662784

P/00/001
ection 29
P/00/008
ion 29(1)
Regulation 3.1(2)

AUSTRALIA
Patents Act 1990

PATENT REQUEST AND NOTICE OF ENTITLEMENT

We GPT LIMITED

of New Century Park, P. O. Box 53, Coventry CV3 1HJ, GREAT BRITAIN

being the Applicant and Nominated Person, request the grant of a patent for an invention entitled OBJECT ORIENTATED SYSTEM which is described in the accompanying standard complete specification.

Steve John ATKINS; Jeffrey Norman FROGGATT; and Leonard William PARKER are the actual inventors of the invention.

The inventors made the invention for and on behalf of the nominated person in the course of their duties as employees of the nominated person.

Convention priority is claimed from the following basic application:

Basic Applicant	Application Number	Application Date	Country	Country Code
GPT LIMITED	9202072.6	31 January 1992	Great Britain	GB

The basic application was the first application made in a Convention country in respect of the invention the subject of this request.

Our address for service is: GRIFFITH HACK & CO
168 WALKER STREET
NORTH SYDNEY NSW 2060

Attorney Code: GH

DATED this 14th day of July 1995

GPT LIMITED
By their Patent Attorney
GRIFFITH HACK & CO

AU9331061

(12) PATENT ABRIDGMENT (11) Document No. AU-B-31061/93
(19) AUSTRALIAN PATENT OFFICE (10) Acceptance No. 662784

(54) Title
OBJECT ORIENTED SYSTEM

(51)[5] International Patent Classification(s)
G06F 015/40 G06F 009/46 G06F 012/00 G06F 013/14

(21) Application No. : 31061/93 (22) Application Date : 06.01.93

(30) Priority Data

(31) Number (32) Date (33) Country
9202072 31.01.92 GB UNITED KINGDOM

(43) Publication Date : 05.08.93

(44) Publication Date of Accepted Application : 14.09.95

(71) Applicant(s)
GPT LIMITED

(72) Inventor(s)
STEVE JOHN ATKINS; JEFFREY NORMAN ~~FROGGART;~~ FROGGATT LEONARD WILLIAM PARKER

(74) Attorney or Agent
GRIFFITH HACK & CO. , GPO Box 4164, SYDNEY NSW 2001

(56) Prior Art Documents
EP 213276
US 4714995
US 4007450

(57) Claim

1. A method of operating an object-oriented system, the system comprising:

means to store an object comprising alterable data; and

means to provide a copy of the object to at least one process remote from said means to store an object,

the method comprising the steps of:

altering the alterable data of the object;

.../2

communicating the existence of the alteration to the copies of the object; and

providing a copy of the object with a copy of the altered data in response to a request from the copy of the object for the altered data.

662784
011
3.2

AUSTRALIA

Patents Act 1990

ORIGINAL

COMPLETE SPECIFICATION

STANDARD PATENT

Invention Title: Object Orientated System

The following statement is a full description of this invention, including the best method of performing it known to us:

GH&CO REF: P20319-V:COS:RK

OBJECT ORIENTATED SYSTEM

This invention relates to an object orientated system

In recent years a new programming technique has been increasingly employed for developing software programs. This technique is referred to as Object Orientation and is applicable to all stages of the software development life-cycle from Analysis to Coding. A prior art programming system of this type is described in the article "The treatment of persistent objects in Arjuna" (The Computer Journal, vol. 32 no. 4, (August) 1989, pages 323-332, Cambridge UK). The main principle employed in object orientation is to construct objects which are an encapsulation of data and the functions that operate on that data such that the following holds true:-

a. Objects only communicate with each other by passing messages.

b. The user of an object need not know anything about the internal implementation of the object, only the services that it can provide (through the receipt of a valid message).

c. The objects chosen should bear some relationship to the problem in hand. In general, Object Oriented software systems contain Objects which are abstractions of entities within the system that the software is intended to monitor or control.

This simple description does not fully characterise Object Orientation as a technique, but acts as a basis to understand the invention. Whilst objects have been defined in software terms, objects may also be thought of as amalgams of software and hardware, e.g. passing a message to an object may result in behaviour which effects the state of hardware which is being controlled - the 'software' part of the object effectively being no more than an interfacing mechanism to the 'hardware' part of the object.

As objects in a telecommunications system only communicate by the means of passing messages this offers a great deal of architectural flexibility in constructing Object Orientated systems. This gives rise to the problem that there is little conceptual difference between passing a message between objects that exist in a single process, that exist in separate processes on the same machine, or between objects on different machines. It is just the mechanism for passing the message that differs in each case. The implication of this is that there is no longer any rigid relationship between objects and processes and machines.

