the wrapper information to the gatekeeper. This wrapper information tells the gatekeeper the rule history indicating which rules fired and hence the reason the message was gated, and the particular routing history for the message.

FIG. **19** illustrates a sample user interface of the GKI **107**, the main gatekeeper screen **1901**. When a gatekeeper logs into a GPO **106**, she identifies which gatekeeper role she has; the GKI **107** then displays in the gatekeeper screen **1901** the messages **1916** that have been gated to that gatekeeper. For each message, there is shown the sender, location, and the subject line of the message. In addition, a status value **1903** informs the gatekeeper whether the message has only been gated, or reviewed. The reason **1905** that the message has been gate is also shown, extracted from the rule history information in the wrapper of the message.

The main gatekeeper screen **1901** also displays a list of the mailbag folders that have been defined by the gatekeeper.

From the GKI **107**, the gatekeeper can release **1909**, return **1911**, forward **1913**, or file **1915** any message, by selection of the appropriate button.

The GKI **107** also provides the gatekeeper the ability to read and edit individual messages. To review a message, the gatekeeper selects the message and clicks on the review button **1907**. FIG. **20** illustrates a sample screen for reviewing messages. This screen **2001** includes specific fields identifying the sender, specified recipients, subject line, creation and arrival dates. In addition, there is shown the reason **2003** the message was gated, the release date **2005** of the message (which is its expiration date as determined when the message was placed in the inbox), and its retain date **2007**, if any. The retain date **2007** is the date after which the GPO **106** will delete the message from the inbox. These dates may be manually changed by the gatekeeper. The gatekeeper can edit the text of the message in the text field **2009**, for example to remove offensive language, or confidential information.

The gatekeeper can immediately release **2011**, return **2013**, or forward **2015** a gated message. Forwarding a gated message enables the gatekeeper to send the message to another gatekeeper for additional review, as described above with respect to FIG. **4**B; an additional screen is displayed (not shown) into which the gatekeeper can add instructions or information for the next gatekeeper. Forwarding a gated message also enables the gatekeeper to send the message to recipient(s) other than the specified recipient(s) for further processing and disposition.

FIG. **21** illustrates a screen **2100** used by a gatekeeper when returning a **8** message to the sender. When returning a message, the gatekeeper can also include in text field **2101** an explanation to the sender of why the message was not delivered. The gatekeeper may also move a message to a mailbag folder.

In one embodiment, when the gatekeeper moves a message from the inbox to one of the mailbags, the expiration date of the message is updated according to the time parameter of the selected mailbag. In this manner, all messages associated with a specific mailbag have the same expiration date. For example, a gatekeeper's inbox may have a interval time parameter of 10 days, the gatekeeper has set the time parameter for a delete mailbag at 30 days, and the last date the mailbag contents were deleted was Jan. 5, 1997. Assume that a message is received with a timestamp of Jan. 1, 1998. When the message is indexed and placed in the inbox of the gatekeeper, the gatekeeping message index **287** is updated with an expiration date for the message of Jan. 11, 1998. Now, assume that on Jan. 6, 1998 the gatekeeper manually reviews the message and decides to move it to the delete mailbag. The expiration date of the message is automatically reset to Feb. 5, 1998, or 30 days after the last delete. This date will be the expiration date for all messages moved to this mailbag, up until Feb. 6, 1998, when the expiration date will be advanced to Mar. 5, 1998.

Automated Review of Gated Messages

Automated review of gated messages is applied to those messages in the inbox of each gatekeeper which have not been manually reviewed prior to their individual expiration dates. The automated review of messages is provided by additional functionality of the GPO **106**, such as part of the program executive. Automated review applies the rules defined by the gatekeeper in the gatekeeping rule base **289** to each expired message in the inbox. Each gatekeeper can write their own rules in the gatekeeping rule base **289** for handling these gated messages, and can apply more detailed analysis and handling of gated messages. Each gatekeeper defines rules for the gatekeeping rule base **289** in the same manner as described above for the rule base **270**.

While the functional operation of the rule engine **283** is the same as with the rule engine **210** of a REPO **102**, because the rules written by a gatekeeper may be different from the rules at a REPO **102**, a given message may be handled in a different manner. Whereas the checkpoints of the REPO **102** are intended to operate on messages that are being rule processed for the first time, the rules of the gatekeeping rule base **289** are used to apply additional, and if necessary, more detailed review of the gated messages. More particularly, these rules can process the messages on the rule history information itself, in addition to all of the other properties and attributes used in the REPOs. This allows for very detailed rules to be applied to gated messages.

For example, if a REPO **102** gates messages greater than 1 Mb, then the gatekeeping rule base **289** rules can check the role of the sender and release the message for certain senders only, deleting the messages from all other senders.

