(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 November 2010 (18.11.2010)

PCT

(10) International Publication Number
WO 2010/131023 A1

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/GB2010/050754

(22) International Filing Date:
10 May 2010 (10.05.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0908041.7 11 May 2009 (11.05.2009) GB

(71) Applicant *(for all designated States except US)*: **TRUE BLUE LOGIC LIMITED** [GB/GB]; 11 Alpha Road, Stretford, Manchester, Greater Manchester M32 9JJ (GB).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **BASAK, Robie Ronjon** [GB/GB]; 11 Alpha Road, Stretford, Manchester, Greater Manchester M32 9JJ (GB).

(74) Agent: **APPLEYARD LEES**; 15 Clare Road, Halifax, Yorkshire HX1 2HY (GB).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

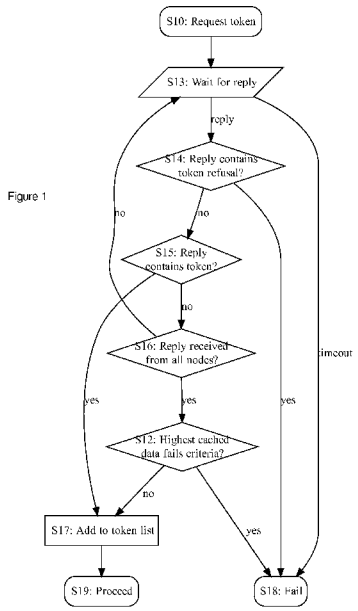
(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

(54) **Title**: IMPROVEMENTS IN AND RELATING TO REPLICATED FILE SERVERS

Figure 1



(57) **Abstract**: Disclosed is a method of controlling the opening of a file, the file being one of a plurality of files on a replicated file server connected to one or more other replicated file servers over a network, each of said plurality of files being associated with a corresponding unique file token, the file token being held on one of the file servers only, the method comprising the steps of: the file server attempts to acquire the file token; and if the file server acquires the file token, the file server permits the opening of the file.

WO 2010/131023 A1

Improvements in and relating to Replicated File Servers

Field of the Invention

The present invention relates to improvements in and relating to replicated file servers. More particularly, but not exclusively, the present invention concerns token passing between file servers.

Background of the Invention

In a prior art configuration, a number of file servers are connected over a network and files are replicated and synchronised across these file servers. Each file server is located at a node of the network. Methods exist to ensure correct synchronisation of files and file structure across the network on the multiple file servers. One of these methods employs the use of tokens.

EP 0 694 839 A2 uses tokens to achieve file server replication and synchronisation. A file server at a node sends all write file system calls to all other nodes as they happen ensuring that a read call on a node to which such a replicated write has been sent does not begin until the write is complete. The disadvantage of this system is that the upstream bandwidth required by a local node when performing a local write is equivalent to the number of other nodes in the system multiplied by the size of the write. This means that local write speeds degrade to a speed that is actually slower than if the user had opened a remote file. The real-time bandwidth required by this system is high and, when used over the Internet, high latencies will be experienced by users.

Tokens are more commonly used in networking technology for electrical signalling wires connecting electrical devices rather than file servers. A token is a notional concept and was first popularised by a token ring local area network technology as described in US patent 4 293 948. Token passing around a ring in this fashion is well understood and documented.

The principle behind tokens and token passing is that, when in a system of interconnected nodes connected by electrical signalling wires, it is required that no two nodes perform the same or similar operation at the same time. For example, no two electrical devices at different nodes using a shared electrical signalling wire should be able to transmit on that wire at the same time. A token is passed between the nodes and no node performs an operation that must not be carried out by two nodes at once until it holds the token. No node passes on the token until it has completed any such ongoing operation.

The token ring technology referred to continually passes the token around in the ring of nodes in case any node needs to use it. This even occurs when the system is idle and no electrical devices are transmitting. Although this network activity occurs when the system is idle, this is practical because the available bandwidth is dedicated solely to the token ring technology and the latency over a local area network is low.

For a system of nodes connected over a wide area network (WAN), using token ring is a problem because there is typically other traffic that also requires bandwidth. It is therefore not practical to use up bandwidth when the system is idle. Additionally, latencies encountered on WANs in practice could mean that on a system with just a few nodes it would take seconds for a token to pass all the way round a system, and tens of seconds to detect and recover a lost or corrupted token. This sort of delay would be unacceptable to users who need to access the file associated with the circulating token and who typically prefer delays under 200ms in order for such delays to appear imperceptible to them.

An alternative to passing tokens around in a ring is to pass them point-to-point between nodes. Many modern WANs are in fact virtual private networks (VPNs) operating over the Internet which is a packet switched network. A problem with token passing over the internet is that tokens are passed as messages and are not guaranteed to reach their destinations. In prior art configurations, in order to successfully pass a token between two nodes it is necessary to transmit a message between the nodes and for both nodes to be in mutual agreement as to whether the message was successfully transmitted. Over an unreliable medium this is impossible and so messaging cannot be used to pass a token reliably. This problem was first published together with its proof by Akkoyunlu et al in 1975 and is now known as the Two Generals' Paradox.

The token ring method deals with the unreliable network and lost messages/tokens problem by re-creating the token if the token does not pass a particular node in the ring after a fixed amount of time. However, with point-to-point token passing this solution is not possible. Without tokens being passed around in a ring no single node is able to perform this type of monitoring.

An aim of the present invention is to provide a file server and method of replication that overcomes or obviates at least one problem associated with the prior art whether mentioned herein or not.

Summary of the Invention

According to a first aspect of the present invention there is provided a method of controlling the opening of a file, the file being one of a plurality of files on a replicated file server connected to

one or more other replicated file servers over a network, each of said plurality of files being associated with a corresponding unique file token, the file token being held on one of the file servers only, the method comprising the steps of: the file server attempts to acquire the file token; and if the file server acquires the file token, the file server permits the opening of the file.

Suitably, the file server attempts to acquire the file token by first checking if it holds the file token and if it does hold the file token it permits the opening of the file without requesting the file token from another file server.

Suitably, if the file server does not hold the file token, the file server attempts to acquire the file token by requesting the file token from the other replicated file servers.

Suitably, the request for the file token is made by transmitting a message peer-to-peer from the requesting file server to each of the other filer servers.

Suitably, if any of the other file servers refuses to pass the file token, the requesting file server does not permit the opening of the file.

Suitably, if none of the other file servers have the file token, the requesting file server recreates the file token and permits the opening of the file.

Suitably, the file token is recreated using information stored on the requesting file server.

Suitably, the file token is recreated using information stored on the requesting file server and one or more of the other file servers.

Suitably, the file token comprises a file version and one of the plurality of other file servers does not pass the file token if the file version of the file token is greater than the file version of the file held on the requesting file server.

Suitably, each file server maintains a local token list of the file tokens it currently holds.

Suitably, opening a file comprises opening a file for writing or for reading.

Suitably, each file server maintains a local token cache of the file tokens it has previously held.

Suitably, the requesting file server recreates the file token using information stored in its local token cache.

Suitably, the requesting file server recreates the file token using information stored in its local token cache and the local token cache of one or more of the other file servers.

Suitably, if the requesting file server has not received the file token and does not receive a response from each of the other file servers within a defined period of time, a timeout occurs and the requesting file server does not acquire the file token and is not permitted to open the file.

Suitably, the files are stored in directories in a hierarchical structure where each file and directory contained within a parent directory is a child of that parent directory, each directory has a corresponding unique directory token held on one of the file servers only, and the directory structure is replicated across the file servers, where a file server cannot permit the creation of deletion of a child file or directory unless it holds the parent directory token.

Suitably, the file server first checks if it holds the directory token and if it does hold the directory token it permits the creation or deletion of a child file or directory.

Suitably, if the file server does not hold the directory token, the file server acquires the directory token by requesting the directory token from all other file servers; and acquires the directory token from one of the other file servers.

Suitably, the directory token comprises directory token data relating to the creation and deletion of first generation child entries.

Suitably, the directory token data relates to the creation and deletion of first and subsequent generation child entries.

Suitably, each file and directory additionally comprises a name sequence number and the directory token comprises directory token data comprising the directory name sequence number, whereby one of the plurality of other file servers does not pass the directory token if the directory token name sequence number is greater than the directory name sequence number.

Suitably, when a new token is created, the parent token name sequence number is incremented.

Suitably, when a token is deleted its parent token name sequence number is incremented.

According to a second aspect of the present invention there is provided a file server arranged to perform the method of the first aspect of the present invention.