The term 'process' is used universally in software circles to mean a portion of software that is running concurrently with other processes on the same machine, i.e. the execution of instructions within a process is not

dependent upon the initiation or completion of instructions within another process. This concurrency is usually achieved by using a multitasking operation system to share system processing time between the processes. If such a sharing mechanism is not provided there can only be one process per machine.

It is common in Object Oriented Systems for a single object to have many users. If the users of an object are objects within the same process then there can be no concurrency involved and messages will arrive sequentially. If the object is to be used by objects in different processes the question of how to handle concurrent accesses must be addressed.

Although processes along will be referred to in the following, exactly the same may be applied for objects distributed on more than one machine.

The problem has been previously addressed by using the standard client-server models. In this approach the object to be used by a number of concurrent objects is placed in a process of its own - this is known as a server process. The objects in other processes (now known as client processes) can then pass messages to the object in the server process in order to utilise its behaviour. The concurrency of the requests being handled by some form of queuing mechanism at the interface to the server process,

provided as part of the inter-process message mechanism.

Whilst this mechanism works, it has a number of drawbacks:-

a. The action of the server process is to deal with concurrency by serialising it, i.e. the messages are just queued and dealt with one after the other. The concurrency which was attempted, effectively has been thrown away. How acceptable this is will depend upon the actual system under consideration, but on systems which have achieved concurrency by the use of multiple machines such a scheme can lead to enforced idle time whilst clients are awaiting server responses and results in non-optimal performance.

b. Message passing between objects within the same process is always much faster than message passing between objects in differing processes. The client-server approach enforces the use of the slower mechanism. If object communications could be restricted within a process as far as is possible, then worthwhile improvements in performance would be gained.

c. Typically Object Oriented designs result in a large number of objects which will need to be used by many different processes. The client-server approach

typically results in a compromise between the two extremes of putting all such objects in a single server process and placing each object in a process of its own. The use of a single server process results in a message bottleneck whereby system operation is slowed down to the speed at which the server queue is serviced, and the number of objects is usually too large to practically allocate a process to each one due to system resource limitations. In large systems a satisfactory compromise commensurate with required system performance cannot be met.

In order to address the above issues (a-c) above the Shared Object model of the invention has been produced.

If objects in differing processes wish to utilise another object between them then this object may be thought of as a shared object. Rather than creating a server process an alternative is to create a copy of this shared object in each of the processes that wish to access it. All messages passing to the shared object appears to be within the same process, in addition there is no serialization of the messages on a queue, i.e. concurrency is preserved. This sounds ideal except that in order for this shared object to be a shared object each copy within each process must be exactly the same.

An object is characterised by both its internal data

and its methods (the functions which act upon its internal data in response to arriving messages). An object's internal data is usually dynamic and varies with time. The object's methods are more static, usually being defined at system build time (or earlier) and do not vary during the system's lifetime. Therefore, when copies are made of a shared object it is only the internal data which must be kept consistent. If the methods used within a process only result in read operations on the shared object's internal data then there is no problem. However, if the methods activated result in internal data change, i.e. write operations, then such changes must be made to every copy of the shared object to maintain consistency.

This can only be achieved by inter-process messaging. A data structure must be created, and maintained, which details each process which is currently utilising a shared object. When the internal data of the shared object is modified in one of the processes this table is utilised to recreate or update the copies held by the other processes.

This shared object approach addresses the problem of concurrency as this is maintained via the multiple copies of the shared object. Inter-process messaging is still employed, but only when a shared object method resulting in internal data change is utilised, faster intra-process messaging being used for all other methods. The problem of allocating objects to server processes and the performance

problems associated with this are eliminated. This is not to say that the client-server model should not be used - there are still circumstances where it would be sensible to use this arrangement over a Shared Object approach. In addition the shared object approach also has some potential drawbacks, namely:-

a. If an object is being shared by many processes a large data table must be maintained. (However, in the client-server approach such a situation would translate into a message bottleneck).

b. In order to ensure that updates to a shared object's internal state are made consistently, object locks must be utilised, i.e. to ensure that the data is not updated from two different processes at the same time.

A first aspect of the invention provides a method of operating an object-oriented system, the system comprising:

means to store an object comprising alterable data; and

means to provide a copy of the object to at least one process remote from said means to store an object,

the method comprising the steps of:

altering the alterable data of the object;

communicating the existence of the alteration to the copies of the object; and

providing a copy of the object with a copy of the altered data in response to a request from the copy of the object for the altered data.

When an image of an object is made in a process, the use of the object by the process is recorded. When any process updates an object's reference copy, a signal indicating that a change has occurred is sent to all the processes which have an image of the object. A signal handler in each process allows the process to re-align its own representation of the object if necessary by re-reading the reference data.

A second aspect of the invention provides an object-oriented system comprising:

means to store an object comprising alterable data;

at least one means to store a copy of the object in a process remote from the said means to store an object;

means to alter the alterable data of the object;

means to inform the at least one means to store a copy

of the object that the alterable data of the object has been altered; and

means to provide a copy of the object with a copy of the altered data of the object in response to a request from the copy of the object for a copy of the altered data.

