

In this example, database program 1701 provides the ability to insert an arbitrary object (here a linked spreadsheet object) and to set or update its contents with native data through two commands, "Insert Object," and "Update Object Contents," found on the "Edit" menu of the database program. When the user selects the "Insert Object" command, the database program 1701 presents the user with a list of the classes of objects defined in the persistent global registry. Once the user has selected the type of object to create and specifies whether the object should be linked or embedded, the database program 1701 creates a default object of that CLASS_ID using the standard client library routine, Create_Object. In our example, a linked default spreadsheet object is created.

To initially set the data, or at any later time when the user wishes to update the data, the user selects the command "Update Object Contents" When "Update Object Contents" is invoked on a selected object, the database program presents a form for the user to fill in with database queries specifying the data to be placed in the selected object. Once the user has completed this form, the database program 1701 causes the data of the object to be changed by invoking the client library routine Send_Data. The range in the manager's spreadsheet 1706 is updated to reflect the scheduling data that was specified in the "Update Object Contents" form.

The spreadsheet program 1705 is used by the user's manager to view the scheduling data for the entire manufacturing project in the manager's spreadsheet 1706. This spreadsheet contains the ranges of data that were actually created by the component project teams as described above. Whenever each project team chooses to update the manager's spreadsheet using Update Object Contents, the manager will have an updated synopsis of project progress.

FIG. 18 shows the flow diagram for function Update_Object_Contents. This function allows the user to specify which data is to be sent to the object. In a preferred embodiment, this function is passed an object that is to have its data set. The function determines the object class and retrieves from the persistent registry which data formats the object supports for setting data. The function determines if it can support any of these data formats (e.g., standard spreadsheet format). If it can, then the function allows the user to specify which data from the database is to be sent to the object. In step 1801, the function displays a standard input selection form for a format that the object server supports (e.g., a spreadsheet). In step 1802, the function inputs the user input selections. In step 1804, the function retrieves the data from the database as indicated by the user selections. In step 1805, the function puts the data in a format that is compatible with the object server. In step 1806, the function invokes the function Send_Data to send the formatted data to the object. In step 1807, if function Send_Data returns an OBJ_OK message, the function returns, else the function continues as step 1808. In step 1808 through 1813, the function waits for the asynchronous invocation of function Send_Data to complete and the function returns. This is the same process as is described in steps 1110 through 1115 of FIG. 11.

FIG. 19 shows the flow diagram for the client library routine Send_Data and the corresponding server processing required. The Send_Data function checks to ensure the server for the selected object can set the object data in the requested format and sends the data to the server if it can. The function Send_Data has three input parameters: a pointer to the selected object data structure, a handle to the data, and the requested format. In step 1901, the function

determines the object CLASS_ID from the selected data structure. In step 1902, if the input format is registered in the persistent global registry for the object class, then the function continues at step 1903, otherwise the function returns ERROR_FORMAT. In step 1903, the function checks in the registry to see if there is an object handler defined for the object there is none, the function continues at step 1906, otherwise it continues at step 1904. In step 1904, the function invokes the object handler to satisfy the send request. In step 1905, if the handler can satisfy the request (it returned OBJ_OK) then the function returns OBJ_OK. If the handler cannot satisfy the request, the program continues at step 1906. In step 1906, the function determines the location of the server for the object class. In step 1907, the function determines whether the server is connected (open). If the server is not open, the function returns ERROR_NOT_OPEN, otherwise, it continues at step 1908. In step 1908, the function checks to see if the server is busy and, if it is, it returns OBJ_BUSY, otherwise it continues at step 1909. In step 1909, the function sends a SEND_DATA message to the server asynchronously, passing it a handle to the data, the requested format, and a pointer to the object. Finally, the function returns OBJ_WAIT_FOR_RELEASE to the client application so that the client knows it must wait for an asynchronous response.

On the server side, when the server library receives the SEND_DATA message, it invokes the callback routine of the server application in step 1910 passing it a OBJ_SENT_DATA notification, a handle to the data, the requested format, and the handle to the object. If the server application successfully processes the request, the callback routine will return an OBJ_OK value to the server library, otherwise the callback will return an error value. In step 1911, the server library sends the message SEND_DATA_DONE to the client library with the value returned by the server application.

Then, when the client library asynchronously receives the SEND_DATA_DONE message in its message handling routine (see FIG. 6A), in step 604, the library invokes the callback routine of the client application passing it an OBJ_RELEASE notification. At this point, step 1810 (see FIG. 18) of the client application function Update_Object_Contents will complete.

Although the present invention has been described in terms of a preferred embodiment, it is not intended that the invention be limited to his embodiment. Modifications within the spirit of the invention will be apparent to those skilled in the art. The scope of the present invention is defined by the claims which follow.

What is claimed is:

1. A method in a computer system of transferring data between a client and a server, the computer system having a persistent global registry for storing data format information, the method comprising the steps of:
 - under control of the computer system,
 - storing in the persistent global registry a plurality of data formats that the server supports;
 - under exclusive control of the client and without arbitration from an external process,
 - determining from the persistent global registry at least one stored data format that the server supports without accessing the server; and
 - selecting a determined data format;
 - under control of the client,
 - sending a request to the server to supply data in the selected data format;
 - under control of the server,