

SYSTEM AND METHOD FOR INJECTING AND CONTROLLING
CONTENT FROM AN EXTERNAL APPLICATION INTO A VIRTUAL
WORLD

SUMMARY

[0001] In order to integrate with 3d virtual worlds such as Second Life and Sun Wonderland, external applications such as data visualization gateways, process modelling systems, artificial intelligences, social networks, and other virtual worlds should have a protocol/language combination for placing content into the virtual space, receiving messages from the content when other content and/or users within the virtual space interact with the content, and/or sending messages to the content for the purpose of modifying it, moving it, or causing it to initiate interaction with other content and users. We describe such a protocol/language combination; and its implementation within an industry-leading virtual world, as exemplary embodiments.

DETAILED DESCRIPTION

[0002] A system and a method for injecting and controlling content from an external application into a virtual world is described below that allows for external applications, such as data visualization, etc. can be injected into a virtual world setting. In other words, a standardized protocol can be provided that allows external applications, like graphical representations of data, to be displayed within the virtual world. For example, multiple users in a virtual world environment can be shown in a "group" environment, the same presentation including graphs and other data visualizing instruments, wherein the presentation can be prepared using a protocol/language

combination that allows for integration of the presentation according to a standardized protocol/language.

Content Injection and Control Protocol

[0003] The steps which an external application should perform in order to use the protocol to render a 3d graph into a virtual world are illustrated in exemplary embodiments below. One embodiment includes presenting a menu to a user in the virtual world when the user clicks a node within the graph, and then rendering another graph into the world when the user selects an option from the menu. For example, the steps can include:

[0004] 1) transmitting definitions of content, including selection of events to be listened to and communicated back;

[0005] 2) rendering content in a virtual world, and forwarding selected events on to an external application; and

[0006] 3) upon receiving of a click event on a menu option, using an external application to transmit definitions or information of a new graph.

[0007] The exemplary embodiment protocol listed above can support the establishment of sessions, transmission of content, and messaging between the content and the external application which generated it. The language is the means by which can be defined, and by which interaction events involving the content can be selected for messaging.

Content Injection Language

[0008] Content injection language can support the creation of programming logic for responding to events without having to send a message out to the owning external application; it can also allow the creation of subroutines which can be invoked via messages from the external application. The language can also provide an object-oriented mechanism

for defining content.

Implementation as a Gateway

[0009] While it is possible, and in some cases preferable, to implement the protocol and language within the virtual world itself, it is also possible to implement them in the form of a gateway to the virtual world. The gateway technique involves using a customized version of the virtual world client software to respond to and manage CIGP sessions, and translate CIGP activity into actions within the virtual world, and virtual world events into CIGP messages. The gateway also translates CIL into a language supported natively within the virtual world.

Embodiment

[0010] Described is an application which creates a 3d graph from stock market data and displays that can allow for a display of a graph within Second Life or within an instance of Sun Wonderland, for example. The graph consists of five rectangular solids arranged in a row and representing yearly average price of a given stock, over five consecutive years. When one of the rectangular solids is clicked, a menu pops up above it, facing the user and offering two options: zoom and exit; when the user selects the "zoom" menu option, a message is sent back to the gateway from the virtual world. The gateway then creates and displays another graph above the first one, containing four rectangular solids arranged in a row and representing average quarterly stock price of the given stock, for each quarter of the year corresponding to the rectangular solid the user clicked.

[0011] The major components of the embodiment are the following:

[0012] - the application which creates the graphs; in this case a Java application (the "application");

[0013] - a SQL database containing average stock prices for one or more stocks, by quarter, for at least five years (the "database");

[0014] - an adapter which communicates with the application using CICP/CIL and which generates the graphs within Second Life (the "Second Life Adapter"); and/or

[0015] - an adapter which communicates with the application using CICP/CIL and which generates the graphs within Sun Wonderland (the "Wonderland Adapter".)