The data for a shared object is replicated only in the processes that are currently actually accessing the shared object. A notification mechanism (BOUing) is used to inform the other processes using an object that the object has changed. These processes are then responsible for re-reading the shared objects' data if necessary. No data is transported by the BOUing mechanism.

The present invention will now be described, by way of example.

The standard implementation of a client-server approach is to employ small pieces of code (sometimes termed message stubs) in the client and server processes which interface with some form of inter-process communications (IPC). The purpose of the code in the client process is purely to relay any requests on the object in the server process as messages to the server process using the IPC mechanism. The IPC mechanism will queue messages at the server process, with the small amount

of code in the server process reading this queue and directing the requests to the object to service them. The whole process can be thought of as a pipe which takes in requests at one end from the client process(es) and delivers them at the other end of the pipe to the appropriate object in the server process. All activity required to service the requests (rather than merely transport it) is handled in the server process.

The implementation of the shared object approach replaces the client-server 'pipe' with a mechanism which deals with requests locally within the process using the shared object. When such requests modify the internal data of the shared object, only then will inter-process communications be utilised to ensure other copies of the object are brought into alignment with the changes.

Objects that are to be shared within a system must be identified as such at compile time. This is achieved by defining a shared object class from which objects wishing to exhibit this characteristic may inherit. Whenever a shared object is created, a copy of that object is created within the requesting process. If this is the first time that object is created then two additional items are created; a copy of the object's data is created in shared memory along with a used-by table into which an identifier for the requesting process is written. Subsequent

creations of this shared object by other processes will result in a copy of the object being created within the requesting process whose internal data will be an exact copy of the data stored in shared memory, and the used-by table is updated with the requesting processes' ID.

The shared memory can be thought of as a global data area accessible from any process.

Whenever a request is made of a shared object, then that request is serviced locally by the process. If the request results in the shared object's internal data being modified, then the shared memory copy of the shared object's data is updated to reflect the change(s). In addition to this it will be necessary to send a signal to every other process using the modified shared object indicating a change has occurred such that they can realign their local copies of the shared object with the modified data stored in shared memory. This signalling process has been given the term Broadcast on Update (or BOU) and is achieved by utilising the used-by table to send a signal to all processes registered as users of the shared object. Each process which contains shared objects must contain a BOU signal handler which will ensure that the BOU signal results in the necessary updates.

It is of course possible that a process will at some point no longer require to share an object. When this

occurs, the shared object will be removed from the process and the used-by table will be modified to delete the requesting processes' identification.

CLAIMS

1. A method of operating an object-oriented system, the system comprising:

means to store an object comprising alterable data; and

means to provide a copy of the object to at least one process remote from said means to store an object,

the method comprising the steps of:

altering the alterable data of the object;

communicating the existence of the alteration to the copies of the object; and

providing a copy of the object with a copy of the altered data in response to a request from the copy of the object for the altered data.

2. A method as claimed in Claim 1, further comprising the steps of:

altering the alterable data of a copy of the object; and

communicating the altered data to the object, whereby the said step of altering the alterable data alters the alterable data of the object to correspond with the altered data held by the said copy of the object.

3. A method as claimed in Claim 1 or Claim 2, in which the said request from the copy of the object for the altered data is made only when the copy of the object needs to use the alterable data.

4. A method of operating an object oriented system substantially as described.

5. An object-oriented system comprising:

means to store an object comprising alterable data;

at least one means to store a copy of the object in a process remote from the said means to store an object;

means to alter the alterable data of the object;

means to inform the at least one means to store a copy of the object that the alterable data of the object has been altered; and

means to provide a copy of the object with a copy of the altered data of the object in response to a

request from the copy of the object for a copy of the altered data.

6. A system as claimed in Claim 5 comprising:

means to alter the alterable data of a copy of the object;

means to provide the object with a copy of the altered data of the copy of the object,

in which the said means to alter the alterable data of the object alters the alterable data of the object to correspond with the altered data of the copy of the object.

7. An object oriented system as claimed in Claim 5 or 6 in which the copy of the object is arranged to request a copy of the altered data only when it needs to use the alterable data.

8. An object oriented system substantially as described.

Dated this 14th day of July 1995

GPT LIMITED
By their Patent Attorneys
GRIFFITH HACK & CO.

ABSTRACT

Object Orientated System

The use of the client-server approach to the problem of multiple users of objects in Object-Oriented Systems is well known.

An alternative approach is disclosed where a shared object is created which is copied to all system nodes requiring access to the object and a used-by table created identifying the nodes holding a copy of the object. When the object is updated by operations at one of the nodes then the copies identified by the used-by table are also updated.