del visor algo bastante simple. Los visores de escritorio presentan un botón que permite al usuario tomar o devolver el objeto. En la figura 6.9 en a) se observa como el usuario puede descargar los objetos, y en b) ponerlos de nuevo. El conocimiento de la aplicación de la situación del usuario se puede ver en c) cuando el sistema determina que el usuario no puede tomar los objetos pues no se encuentran en su computador móvil. Cuando el usuario no se encuentra en el lugar el botón se desactiva. El efecto del comando del usuario y el estado del botón se logran gracias al modelo. Descargar o tomar el objeto es una operación de redirección de enlace entre un objeto virtual y el dispositivo que lo porta. Tener conocimiento de qué dispositivos tiene el usuario se realiza con una simple consulta al nodo que lo representa. Una vez ubicado el dispositivo del usuario, los objetos se trasladan allí.

⁵Más detalles sobre la implementación pueden verse en Ganapathy et al. (2003) y Kato y Billinghurst (1999).

6.2 Arquitectura del sistema

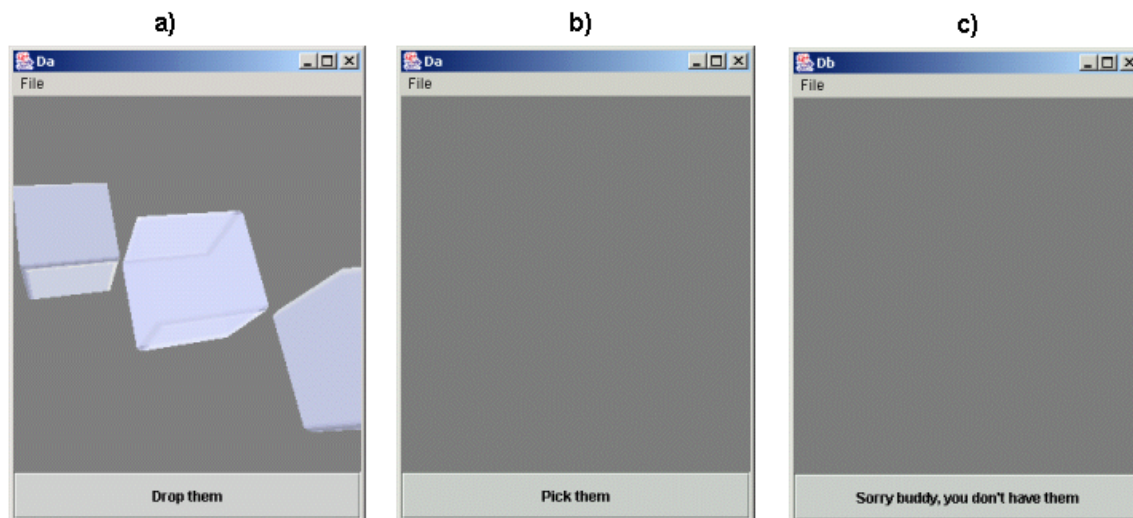


Figura 6.9: Estados del visor de escritorio.

Tanto el visor de realidad aumentada como el de escritorio son notificados de los cambios y determinan si tienen o no el objeto virtual.

El activar o desactivar el botón se determina cuando el localizador decide en que lugar se encuentra el usuario. Si un visualizador de escritorio es notificado de un cambio en la relación que indica la ubicación del usuario, éste verifica si se trata del computador donde se está ejecutando, en caso de que sí, debe activarlo, pues el usuario acaba de llegar. De lo contrario debe desactivarlo, pues no está allí.

El visor de realidad aumentada, que se ejecuta en el computador móvil del usuario, tiene la función de estar atento a cambios en la ubicación de los objetos virtuales y presentar los objetos 3D como se observó en la figura 6.3. Si según el modelo, la ubicación de los objetos virtuales es el computador móvil, estos deben presentarse en RA. El sistema RA es el mismo de PWF, haciendo uso de la librería de visión ARToolkit (Kato y Billinghurst, 1999). No se profundizará en los detalles del funcionamiento de este sistema pues se encuentra fuera del objetivo de este trabajo.