Database Structure

[0016] The database is a SQL database containing one table named stock_prices, and offering a JDBC interface for querying. The SQL definition of the stock_prices table follows:

[0017] create table stock_prices (ticker char(4), year int, quarter int, price double);

[0018] Assume that an entry in the stock_price table with quarter = 0 represents the average price for the year. The stock_price table contains the following rows:

ticker	year	quarter	price
MSFQ	1995	0	12.75
MSFQ	1995	1	10
MSFQ	1995	2	12
MSFQ	1995	3	14
MSFQ	1995	4	15
MSFQ	1996	0	14.5
MSFQ	1996	1	16

MSFQ	1996	2	15
MSFQ	1996	3	14
MSFQ	1996	4	13
MSFQ	1997	0	11.5
MSFQ	1997	1	12
MSFQ	1997	2	12
MSFQ	1997	3	11
MSFQ	1997	4	11
MSFQ	1998	0	13.25
MSFQ	1998	1	11
MSFQ	1998	2	12
MSFQ	1998	3	14
MSFQ	1998	4	16
MSFQ	1999	0	21.5
MSFQ	1999	1	17
MSFQ	1999	2	20
MSFQ	1999	3	24
MSFQ	1999	4	25

Flow of the Application

[0019] The application is a Java application which displays a window with a text field labeled "ticker", a text field labeled "start year", a text field labeled "end year", a text field labeled "adapter address", a text field labeled "origin x,y,z", a text field labeled "cell width", and a button labeled "generate".

Assume the user enters "MSFQ", "1995", "1999", "[192.168.1.2:3333](#)", "1,1,1", and ".5" in the respective fields, and clicks the "generate" button.

Assume also that the ip address of the machine the application is running on is [192.168.1.3](#), and the application listens on port 3333. The application performs the following processing steps:

[0020] 1) queries the database using JDBC and the SQL "select year, price from stock_price where ticker = 'MSFQ' and year >= 1995 and year <= 1999 and quarter = 0 order by year";

[0021] 2) read the results from the database into a linked list of objects, keeping track of the highest and lowest prices read (assume lowest price read stored in minprice variable, highest stored in maxprice variable);

[0022] 3) open a TCP/IP socket to the adapter using the address the user entered ("192.168.1.2");

[0023] 4) write "return_address 192.168.1.3:3333" to the socket, followed by newline;

[0024] 4.5) write "graph 1" to the socket, followed by newline; this is the graph id;

[0025] 5) write "node_menu 1, zoom" to the socket, followed by newline;

[0026] 6) set variable minheight to cell width (in this case .5) divided by ten;

[0027] 7) set variable maxheight to cell width multiplied by 5;

[0028] 8) loop through the linked list, with variable loopindex ranging from 1 to 5; for each record do the following:

[0029] a) set variable height to (minheight + (maxheight - minheight))

* (price - minprice) / (maxprice - minprice);

[0030] b) set variable xlocation to the x value from the origin the user entered, plus (cell width * (loopindex - 1));

[0031] c) write a line to the socket containing the following (substitute actual values for the variable names):

[0032] node recsolid,loopindex,xlocation,1,1,.5,height
<newline>;

[0033] 9) write "done 1,1,1,maxX,maxY,maxZ" to the socket, followed by newline; where maxX etc. are the values corresponding to the opposite corner of the entire graph structure from the origin;

[0034] 10) close socket; and

[0035] 11) create and retain a structure in memory (the "nodecache") which allows all values pertinent to a given graph node to be retrieved in one object, using graph id and loopindex (loopindex can also be referred to as node id) as the key (Hashtable would work for this).