According to a third aspect of the present invention there is provided a computer-readable recording medium having recorded thereon program code instructions arrange to cause a file server to execute the method of the first aspect of the present invention.

Brief Description of the Drawings

For a better understanding of the invention, and to show how embodiments of the same may be carried into effect, reference will now be made, by way of example, to the accompanying diagrammatic drawings in which:

Figure 1 shows a flow chart for a file server requesting a token;

Figure 2 shows a flow chart for a file server receiving a token request;

Figure 3 shows a flow chart for a file server requesting the creation of a file;

Figure 4 shows a flow chart for a file server receiving a creation notification;

Figure 5 shows a flow chart for a file server receiving a deletion notification; and

Figure 6 shows a flow chart for a file server requesting the deletion of a file.

Description of the Preferred Embodiments

In a first embodiment, a file server is provided. The file server is implemented using a stack-based architecture. In this architecture, different component layers are notionally stacked together and lower layers provide services to upper layers. The lowest component layer in this architecture is the wide area network (WAN). A typical implementation on a WAN is the Internet. Services provided by this component are an unreliable message passing service as implemented for example with UDP and a reliable stream service as implemented for example with TCP.

Sitting above the WAN is a message caching layer. The message caching layer sits beneath a token passing layer and caches requests from, and replies to, the token passing layer so that these can be resent and duplicates removed as required. The token passing layer passes tokens to other file server token passing layers and communicates to an upper replicated file system layer that the token is held. The replicated file system layer is able to lock and unlock the tokens in the token passing layer. Tokens are described below.

Above the token passing layer is the replicated file system layer. This layer implements a fault-tolerant, distributed multi-master replicated file system and handles the creation, modification

and deletion of files and directories. Multi-master refers to the fact that any file server node may initiate a change. It uses the token passing layer to determine if and when a given create, open, rename or delete request from a user application layer, which sits above it, should succeed. The user application layer connects the user applications to the replicated file system.

Each instance of this stack represents a single node. The system as a whole contains a plurality of nodes and thus a plurality of stacks.

The file server provides a file system having a lowest level root directory containing other directories which branch off from this root directory and files which are stored in these directories. The file server is connected to a network at a node of the network. Other file servers are also connected to this network at their respective nodes. The file servers are replicated in that they aim to replicate the file systems, the same directory structure and file content, across the servers. When these servers are synchronised, the file servers are truly replicated and the file systems are the same across all file servers.

One or more users have client computers connected to a particular file server. These client computers access the files stored on the file server. When a file is opened by a first user on a file server at a particular node, a second user connected to different file server at a different node is restricted in the possible operations they can perform on that same file replicated on their file server. Opening a file comprises opening a file for writing or for reading. These restrictions are such that the second user's file server effectively behaves as though the file accessed by the first user is located on the same file server as the second user rather than a different file server. Once changes have been made to a file system on any particular file server, these changes are replicated and synchronised across all of the file servers on the network.

In order to achieve the file restriction, replication and synchronisation across file servers connected to a network, tokens are used. The principle behind tokens and token passing is that, when in a system of interconnected nodes, it is required that no two nodes perform the same or similar operation at the same time: a token is passed between the nodes in some fashion; no node performs an operation that must not be carried out by two nodes at once until it holds the token; and no node passes on the token until it has completed any such ongoing operation.

A token comprises token data which is stored on the file server. Each file and directory on a file system has an associated token, and whilst the file systems are replicated across different file servers, the tokens are not. In this embodiment the token data comprises a token identifier

and a token version. The token identifier is information relating to which file or directory it is associated, that is, the token identifier is the full path name of a directory or file.

The token version is an integer value and corresponds to the latest available revision number for any given file. When a file is amended, the corresponding token version is incremented by one.

The replicated file system layer may modify the token data only if the local file server holds the token. When a token is requested, the requesting file server makes this request subject to some criteria which must be met for the token passing to succeed. If a file server receives a request with criteria it will pass the criteria in the request to its replicated file system layer together with the token data that it already holds. The replicated file system layer will then perform the comparison and veto the request if required.

When a token is requested in this embodiment, the requesting file server makes this request subject to the criterion that the token data that contains the file revision number is equal to or lower than the version number of the replica of the file held on the requesting server. The server that receives this request will only pass the token if this criterion is held. This prevents the token being passed unless the requesting server already holds the latest version of the file.

The token is passed around between file servers and acts as a lock. If a user wishes to perform an operation on a file stored on a particular file server, the file server must first obtain the token before this operation can occur. If not held by the file server, the token must be passed to it. Tokens are therefore passed point-to-point between file servers at different nodes on a network.

Each file server maintains a local token list, a local token cache and a deleting token list. The local token list comprises a list of token data. On any particular node a token is referred to as being held when the corresponding token identifier is present in the local token list on that node. Passing a token between file servers is achieved by passing the token data in a network message over the network from one node to another. When a file server passes a token, the token data is deleted from its local token list. When a file server is passed a token, the token data is added to its local token list. When a file server requests a token, the rules that determine whether the token can be passed are described below. The local token cache is identical to the local token list except that it is used to store a copy of each token as it was last seen, in order that lost tokens do not result in lost token data. The deleting token list comprises a list of all token identifiers representing tokens that are in the process of being deleted.

Referring to figures 1 and 2, a token is requested S10 by a requesting file server. The requesting file server sends this request to all file servers on the network by broadcasting, multicasting or sending messages in turn. When a file server receives a request S20 it determines whether it has already made a request for the same token S21 and if it already has an outstanding request for the same token then it refuses the request S28 and processing terminates. If the token identifier is in the deleting token list S29 then it refuses the request and processing terminates. If the token identifier is not present in the local token list S23, then it declines the request S27, sends back the token data from the cached token list if present, and processing terminates. Otherwise it passes the corresponding token data and the criteria from the request to the replicated file system layer for checking S22. If the result is that the token data fails the criteria in the request then it refuses the request S28 and processing terminates. If it does hold the token then it determines whether the token is still needed S24. If the token is still needed it refuses the request S28 and processing terminates. If the token is not needed then it removes the token data from its local token list S25 and passes the token to the requesting file server S26.

Once a request for a token is made, the requesting file server waits until replies are received S13. If a reply contains a refusal S14 then it determines that a file server already holds the token and cannot give it up so the request fails S18 and a negative response is sent to the replicated file system layer. If a reply contains the token S15, it adds the token to its local token list S17, stops waiting for replies and informs the replicated file system layer that the token is now held.

If a reply is declined because the token data held by the responding file server does not meet the criteria in the request S14, then the requesting file server can draw the conclusion that the token cannot be acquired at all without having to wait for any further responses from other file servers, and the request fails S18. It then passes this negative response to the replicated file system layer.

If a reply is declined because the token is not held, the file server continues to wait for replies. If all file servers have replied S16 and all have declined without passing the token, then the token is not held by any of the file servers. The requesting file server compares the token data in its local token cache with the cached data in its the decline messages and determines which is the most up to date. This most up to date token data is checked against the criteria in the original request S12. If the criteria fails, then the request fails S18. Otherwise the requesting file server adds the most up to date token data to the local token list; thus recreating the token.

If the requesting file server is still waiting for a response after a defined period of time then timeout occurs, the request fails S18, and a response is sent to the replicated file system layer that the token is not available.

If two computer systems on the same file server request the same token, the file server does not attempt to act twice but instead can either refuse the second request immediately or merge the two requests together.

Using this mechanism, the Two Generals' Paradox is avoided by not requiring that both nodes involved in the passing of a token agree on the result; only the node requesting the token need know the result. The node handing over the token only needs to know if it attempted to hand over the token to make sure that it doesn't consider itself to still have the token. A possible consequence of this is that the token could end up lost. This is a relatively unlikely occurrence due to the message retransmissions carried out by the message caching layer. If it does happen then, as described above, the token can be recreated in a manner that is guaranteed not to be carried out by more than one node at once.

A token is created when its corresponding file is created. In order to create a file it must first be determined that an identical file has not been created on a different file server since prior to synchronisation, this new file would not yet be replicated across all servers. To determine this, the token for the file to be created is requested from all other file servers. If all replies are received and the token is not held, the file can be created along with a corresponding token.

A token is therefore implicitly created when it is first used. If a node receives a request for a token that is not in its local token list, then it acts as if the token already exists but is not held locally.

