

(properties), but it is readily appreciated that alternative methods of representing the content, for example through pointers or other mechanisms, are also within the scope of the present invention. The interpretation and use of the <name,value> pairs (or other content identifier) is left to client applications. A GSO may have multiple <name,value> pairs with the same general name, thus defining an 'activity list' for the name. For example, a given GSO having a name of 'Design' could include a <name,value> pair defining a document to be modified and a <name,value> pair defining a chat session. Members in the access control list of 'Design' can simultaneously access the document and chat session to synchronously collaborate on the design. The value field is typically a text field but could optionally also support different data types, e.g. Integer, Boolean etc. In particular, a GSO supports binary content, i.e. the value field could be a large binary object. In general, the value field could support types similar to relational database systems.

[0031] Advantageously, associated with each variable property (activity, or item of information) is an access history. The access history may be used to track the member's access to the individual data item, or activity. In one embodiment, the access history may be used for security purposes, to ensure that unauthorized members have not somehow gained access to the variable property associated with the object. In other embodiments, the access history could be used to verify that a member stores the most up to date version of an item of information. In other embodiments, the access list could be used as input to programs that seek to identify member interest in certain items of information. The access history could take a variety of forms, including an ordered list of members having recent access, the types of the most recent access (modification or simply reading), etc. It should be noted that the inclusion of the access history is not a requirement, but rather a feature, of the present invention.

[0032] The access list in the general properties of the GSO manages a list of members having access to the object. The member list may include member objects or group objects. The member list both controls access and serves as a distribution list for broadcasting notifications about creation of and modifications to a GSO. The member list is dynamic and allows adding new members or removing existing members. Since the member list is also a property of the GSO, any modification to the member list is not only stored but also broadcast to the other members of a GSO. As mentioned above, one of the general properties included in a GSO is member status information. The member status information may be used to control access to the variable properties by the members. Thus, different members having different roles may access objects in different manners. For example, some members may have modify access, whereas some may only have read access. In addition, some members may have no access to particular variable properties in the activity list of the associated with the object.

[0033] An important aspect of the present invention is that the server process supports more complex collaboration through the aggregation of GSOs into hierarchies, graphs, or other structures. Thus, GSOs can have arbitrary pointers to other GSOs. Each GSO within such a structure can have different membership. This approach allows for fine-grained access control to the data. Additionally, the server process may provide convenience functions to help manage the

membership within these structures; e.g. when adding to or removing members from a single GSO, the server might provide options to propagate this operation to related GSOs. Likewise, when adding a new GSO to an existing GSO, the server may support conventions such as: aggregating the member lists of both GSOs, inheriting the member list of the existing GSO, or allowing the member list of the new GSO to prevail.

[0034] In one embodiment, a relationship data base is provided to track the relationships. This allows, for example, aggregating GSOs into hierarchies, graphs, or other structures required by more complex clients. For example, referring now to FIG. 3, a diagram is provided for illustrating a number of different relationship structures that may exist between GSOs. GSOs 60, 62 and 64 are hierarchically related. Thus, modifications to GSO 60 could be propagated to any object downwards in the hierarchy. A group of GSOs is shown having a variety of relational interconnections. In one embodiment, the relational database stores, for each object, a pointer to one or more other objects that are interested in changes to any attribute of a given object. When the given object is modified, each interested object is notified of the change to the object. The relational database may be organized in numerous manners known to those of skill in the art, and the exact structure of the database is not a limitation of the invention. Rather, any database that assists in identifying related objects may be substituted herein.

[0035] Referring now to FIG. 4, a state diagram is provided for illustrating the operation of a client that interfaces with the collaboration server of the present invention. The functionality represented by the state diagram may be implemented in the client API. The states include an Idle state 40, a Request state 42 and an Update Object state 44. When the client desires to modify a GSO, it forwards a Request package to the server. The Request package generally includes a request function and request content. Request functions include but are not limited to creating, deleting, reading, modifying or adding a property to the object. The request content identifies the attribute of the data structure seeking modification/creation/deletion and the desired content of the attribute. For example, the request function could be to add or remove a member from a list of members. Or the client may seek to add a variable property, and include the name of the property and the desired value of the property.

[0036] Once the request is forwarded to the server, the client transitions to Request state 42, where it remains until a response is received from the server, or alternatively until a predetermined time period expires. The timeout may expire as a result of the response being dropped somewhere between the client and server, or alternatively as a result of the client not being authorized to modify the object. In the event of a Timeout, the state transitions back to Idle. In the event that a Response package is received from the server, the state transitions to Update Object state 44. The system may be implemented in a variety of manners. The Response package could simply indicate to the client that the Response is granted, indicating that the client should update the Object with the desired value that it has stored. Alternatively, the Response package could include the modified value, with the client always updating the object with the value of the object received from the Server. Either way, when the object is updated, the state returns to Idle state.