The rule structure is applicable as before, with rules having both antecedent and consequent components. The rule actions include the gate, delete, release, and return actions described above.

In addition, a review action, and mailbag actions are provided for rules at the gatekeeping rule base **289**. A review action specifies that the message is to be manually reviewed by a gatekeeper. During such review the gatekeeper administrator can edit the message. When a rule with a review action is fired, then the rule engine **283** moves the message to a review mailbag by updating the folder association for the message in the gatekeeping message index **287**. This review action is useful because it helps the gatekeeper identify messages which need to be specifically reviewed, and which the gatekeeper did not previously consider or attempt to review (since the message was automatically reviewed due to it expiring).

Whereas the actions applied by the rules at the REPO **102** are effectively applied immediately, the actions available at the GPO **106** include actions for moving the message into one of the mailbags for delayed, periodic execution of the action for the mailbag. A mailbag action takes as an argument the name of one of the mailbags in the master folder table **290**. The message is then associated with the specified mailbag in the gatekeeping message index **287**. The message will be acted upon according to the appropriate action for the mailbag when the time parameters of the specified mailbag are satisfied. For example, a rule may move a message into a mailbag with a retain date set for the end of the current month, at which time the message is deleted with other messages in the folder.

Referring now to FIG. **22** there is shown a flow graph of the operation of the GPO in providing automatic review of messages. The GPO **106** periodically awakens the distribution engine **284** as a daemon process to process messages that have indexed in the inbox of each gatekeeper. For each message in the inbox (**2202**), the distribution engine **284** determines **2204** whether the message is expired, that is whether the current date is equal to, or greater than the expiration date. If the message is not expired, the distribution engine **284** continues with the next message in the inbox.

If the message is expired, then the distribution engine **284** invokes **2206** the rule engine **283** to apply the rules from the gatekeeping rule base **289** to the message. These rules are applied by the rule engine **283** as described above until one of the rules fires, or all rules are applied. As described above, the rule engine **283** returns an action list to the distribution engine **284** for the message. This action list identifies the action to be taken upon the message. Generally, the distribution engine **284** updates **2208** the gatekeeping message index **287** to indicate the specific action taken for the message. The distribution engine **284** then updates **2210** the distribution list, routing history, and rewraps the message, if necessary (some actions, such as delete, do not need the message to be rewrapped). The distribution engine **284** then continues with the next message. More specifically then, the distribution engine **284** handles the actions as follows:

On a return action, the distribution engine **284** updates the gatekeeping message index **287** to indicate that the message is being returned. The distribution engine **284** rewraps the message with updated history information, to indicate both that the message has been handled by this gatekeeper (storing the gatekeeper id in the wrapper) and to indicate the action taken. The distribution engine **284** continues with the next message.

On a delete action, the distribution engine **284** updates the gatekeeping message index **287** to indicate that the message is being deleted. There is no need to rewrap the message. The message is then deleted. The distribution engine **284** continues with the next message.

On a forward action, the distribution engine **284** updates the gatekeeping message index **287** to indicate that the message is released. The message is rewrapped with the gatekeeper ID and action taken (here forward and the reasons). The distribution list is updated to place the GPO **106** of the new recipient as the next GPO **106** on the list; the current GPO **106** is added at the end of the distribution list. The new recipient may be another gatekeeper, or it may be any other recipient to whom it is useful to send the message for further processing and response. The distribution engine **284** then sends the message to the next GPO on the distribution list. The distribution engine **284** continues with the next message.

On a release action, the distribution engine **284** updates the gatekeeping message index **287** to indicate that the message is released. The distribution engine **284** rewraps the message with the updated routing history so that the current gatekeeper, which is the first one on the routing history list, becomes a past gatekeeper. The next gatekeeper on the routing list is copied to the recipient list for the message. This enables the message to be routed to another gatekeeper, if there is one. This will happen when the original REPO that gated the message indicated that more then one gatekeeper was to review the message. In this case, after the present gatekeeper is done, the recipient list is updated so that the next gatekeeper may receive the message for review.

On a review action, the distribution engine **284** updates the gatekeeping message index **287** to place the message in the review mailbag for subsequent review by the gatekeeper.

As noted, an action for a message may be a mailbag action, which is to move the message to a particular mailbag. Here, the distribution engine **284** updates the gatekeeping message index **287** with an ID of the mailbag (as taken from the action). The distribution engine **284** checks whether the time parameter for the designated mailbag has expired. If so then, the distribution engine **284** performs the function associated with the mailbag, whether to release, delete, forward or so forth, the messages in the mailbag.