To delete a file, a node first acquires the corresponding token using the procedure detailed above and then deletes the file. It then adds the token data to the deleting token list and deletes the token data from the local token list and the local token cache. The token passing layer then sends a message to each other node stating that it holds the token and that it is deleting the file. When a node receives such a message it informs its replicated file system layer that the file is being deleted. The replicated file system layer then deletes the file and the token passing layer responds back to the node that is performing the deletion confirming that the file deletion has been successful. Retries are sent until all nodes have responded and confirmed that they have deleted the file. Once all nodes have responded, then the node performing the deletion removes the token from its deleting token list.

To create or delete a directory, the same process is followed as for creating or deleting a file.

Providing a true, fault-tolerant replicated file system that is replicated between distant nodes while at the same time behaving exactly the same as a non-replicated file system present only at the user's own node is particularly difficult. To make it appear exactly the same, it would need to reproduce local file system semantics precisely. In most cases however it is not necessary to replicate file system semantics precisely. Exact semantics are needed for particular specialised applications such as database engines and email server queues but many of these applications already provide a remote network protocol and do not need replication at the file system level.

Files being used to store common user documents are those such as word processing, spreadsheet and presentations files. The applications that manage these files typically apply an access pattern that requires few of the full set of file system semantics to be preserved. For example, some of these applications may apply the following workflow: the application locks and opens a file; the application reads the entire file into memory or into a temporary local file; the user interacts with the application and makes changes into the application's local copy; the user saves the document, in which case the application writes the entire local copy back to the file, and when the user is finished, the application closes and unlocks the file.

During the entire time that the application has locked the file, another application running even at the same site is unable to access the file. This is normal behaviour as far as both the user and the application are concerned and indeed this mandatory locking is the default behaviour on some operating systems. Therefore, a replicated file system is still possible even without replicating file system semantics associated with concurrent access to single files, and such a replicated file system will still support the use of common user applications such as word processors, spreadsheets and presentation tools.

To prevent concurrent writes and lost updates, a node will acquire a token for a file before opening it for writing. Lost updates are described below. This will result in an exclusive lock equivalent to the mandatory locks used in non-replicated systems. Writes to the file will only occur locally and will not be immediately replicated to other nodes. Other nodes will not be able to acquire the token that corresponds to the file.

When writing is complete and the local file is closed, the node will inform all other nodes that an update is available, and will refuse to pass the token back to a node which does not have the latest copy. When a node receives a notification that an update is available, it will download the file or suitable file delta to update its local replica. A file delta is the difference between two revisions of the same file. If the file or delta is particularly large, or downloading immediately would lead to too much bandwidth use, the file is added to a list of files to be

replicated later. If updates to many nodes are required, the nodes could use a distribution mechanism such as BitTorrent to spread out the upstream bandwidth required and improve performance.

Typically, file systems store more data about files and directories than solely their names and content. This extra data can include ownership information, permission bits, access control lists and extra streams and is referred to herein as metadata. For the purposes of replication and token passing, it is sufficient to consider this metadata as part of the file data. Like file data, the corresponding token is held for a given file or directory while its metadata is modified; a metadata change creates a new file or directory revision; and the metadata is replicated at the same time as file data.

After an update has been made to a file or to file or directory metadata on one node, there is a period of time during which this change has yet to be replicated to other nodes. For robustness, the situation where one or more nodes are not accessible also needs to be considered. It is a valid state of the system for different nodes to have different revisions of a file at the same time.

It is a design requirement that lost updates are prevented. A lost update occurs when: a first user opens a file on first node for editing; a second user opens the same file on second node for editing; the first user then changes and saves the file; and the second user then makes different changes and saves the file such that there are then two versions of the original file. When replication occurs one of the versions is overwritten and the changes represented in that file have been lost.

Token data and criteria are used to ensure that a file cannot be opened for read or write access on a node which has a file with an old revision. All changes to files and file and directory metadata are counted by incrementing the token data integer representing the file version.

Each node's replicating filesystem layer keeps track of a version number against every file and directory on the system. When acquiring a token, in order to make a modification to a particular file or directory, the request is sent with the criterion that the token data version number must be equal to or less than the version number that the node already holds. This ensures that the node does not acquire the token that corresponds to a file or directory if the local copy is out of date. If a node does have a request refused because of criteria failure, then it is able to act by synchronizing the latest copy of the file or directory before requesting the token again as necessary.

When a node makes a change to a file or directory after it has acquired the corresponding token, then it increments the token version number integer both in the replicating file system layer and in the local token list before announcing the new revision to other nodes for synchronization. This ensures that lost updates do not occur as no node is able to make a change to a file or directory before first acquiring the corresponding token and no node can acquire the corresponding token if it has an old version.

Whenever an operation such as a file creation, modification, or deletion fails and the cause for the failure can automatically be eliminated, then the system eliminates the cause automatically and retry the operation such that intermediate failure is hidden from the user. For example, if the user attempts to open a file but a more recent version of the file is held on a different node, the attempt to acquire the corresponding token will fail. The node on which the user made the original request automatically replicates the newer version of the file, requests the token again, successfully acquires it and allows the user to continue; thus the intermediate failure has been hidden from the user.

In a second embodiment, a file server is provided having the same features of that of the first embodiment and the token data additionally comprising an instance number and a name sequence number (nsn). When a token is created it is assigned an instance number which remains constant during its lifetime. This instance number is incorporated into the token identifier such that the combination of instance number and filename are used to identify a particular token.

Each node's replicating filesystem layer additionally keeps track of the nsn and instance numbers as well as the version number as in the first embodiment against every file and directory on the system. This tracking is referred to as local replica tracking.

When a token is requested in this embodiment, the requesting file server makes this request subject to the criteria that both the sequence number and the name sequence numbers in the token data are equal to or lower than the version number and name sequence number held against the corresponding file in local replica tracking on the requesting server. The server that receives this request will only pass the token if this criteria is held. This prevents the token being passed unless the requesting server already holds both the latest version of the file and the correct entries for any subdirectories.

Additionally each node keeps a list in the replicating filesystem layer referred to as the deletion cache. Each entry in this list comprises a filename, a corresponding instance number and a list of outstanding nodes. Entries in the deletion cache are expired automatically after a defined period of time.

Each file or directory is a child that sits under a parent directory in the file system. The child has an associated child token and the parent an associated parent token. The rules for the management of the nsn are referred to as the nsn management rules and are as follows: when a new token is created, the parent token nsn is incremented by one and this new value is also assigned to the nsn of the new token; and as a token is deleted, the parent token nsn is repeatedly incremented until it is at least the value of its own nsn, with a minimum increment of one. The value of the instance number of a new token is copied from the value of the parent nsn after the parent nsn has been incremented in this way. Other incrementation schemes may be possible.

As an example, following the rules above, consider the following. Tokens will be denoted (name, instance number, nsn). A root token has values (root, 1, 1). A directory 'foo' is created under root, the root nsn is incremented by one giving (root, 1, 2) and the new directory has values (foo, 2, 2). A directory 'bar' is created under 'foo', the 'foo' nsn is incremented by one giving (foo, 2, 3) and the new directory has value (bar, 3, 3). The directory 'bar' is deleted, the 'foo' nsn is incremented by one giving (foo, 2, 4). A directory 'bar' is created again under 'foo', the 'foo' nsn is incremented by one giving (foo, 2, 5) and the new directory (bar, 5, 5).

A token is created when its corresponding file is created. In order to create a file it must first be determined that an identical file has not been created on a different file server since prior to synchronisation, this new file would not yet be replicated across all servers.

To delete a token, a node first acquires the corresponding token using the procedure detailed above. It then adds the token data to the deleting token list and deletes the token data from the local token list and the local token cache. The token passing layer then sends a message to each other node stating that it holds the token and that it is deleting the token. When a node receives such a message it informs its replicated file system layer that the token is being deleted. This is referred to herein as a deletion notification. The replicated file system layer then acts as described below and the token passing layer responds back to the node that is performing the deletion confirming that the token deletion has been successful. Retries are sent until all nodes have responded and confirmed that they have deleted the token. Once all nodes have responded, then the node performing the deletion removes the token from its deleting token list.