[0036] The complete text written to the adapter by the above process is:

return_address 192.168.1.3:3333

graph 1

node_menu 1,zoom

node recsolid,1,1,1,1,.5,.35625

node recsolid,2,1.5,1,1,.5,.785

```
node recsolid,3,2,1,1,.5,.05
node recsolid,4,2.5,1,1,.5,.47875
node recsolid,5,3,1,1,.5,2.5
done 1,1,1,3.5,2.5,1.5
```

[0037] In an exemplary embodiment, a user within the virtual world can click on a second node within the graph (the one representing the price for 1996), and then selects the zoom option from the resulting menu. This will cause the adapter to open a socket to the application (using the ip address and port originally sent by the application in the line that starts with "return_address") and write the following lines to the socket:

```
menu_select 1,2,1
done
```

[0038] The "1,2,1" following "menu_select" corresponds to the graph id (1), the node id (2), and the menu option id (1). Upon receipt of the above lines, the application retrieves the corresponding node object from the nodecache and performs the steps described above to transmit a second graph definition to the adapter. The origin for the graph will be the x,y,z location of the node the user clicked, but with the max height from the first graph + cell width added to the y component. The SQL used in this case is different as well:

[0039] select quarter, price from stock_price where ticker = 'MSFQ' and year = 1996 order by quarter

[0040] The complete text written to the adapter is:

```
return_address 192.168.1.3:3333
```

```
graph 2
node_menu 1, zoom
node_recsolid, 1, 1.5, 4, 1, .5, 2.5
node_recsolid, 2, 2, 4, 1, .5, 1.68334
node_recsolid, 3, 2.5, 4, 1, .5, .86667
node_recsolid, 4, 3, 4, 1, .5, .05
done 1.5, 4, 1, 2, 6.5, 1.5
```

Sun Wonderland Adapter

[0041] The Sun Wonderland Adapter is a modified Sun Wonderland client which listens on a socket for content injection requests. Upon receipt of a content injection request (consisting of lines of text described in the Application Flow section), the customized Wonderland client writes the request into a file which can be retrieved via http. The Wonderland client then sends a `InjectionCellMessage` to the server with the url of the file and the bounds of the entire graph structure, signaling creation of a new `InjectionCell`. The bounds consists of two 3d points defined by the two sets of x,y,z values following "done". The server will in turn send the message out to all connected clients, causing them to create new `InjectionCell` objects within the virtual 3d space, positioned as directed by the bounds values. The `InjectionCell` object within a given Wonderland client will retrieve the url from the message, then retrieve the file using the url, and then create `InjectionShape` objects (custom subclass of the standard Java3D class "Quad") that are equivalent to the rectangular solids defined by the origin, base width, and height values contained in the file. The `InjectionShape` objects will each be registered to listen to mouse click events, allowing them to create additional 3d objects representing a menu, when clicked by the user. The menu objects created will also listen to click events, allowing

detection of the user selecting a particular menu option. When selection of a menu option is detected, the Wonderland client will send a `InjectionCellClickMessage` to the server, which will in turn send the message out to all clients. The client which received the original socket connection from the application will create a socket connection to the application, sending over the id's of the graph, node, and menu option which was clicked.

Second Life Adapter

[0042] The Second Life Adapter is a Java Servlet which in addition to responding to HTTP requests, listens on a socket for content injection requests. Upon receipt of a content injection request, the servlet creates a cache of objects corresponding to each "node" line of the request, and indexed by x,y,z position. The servlet then makes a xmlrpc request to Second Life, invoking code within an object in Second Life called the "assistant" and passing the x,y,z origin values of the graph, and the cell width. The assistant object rezzes rectangular solid prim objects in the positions defined by the request; these objects contain a script which, upon rezzing, executes an HTTP get to the servlet, passing its x,y,z position as request parameters. The servlet uses the x,y,z position passed to retrieve the corresponding node object from the cache, and returns the height specified for it in the original content injection request, as well as the node id, graph id, and the defined menu options. The prim in Second Life then makes its height the same as the value received from the servlet, and registers itself to provide the appropriate menu options to the user upon click. When the "zoom" menu option is selected, the prim makes an http request to the servlet, passing the graph id, node id, and menu option id. The servlet then opens a socket to the application and writes the "menu_select" line described above.

Claims:

1. A system for injecting and controlling content from an external application into a virtual world, comprising:
 - providing a protocol/language for placing content into a virtual space;
 - receiving messages from the content when other content and/or users within the virtual space interact with the content; and or
 - sending message to the content for the purpose of modifying an object in the virtual space.