6.2.2. Localizador El localizador se encarga de determinar qué equipo se encuentra utilizando el usuario o si el usuario no se encuentra en ninguno. En la figura 6.10 se presenta el localizador. *AwareLocator* usa el sistema de ubicación de PWF. La ubi-

6.2 Arquitectura del sistema

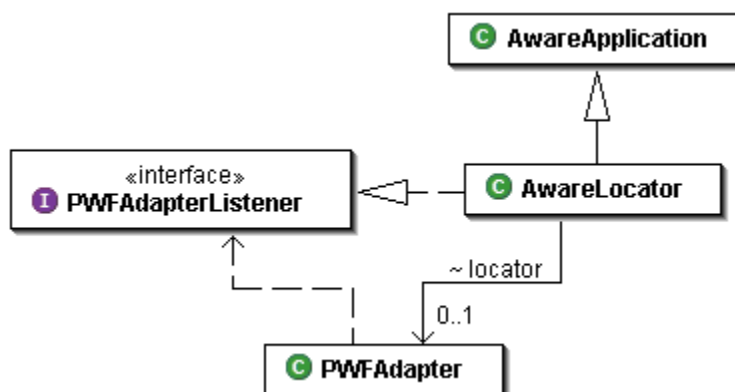


Figura 6.10: Arquitectura del localizador.

cación se hace con una cámara de video y patrones gráficos que pueden ser reconocidos por ARToolkit.

La lógica del localizador es trivial con el uso del modelo. Al sistema de ubicación se le asignan un conjunto de patrones y a cada patrón se le asigna una ubicación. Una vez se reconoce un patrón se obtiene el nombre la ubicación. El sistema de ubicación notifica al *AwareLocator* cuya reacción es ejecutar una operación de redirección de arista entre la entidad que representa el usuario y la entidad del equipo de escritorio que representa la ubicación. El localizador asocia un atributo de tiempo a la relación entre el usuario y el equipo. Este atributo es útil para comprobar cuando estuvo el usuario en un lugar para poder mantener temporalmente una relación en caso de que falle el sistema de ubicación.

6.2.3. Separación Modelo-Vista Una tercera aplicación se implementó con la intención visualizar el estado del grafo y demostrar en la arquitectura la aplicabilidad del concepto de separación Modelo y Vista (Krasner y Pope, 1988). La aplicación monitorea el modelo y lo presenta como un grafo que cambia durante la ejecución (Ver figura 6.11).

6.2 Arquitectura del sistema

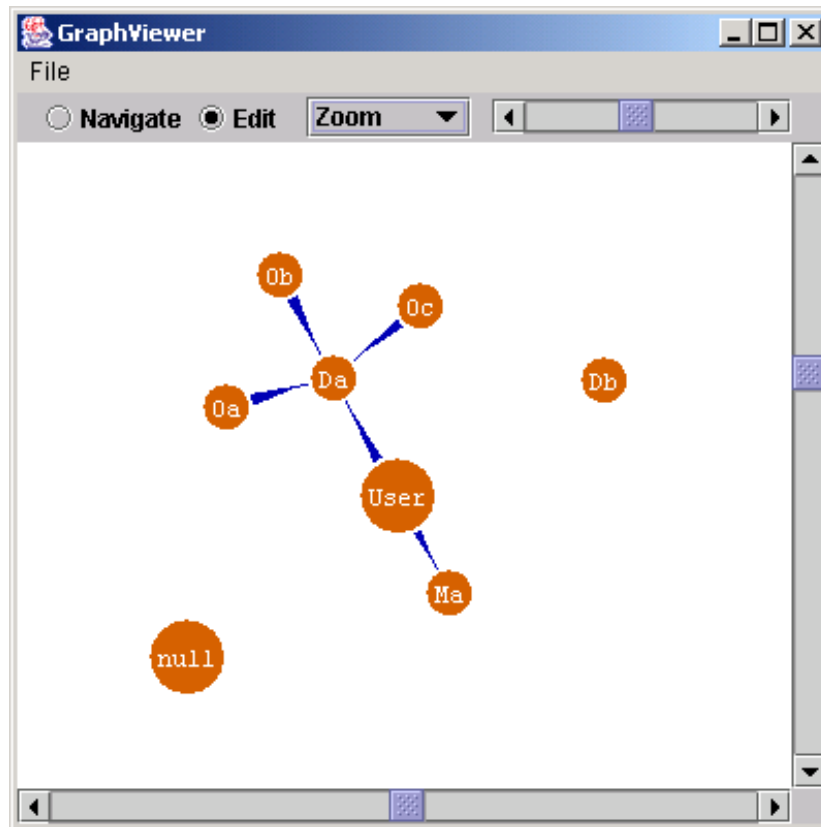


Figura 6.11: Monitor del modelo.