Referring to figure 3, when a user attempts to create a new file or directory on a file server on a particular node S110, the replicating file system layer first requests the token that corresponds to the parent directory of the file or directory to be created S111. In this scenario this token is referred to as the parent token. The parent token is requested subject to the

criteria that both the version number and nsn from the local replica tracking associated with the parent directory are equal to or exceed the corresponding version number and nsn on the parent token being requested. If the parent token is not successfully acquired then the creation attempt fails S119. If the parent token is successfully acquired then the nsn management rules are followed to update the parent token S113 and to create a child token by adding a new entry to the local token list S114. The entry in local replica tracking that corresponds to the parent token is updated to match the updated nsn in the parent token S115. A new entry is added to local replica tracking to match the child token that has been created S116. The operation succeeds S117 and the new file or directory is created. The node then announces the creation by sending a notification to each other node S118. This notification comprises the token identifier and instance number that correspond to the file or directory that has been created. The notification is sent in the form of a message using the message caching layer. This type of notification is referred to herein as a creation notification.

Referring to figure 6, when a user attempts to delete a file or directory on a file server on a particular node S140, the replicating file system layer first requests both the token that corresponds to the parent directory of the file or directory being deleted S141 and the token that corresponds to the file or directory being deleted S142. In this scenario, the token that corresponds to the parent directory is referred to as the parent token and the token that corresponds to the file or directory being deleted is referred to as the child token. Both tokens are requested subject to the criteria that both the version numbers and nsns from local replica tracking that are associated with the parent and child tokens respectively are equal to or exceed the corresponding numbers on the corresponding parent and child tokens. If both tokens are not successfully acquired then the deletion attempt fails S14A. If both tokens are successfully acquired then the nsn of the parent token S144 and the corresponding nsn in local replica tracking S145 is incremented following the nsn management rules, the entry corresponding to the child token in local replica tracking is deleted S146, a corresponding entry is added to the deletion cache S147 and the file or directory deletion proceeds as described in the first embodiment S148. This results in the deletion of the child token which causes the replicating file system layer on every other node to be notified of the token deletion via each corresponding token passing layer S130. The token identifier and instance number of the corresponding file or directory that has been deleted is thus included in this notification. Additionally the nsn increment that was used by the nsn management rules is added to this notification. This type of notification is referred to herein as a deletion notification.

Referring to figure 5, when a node receives a deletion notification S130, it does the following. If the deletion cache contains an entry with a matching token identifier and instance number S131 then the notification is ignored S138. If local replica tracking contains an entry with a matching token identifier but with a higher instance number S132 then the notification is

ignored S138. If local replica tracking contains any entries that are children of the token identifier in the notification S133 then the notification is ignored S138. Otherwise the token identifier and instance number are added to the deletion cache S134, the nsn of the parent entry in local replica tracking is incremented by the nsn increment from the notification S135 and the entry in local replica tracking with the identifier that matches the token identifier in the notification is deleted S136. The node then deletes the file that corresponds to the deletion notification S137.

Referring to figure 4, when a node receives a creation notification S120, it does the following. If the deletion cache contains an entry with a matching token identifier and instance number S121 then the notification is ignored S129. If an entry in local replica tracking with the same identifier as the identifier in the notification exists but with a higher instance number than the instance number in the notification S122 then the notification is ignored S129. If an entry in local replica tracking exists with the same identifier as the identifier in the notification but with a lower instance number than the instance number in the notification S123 then a deletion notification is created with the same identifier but with the lower instance number from local replica tracking. This deletion notification is then processed as if it has arrived from the network S124. If the deletion notification is ignored S125 then the original creation notification is also ignored S129. Otherwise processing of the original creation notification continues. If an identifier corresponding to the parent directory of the directory or file referred to by the identifier in the notification does not exist in local replica tracking S126 then the notification is ignored S129. Otherwise an entry in local replica tracking is created that corresponds to the token identifier in the notification with the initial nsn and instance number of this entry taken from the instance number in the notification and with a version number of zero S127. The parent nsn in local replica tracking is then incremented by one S128.

When the acquisition of a token fails because of an nsn criteria failure, the requesting node attempts to resynchronise the corresponding entry in local replica tracking from the node that responded with the highest nsn in its failure reply. The requesting node holds processing on all incoming notifications until the synchronisation is complete. The node that responds to the synchronisation request only does so if it holds the corresponding token; otherwise the synchronisation attempt fails. The response to the synchronisation request comprises the nsn of the entry requested and a list of all child entries. This list is generated atomically. On receiving the response, the requesting node updates its entries in local replica tracking to match and then recursively synchronises any newly created child entries using the same procedure.

In this embodiment, tokens in the system can be categorised into directory tokens and file tokens. Directory tokens correspond to directories in the replicated file system and file tokens

correspond to files in the replicated file system. Modifications to files and to file and directory metadata are arbitrated by the use of tokens in the same manner as they are in the first embodiment. As files cannot have children, the nsn and related management remain effectively unused in file tokens and are only meaningful for directory tokens. Directory tokens are acquired by a file server in the same way as file tokens.

In a third embodiment, a file server is provided having the same features of that of the second embodiment. In addition, the replicated file system interacts with the user via hooks provided by the operating system. For example, Linux provides FUSE for this purpose. The user can then access the local file system as normal, and the replication layer manages replication and intercepts file access calls such as open to make sure that tokens are present when required.

In a fourth embodiment, a file server is provided having the same features of that of the second embodiment. In addition, software running the replicated file system on a system provides the same interface to LAN-based non-replicated file systems using standard network protocols. For example the replicated file system would be manifested as an application on a system which presents an NFS or SMB/CIFS interface to client computers. A further variation of this embodiment uses hooks provided in existing network file system serving software, such as the vfs hooks provided with Samba.

In a fifth embodiment, a file server is provided having the same features of that of the second embodiment. Here, the replicated filesystem is incorporated into a hardware device which can store data. The device can link to one or more other devices to provide a distributed, replicated, multi-master file system. By linking to other network attached storage devices with the same capability, the user gets a fully distributed system.

The user presentation and interface is designed such that users see the system as a LAN-based shared collaboration area with the property that it automatically synchronises to systems at other locations.

The devices link together either by using a pre-existing VPN (for example corporate VPNs are common), a VPN managed by the device manufacturer, a third party system (such as Hamachi), or unencrypted. The connection can be mediated by one or more central servers (such as with ICE/STUN/TURN and/or dynamic DNS services) to assist with locating nodes with dynamic addresses and traversing NAT devices.

The devices can optionally be sold in multipacks already linked to make installation easy for buyers.

In a sixth embodiment, a file server is provided having the same features of that of the second embodiment. Here the local file system access method is enhanced so that users do not have to have a permanent network installation. One of the nodes in the system is presented at a centrally accessible location and mediates connections between other nodes. The user presentation and interface is designed so that although access to the replicated file system is local (except for non-real-time replication and token passing latency), users see the system as a shared, central collaboration area.

Token transfer may be managed by a combination of user intervention, automatic requests on file access and automatic return of tokens to the central location on file closure. This embodiment would be particularly useful for mobile users such as those using laptops.

In a seventh embodiment, a file server is provided having the same features of that of the second embodiment. Additionally there is an extra component of the system, which is a application that can be installed on users' systems. The application will run in the background and send and receive information about ongoing file system operations from the replicated file system layer and/or the file system interaction layer.

In the case that the user attempts an operation that cannot be carried out immediately, the application will be able to prompt the user with further instructions.

For example, if a user attempts to rename a directory tree and this operation is carried out via a file system hook, then the hook is expected to return quickly with either a success or a failure. However, this replicated, distributed system may not be able to do so if it cannot acquire all the required tokens quickly due to bandwidth or latency constraints. Further, if an operation fails then the hook may only allow the system to present a small number of error codes, none of which represent the actual error that took place. In this situation, depending on configuration the system may do any of the following:

1. The hook returns failure, but the auxiliary application prompts the user to carry out the same operation with a user interface that can deal appropriately with the delay, optionally displaying progress as it does so.

2. The hook returns failure with the most appropriate code, and the auxiliary application additionally notifies the user with a more accurate error, optionally together with analysis, alternative options and with processes to correct the error.

3. The hook blocks until the operation is complete, but the auxiliary application notifies the user of the reason for the delay and optionally displays progress.

4. The hook blocks until some reasonable time has been exceeded, and the auxiliary application interacts with the user as in item 3.

This system may apply to any operation, not just to renaming directories. For example, a delay in acquiring a token to open a file may also present similar behaviour.

In an eighth embodiment, a file server is provided having the same features of that of the second embodiment. Additionally the system of nodes comprises a single configuration token with a special name in a namespace not shared by any other layer. The token data for this configuration token is set to contain the subset of the system configuration for which compatibility is required.

During startup, a node will acquire the configuration token and lock it by vetoing token transfer to any other node until the startup procedure has either completed or failed. The minimum configuration required to locate the other nodes in order to communicate and acquire the token will be taken from either a cached previous good configuration or supplied by the system administrator.

Once the configuration token has been acquired, the token data is examined to determine the correct configuration, the system configures itself accordingly, the token veto is released and startup may continue.

Alternatively, if the system administrator has specified a particular configuration, the configuration token request may be set to contain criteria that the configuration must match the supplied one. The startup will then only succeed if the supplied configuration matches the configuration held on the token.

Before the configuration token data is changed, the node performing the change will first acquire the token, lock it, and either verifiably push the new configuration data to all other nodes or verify that the system is not running on any of the other nodes before unlocking it again. This ensures that all nodes are running a compatible configuration and provides a means to easily change the configuration across all nodes in a reliable manner.

In a ninth embodiment, a file server is provided having the same features of that of the second embodiment. Additionally a read-only configuration module is provided which enables an administrator to determine how the system handles read-only operations.

In the replicated filesystem described, a node will normally have a copy of each file but unless the token is held there is no guarantee that the file is at the latest revision. If a token or even the entire network is unavailable it is still possible to present the user of the replicated filesystem with a read only copy.

A disadvantage is that a file while opened read-only is typically not modifiable. If updates occur on other nodes then the local node will be unable to update its live copy until the local access is closed and in typical use this time delay may be measured over days, if for example a file has been left open. The read-only configuration module allows the administrator to select the option to present the user with a warning using the auxilliary GUI if this occurs. A second selectable option is to go without this feature in order to avoid confusing users with out of date copies.

Additionally, a node will allow and count multiple opens to a single file and not treat the file as closed until all handles have been closed. Whilst the system uses mandatory locking between different nodes, it may permit concurrent access on a single node without affecting integrity to support these programs.

In a tenth embodiment, a file server is provided having the same features of that of the second embodiment. Additionally, means are provided for dealing with stuck tokens where in case of network partitioning, a token may end up on the opposite side of the partition from where it is needed.

1. The user could specify for each file or set of files which nodes have priority for the tokens, and each node could automatically pass the token to the priority site once the token is no longer required.

2. When a stuck token does occur, the token could be transferred manually if communication between the two sites by some other means is possible for example telephone or fax. When the node requesting the token fails to acquire it because not all nodes are reachable, it could generate a request code for the token based on the token request message. The user could communicate this request code to a user at the other side of the partition, and a reply code could be sent back in the same way.

3. If a node has become unavailable permanently, then a user or administrator can declare this to one of the other nodes. That node will then contact all the remaining nodes to remove the lost node from the set of nodes in the system. Once this is complete, any node can appropriate any of the stuck tokens in the usual manner.

In an eleventh embodiment, a file server is provided having the same features of that of the second embodiment. Additionally, means are provided to ensure temporary files are not replicated. The system may match these files by examining their hidden status or by matching their filenames against a set of patterns, and bypass usual access control through the replicated file system so that their use is entirely local and not replicated.

In a twelfth embodiment, a file server is provided having the same features of that of the second embodiment. Additionally, standard access control features are provided. Access

control is available to the administrator as an independent option on each connected file server.

When access control is enabled on a particular node, the administrator chooses the authentication realm to which the given node belongs. Nodes on the system of connected file servers do not necessarily have to belong to the same authentication realm or any authentication realm.

Access control lists are maintained for each file on a given node in the same manner as in existing implementations and is replicated as file metadata in the same manner as in the first embodiment.

When nodes on the system of connected file servers do not all belong to the same authentication realm, access control lists may contain entries that refer to security identifiers which cannot be understood by a particular node. In this embodiment, access control ignores such entries while acting on the remaining entries as normal.

Attention is directed to all papers and documents which are filed concurrently with or previous to this specification in connection with this application and which are open to public inspection with this specification, and the contents of all such papers and documents are incorporated herein by reference.

All of the features disclosed in this specification (including any accompanying claims, abstract and drawings), and/or all of the steps of any method or process so disclosed, may be combined in any combination, except combinations where at least some of such features and/or steps are mutually exclusive.

Each feature disclosed in this specification (including any accompanying claims, abstract and drawings) may be replaced by alternative features serving the same, equivalent or similar purpose, unless expressly stated otherwise. Thus, unless expressly stated otherwise, each feature disclosed is one example only of a generic series of equivalent or similar features.

The invention is not restricted to the details of the foregoing embodiment(s). The invention extends to any novel one, or any novel combination, of the features disclosed in this specification (including any accompanying claims, abstract and drawings), or to any novel one, or any novel combination, of the steps of any method or process so disclosed.

Claims

1. A method of controlling the opening of a file, the file being one of a plurality of files on a replicated file server connected to one or more other replicated file servers over a network, each of said plurality of files being associated with a corresponding unique file token, the file token being held on one of the file servers only, the method comprising the steps of:

the file server attempts to acquire the file token; and

if the file server acquires the file token, the file server permits the opening of the file.

2. The method of claim 1, wherein the file server attempts to acquire the file token by first checking if it holds the file token and if it does hold the file token it permits the opening of the file without requesting the file token from another file server.

3. The method of claim 2, wherein if the file server does not hold the file token, the file server attempts to acquire the file token by requesting the file token from the other replicated file servers.

4. The method of claim 3, wherein the request for the file token is made by transmitting a message peer-to-peer from the requesting file server to each of the other file servers.

5. The method of claim 3 or claim 4, wherein if any of the other file servers refuses to pass the file token, the requesting file server does not permit the opening of the file.

6. The method of any one of claims 3 – 5, wherein if none of the other file servers have the file token, the requesting file server recreates the file token and permits the opening of the file.

7. The method of claim 6, wherein the file token is recreated using information stored on the requesting file server.

8. The method of claim 7, wherein the file token is recreated using information stored on the requesting file server and one or more of the other file servers.

9. The method of any one of claims 3 – 8, wherein the file token comprises a file version and one of the plurality of the other file servers does not pass the file token if the file version of the file token is greater than the file version of the file held on the requesting file server.

10. The method of any preceding claim, wherein each file server maintains a local token list of the file tokens it currently holds.

11. The method of any preceding claim, wherein opening a file comprises opening a file for writing or for reading.

12. The method of any preceding claim, wherein each file server maintains a local token cache of the file tokens it has previously held.

13. The method of claim 12, wherein the requesting file server recreates the file token using information stored in its local token cache.

14. The method of claim 13, wherein the requesting file server recreates the file token using information stored in its local token cache and the local token cache of one or more of the other file servers.

15. The method of any one of claims 3 – 14, wherein if the requesting file server has not received the file token and does not receive a response from each of the other file servers within a defined period of time, a timeout occurs and the requesting file server does not acquire the file token and is not permitted to open the file.

16. The method of any preceding claim, wherein the files are stored in directories in a hierarchical structure where each file and directory contained within a parent directory is a child of that parent directory, each directory has a corresponding unique directory token held on one of the file servers only, and the directory structure is replicated across the file servers, where a file server does not permit the creation or deletion of a child file or directory unless it holds the parent directory token.

17. The method of claim 16, wherein the file server first checks if it holds the directory token and if it does hold the directory token it permits the creation or deletion of a child file or directory.

18. The method of claim 17, wherein if the file server does not hold the directory token, the file server acquires the directory token by requesting the directory token from all other file servers; and acquires the directory token from one of the other file servers.

19. The method of any one of claims 16 – 18, wherein the directory token comprises directory token data relating to the creation and deletion of first generation child entries.

20. The method of claim 19, wherein the directory token data relates to the creation and deletion of first and subsequent generation child entries.

21. The method of any one of claims 16 – 20 wherein each file and directory additionally comprises a name sequence number and the directory token comprises directory token data comprising the directory name sequence number, whereby one of the plurality of other file servers does not pass the directory token if the directory token name sequence number is greater than the directory name sequence number.

22. The method of any claim 21 wherein when a new token is created, the parent token name sequence number is incremented.

23. The method of claim 22 wherein, when a token is deleted, its parent token name sequence number is incremented.

24. A file server arranged to perform the method of any preceding claim.

25. A computer-readable recording medium having recorded thereon program code instructions arrange to cause a file server to execute the method of any of claims 1 – 23

26. A method of controlling the opening of a file substantially as described with reference to figures 1 – 6 of the accompanying drawings.

27. A file server as described with reference to figures 1 – 6 of the accompanying drawings.

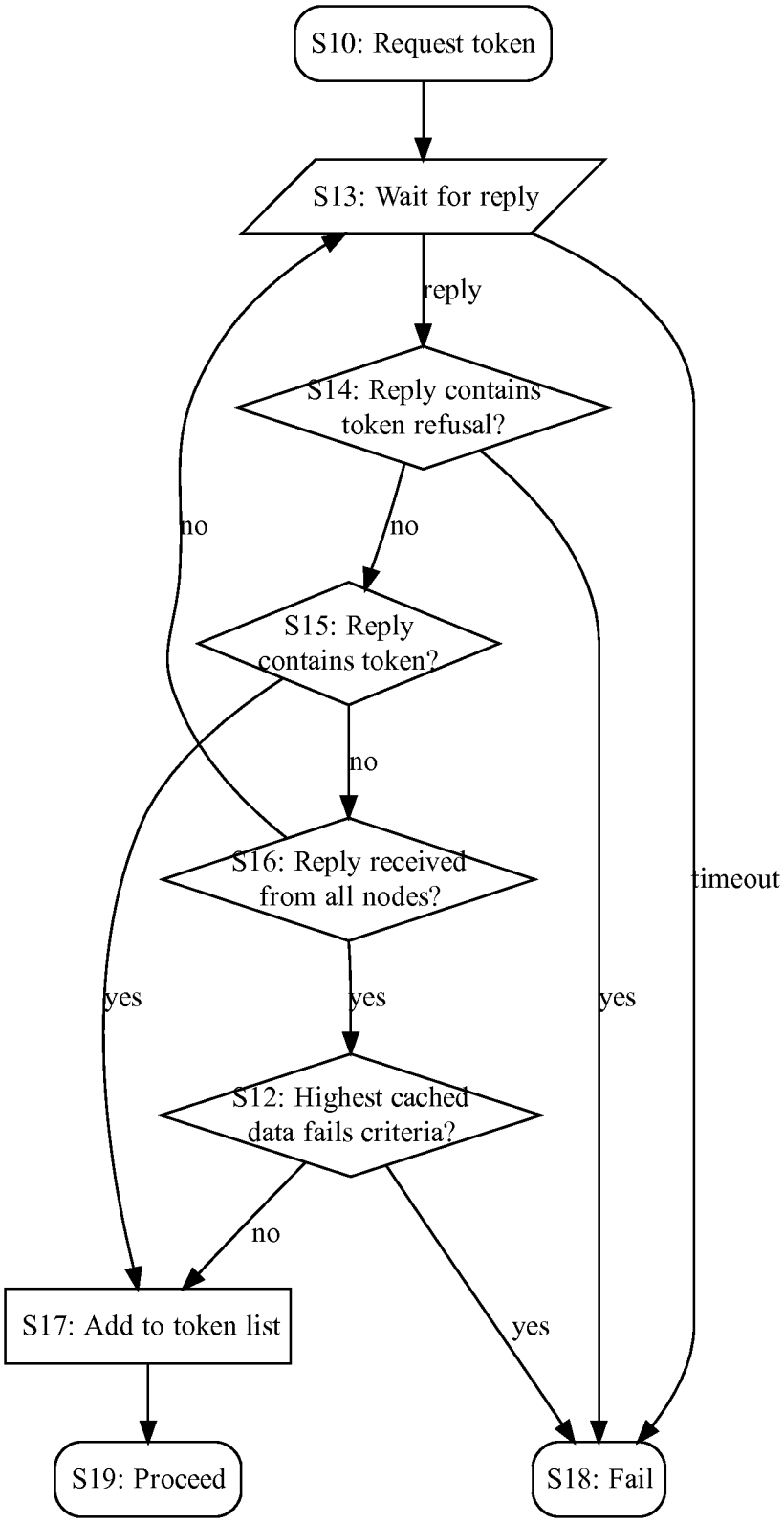

S10: Request token
S13: Wait for reply
reply
S14: Reply contains token refusal?
no
no
S15: Reply contains token?
no
S16: Reply received from all nodes?
timeout
yes
yes
yes
S12: Highest cached data fails criteria?
no
yes
S17: Add to token list
S19: Proceed
S18: Fail

Figure 1

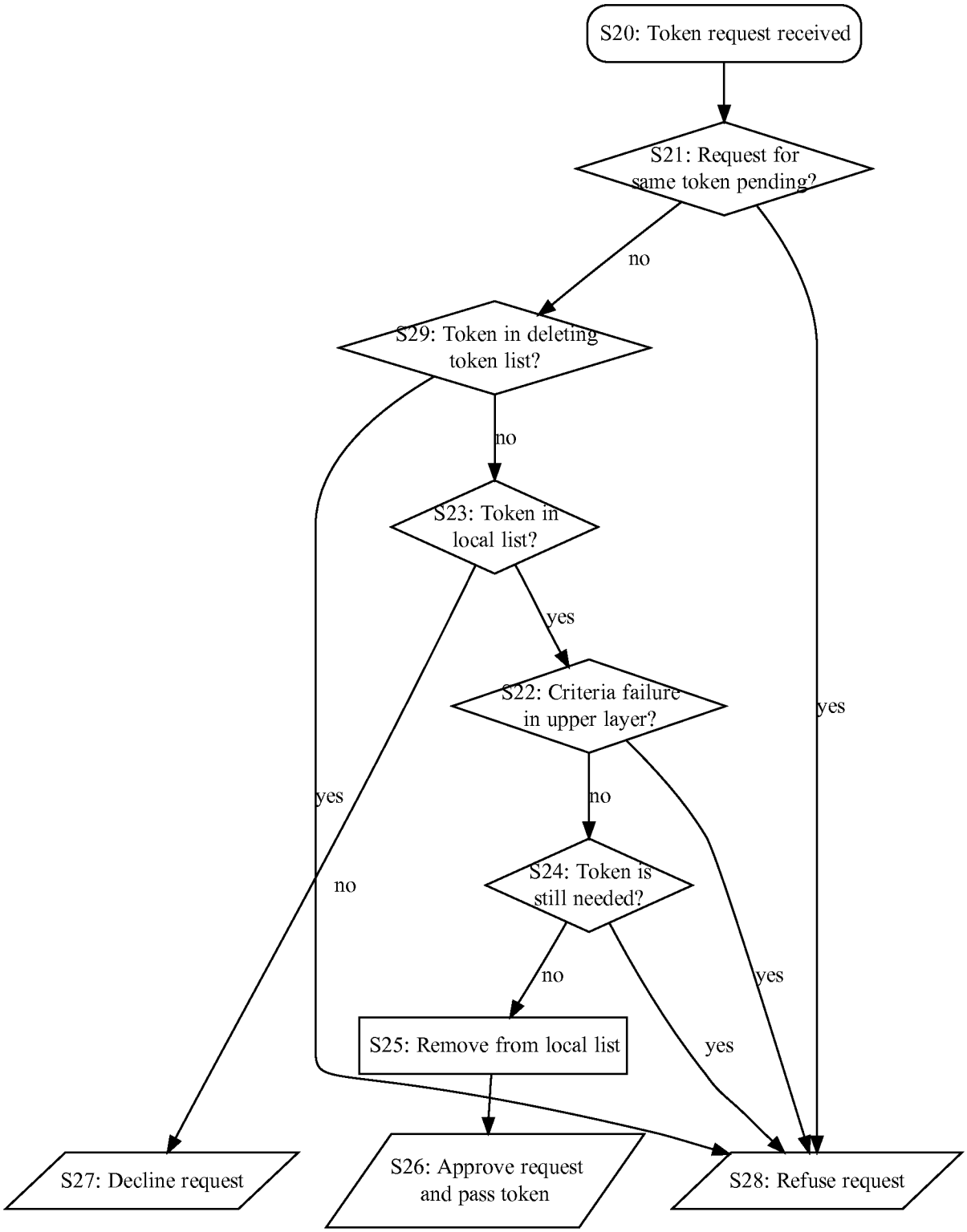

S20: Token request received
S21: Request for same token pending?
no
yes
S29: Token in deleting token list?
no
yes
S23: Token in local list?
yes
no
S22: Criteria failure in upper layer?
yes
no
S24: Token is still needed?
no
yes
S25: Remove from local list
yes
S27: Decline request
S26: Approve request and pass token
S28: Refuse request

Figure 2

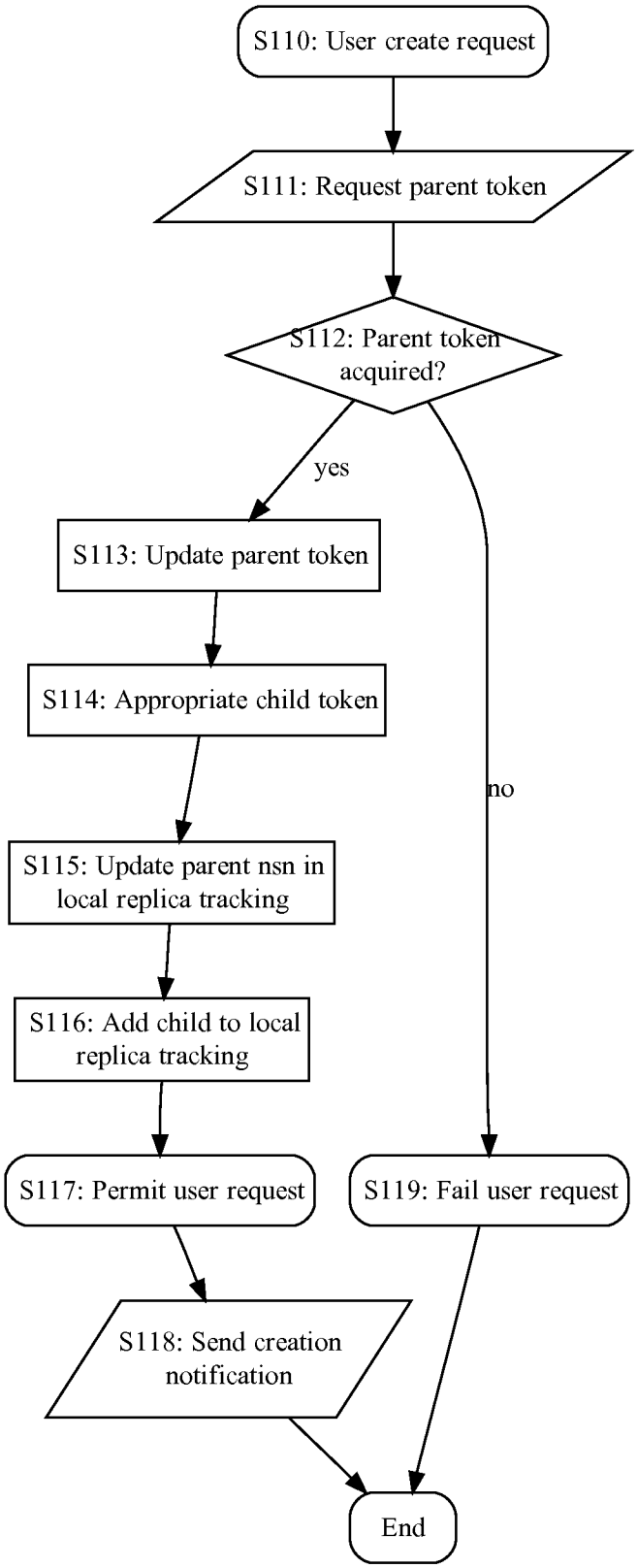

S110: User create request
S111: Request parent token
S112: Parent token acquired?
yes
no
S113: Update parent token
S114: Appropriate child token
S115: Update parent nsn in local replica tracking
S116: Add child to local replica tracking
S117: Permit user request
S119: Fail user request
S118: Send creation notification
End

Figure 3

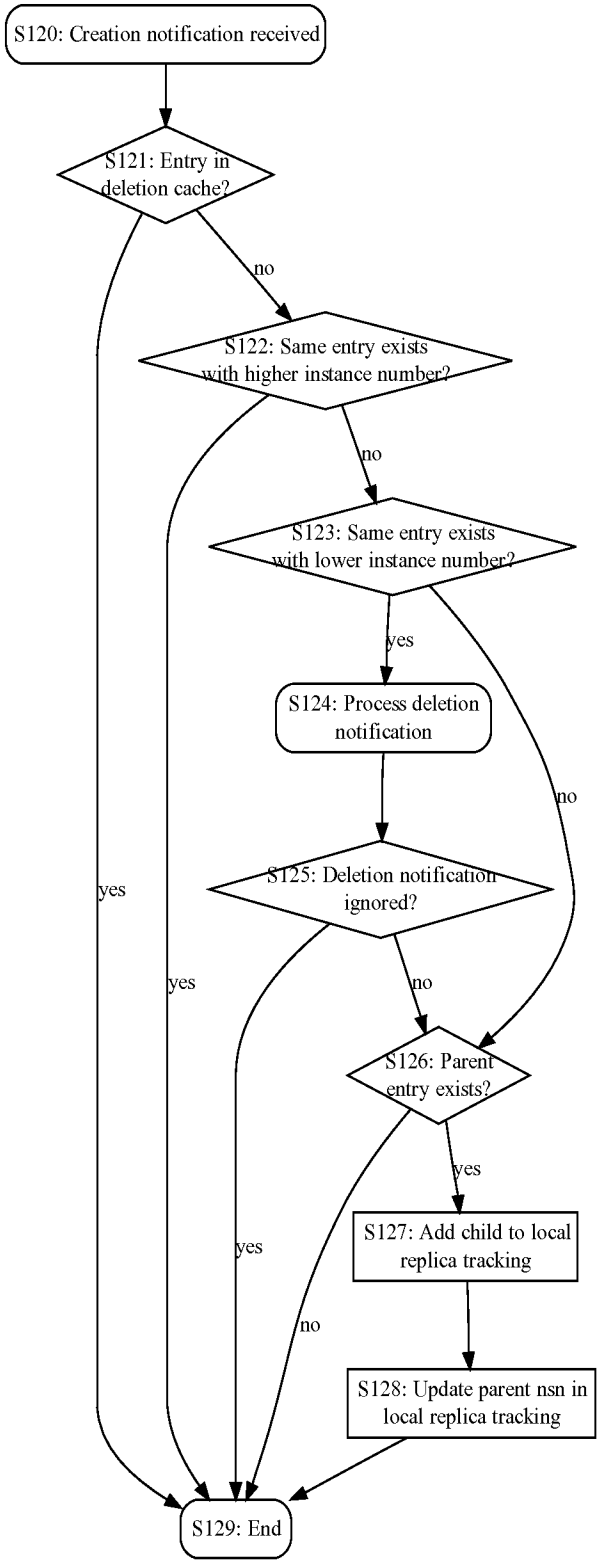

S120: Creation notification received
S121: Entry in deletion cache?
no
S122: Same entry exists with higher instance number?
no
S123: Same entry exists with lower instance number?
yes
S124: Process deletion notification
no
S125: Deletion notification ignored?
yes
yes
no
S126: Parent entry exists?
yes
S127: Add child to local replica tracking
yes
no
S128: Update parent nsn in local replica tracking
S129: End

Figure 4

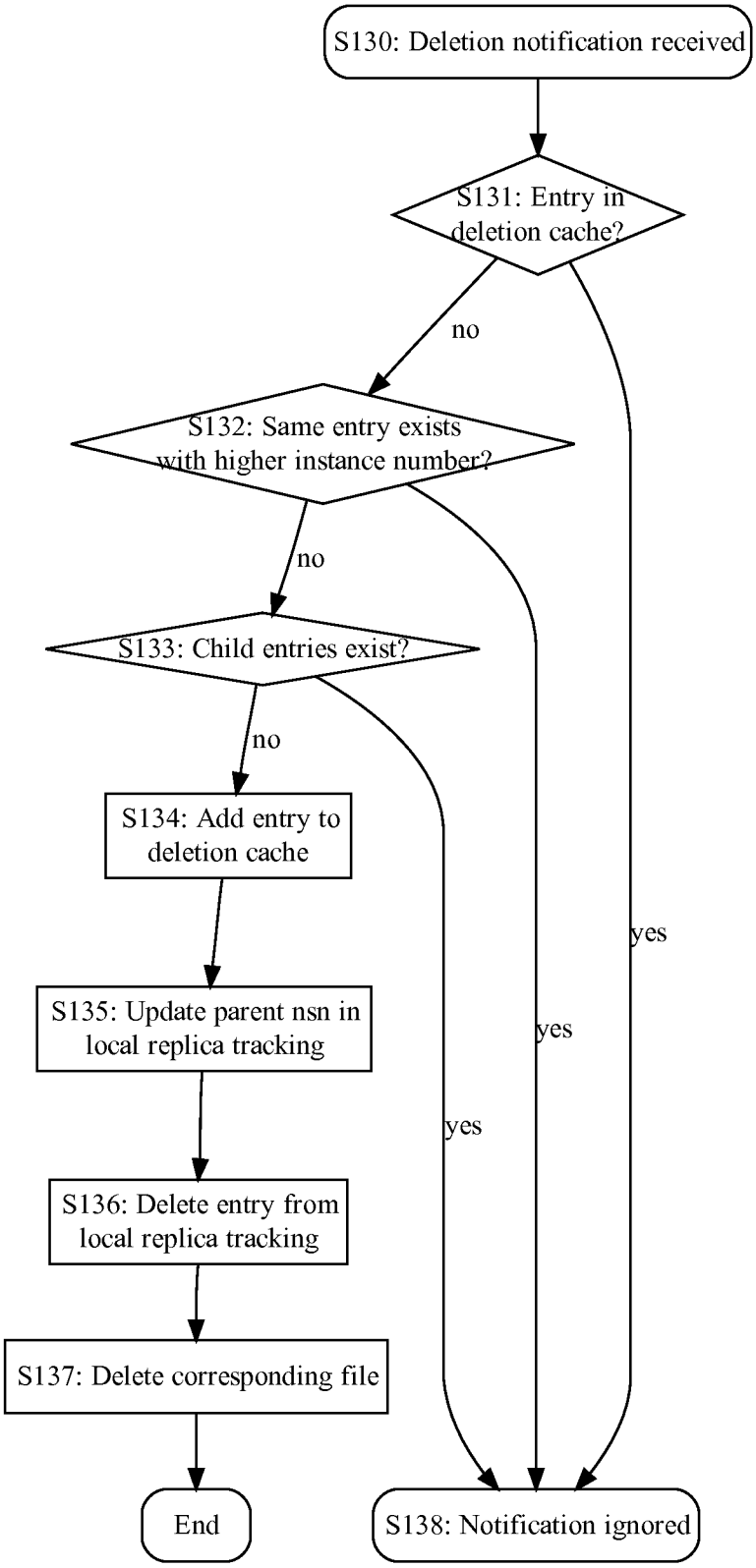

S130: Deletion notification received
S131: Entry in deletion cache?
no
yes
S132: Same entry exists with higher instance number?
no
yes
S133: Child entries exist?
no
yes
S134: Add entry to deletion cache
S135: Update parent nsn in local replica tracking
S136: Delete entry from local replica tracking
S137: Delete corresponding file
End
S138: Notification ignored

Figure 5

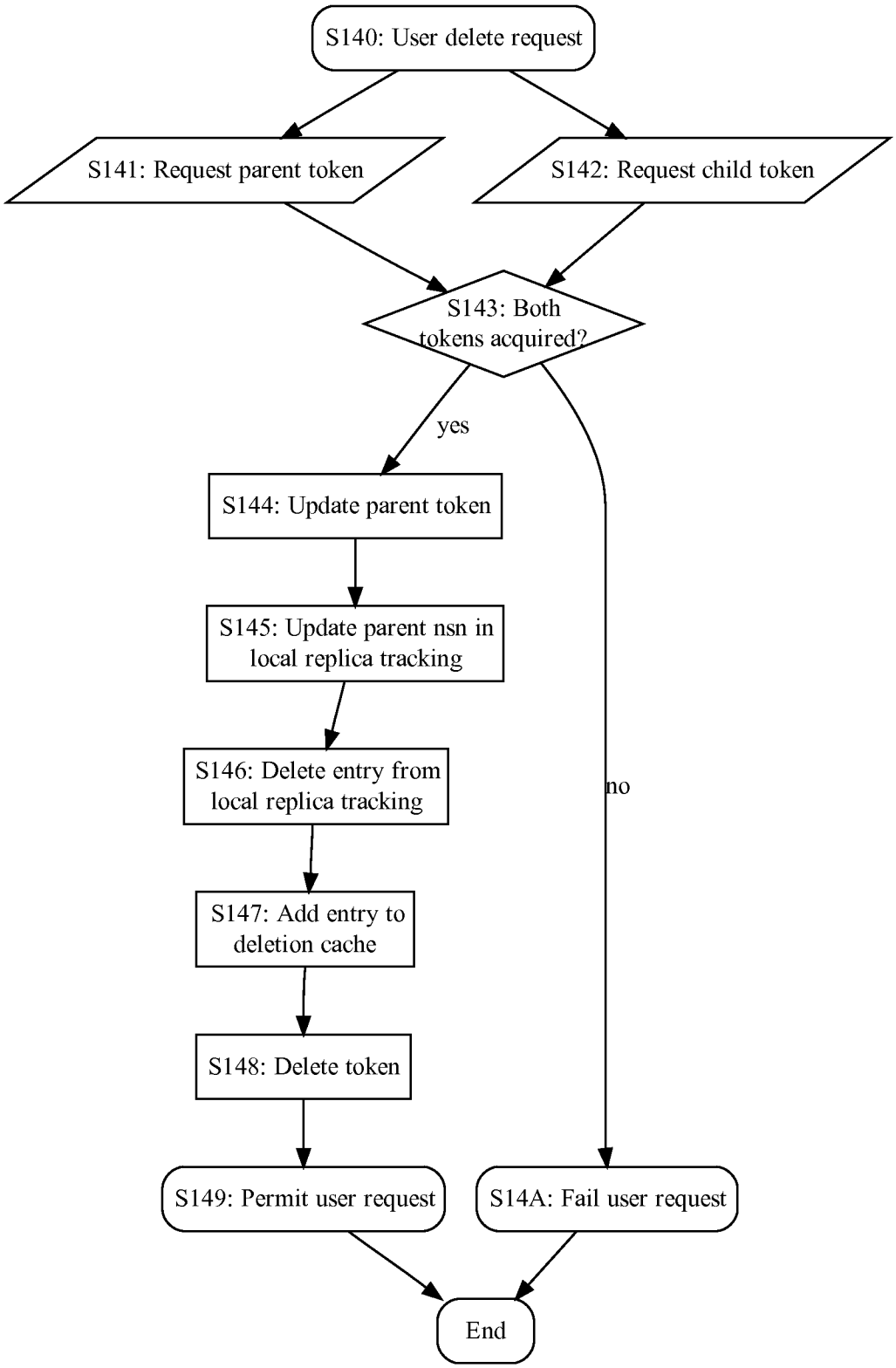

S140: User delete request
S141: Request parent token
S142: Request child token
S143: Both tokens acquired?
yes
no
S144: Update parent token
S145: Update parent nsn in local replica tracking
S146: Delete entry from local replica tracking
S147: Add entry to deletion cache
S148: Delete token
S149: Permit user request
S14A: Fail user request
End

Figure 6

INTERNATIONAL SEARCH REPORT

International application No
PCT/GB2010/050754

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F17/30
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2008/319996 A1 (COOK STEVEN D [US]) 25 December 2008 (2008-12-25) paragraph [0008] - paragraph [0009] paragraph [0021] - paragraph [0023] -----	1-27
A	US 5 689 706 A (RAO CHUNG-HWA HERMAN [US] ET AL) 18 November 1997 (1997-11-18) * abstract column 2, line 40 - column 3, line 17 -----	1-27
A	EP 0 720 091 A2 (IBM [US]) 3 July 1996 (1996-07-03) * abstract page 1, line 7 - line 12 column 6, line 42 - column 8, line 30 ----- -/--	1-27

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 July 2010	28/07/2010

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	de Castro Palomares

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/GB2010/050754

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2009/049153 A1 (MCFADDEN RENATA RAND [US] ET AL) 19 February 2009 (2009-02-19) * abstract paragraph [0006] paragraph [0028] - paragraph [0030] -----	1-27
A	EP 0 694 839 A2 (AT & T CORP [US]) 31 January 1996 (1996-01-31) cited in the application * abstract page 3, line 1 - line 25 page 11, line 27 - page 12, line 45 -----	1-27

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/GB2010/050754

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2008319996 A1	25-12-2008	CN 101329681 A	24-12-2008
US 5689706 A	18-11-1997	NONE	
EP 0720091 A2	03-07-1996	DE 69521016 D1	28-06-2001
		JP 3062070 B2	10-07-2000
		JP 8263355 A	11-10-1996
		US 5634122 A	27-05-1997
US 2009049153 A1	19-02-2009	NONE	
EP 0694839 A2	31-01-1996	CA 2152528 A1	30-01-1996
		DE 69522394 D1	04-10-2001
		DE 69522394 T2	23-05-2002
		JP 3476973 B2	10-12-2003
		JP 9091185 A	04-04-1997
		JP 2003263355 A	19-09-2003