Distributed Gatekeeping

A release action allows a message to continue from the GPO **106** unabated by any further rule processing by another other REPO **102** or GPO **106**. Once released, the message is sent to the next GPO **106**, if any on the distribution list are included in the wrapper. This allows for continued handling according to the original REPO **102** rules. Each GPO **106** tags the message with its ID indicating that it has reviewed the message.

Once the last GPO **106** processes the message, the last item on the distribution list is the original REPO **102**; this data item was placed in the wrapper by the distribution engine **230** of the original REPO **102**. Thus, the message is returned to the REPO **102** for further distribution. The next distribution destination may be to another GPO **106** in the network which will apply its own rules. This feature enables very complex rule processing, and fully distributed and independent rule handling by any number of different GPOs **106**.

When the original REPO **102** gets the message back, it reprocesses the message and applies its rule from the rule base **270** again. The list of gatekeepers in the wrapper who have reviewed the message is an "innoculant," and is used to prevent the REPO **102** from re-gating the message back to one of these gatekeepers, though the message may be gated to another gatekeeper at the same GPO **106**. The REPO **102** compares the list of gatekeepers who have seen the message, with the resulting list of gatekeepers from the new rule firing. If there is a match, then the message is not sent to the matching gatekeeper because that gatekeeper has already reviewed the message once before. In this manner, the message is gated only to gatekeepers who have not yet reviewed the message. Reprocessing the message in this manner is desirable because there may have been changes in the rules of the original REPO **102** during the time period the message was being evaluated at the various GPOs **106**, but it is assumed that a gatekeeper who has once reviewed a message need not review it again.

In summary, the present invention, in its various embodiments, provides a system, method, and various software products for controlling the distribution of data objects, including e-mail messages, on a communication network. The present invention applies business rules which can implement corporate communication policies, to such data objects as they are transferred through a post office or similar mail transfer agent. The business rules that are satisfied by the properties and attributes of a data object generate a set of actions to be applied to the data object. These actions are applied to the data object. The data object may be gated such that it is not delivered to its specified recipients, but rather to a gatekeeper who may manually review the message, or may allow it to be further reviewed automatically by yet another set of business rules. In this manner, fully distributed gatekeeping of any and all messages and data objects can be enforced in a networked environment.

We claim:

1. A post office for receiving and redistributing e-mail messages on a computer network, the post office comprising:

a receipt mechanism that receives an e-mail message from a sender, the e-mail message having at least one specified recipient;

a database of business rules, each business rule specifying an action for controlling the delivery of an e-mail message as a function of an attribute of the e-mail message;

a rule engine coupled to receive an e-mail message from the receipt mechanism and coupled to the database to selectively apply the business rules to the e-mail message to determine from selected ones of the business rules a set of actions to be applied to the e-mail message; and

a distribution mechanism coupled to receive the set of actions from the rule engine and apply at least one action thereof to the e-mail message to control delivery of the e-mail message and which in response to the rule engine applying an action of deferring delivery of the e-mail message, the distribution engine automatically combines the e-mail message with a new distribution list specifying at least one destination post office for receiving the e-mail message for review by an administrator associated with the destination post office, and a rule history specifying the business rules that were determined to be applicable to the e-mail message by at least one rule engine, and automatically delivers the e-mail message to a first destination post office on the distribution list instead of a specified recipient of the e-mail message.

2. The post office of claim 1, wherein the actions are selected from a group comprising:

deleting an e-mail message instead of delivering the e-mail message to a specified recipient;

delivering the e-mail message to a recipient other than a specified recipient;

returning the e-mail message to the sender; and,

deferring delivery of the e-mail message to a later time.

3. The post office of claim 1, wherein, each business rule includes at least one antecedent, each antecedent defining an attribute, an operator, and value, wherein the attributes are selected from a group including:

a number of attachments;

a size of attachments; and

a text of the message.

4. The post office of claim 1, wherein, each business rule includes at least one antecedent, each antecedent defining an attribute, an operator, and value, wherein the attribute is determined by statistical performance data of the post office.

5. A post office for receiving and redistributing e-mail messages on a computer network, the post office comprising:

a data base storing an organizational hierarchy of the business, the hierarchy including a plurality of roles, each role associated with a user;

a receipt mechanism that receives an e-mail message from a sender, each e-mail message having at least one user specified as a recipient;

a database of business rules, each business rule specifying an action for controlling the delivery of an e-mail message as a function of an attribute of the e-mail message, wherein at least one business rule defines an action for prohibiting or deferring delivery of an e-mail message based upon a role of a recipient user in the organizational hierarchy;

a rule engine coupled to receive an e-mail message from the receipt mechanism and to the data base to selectively apply the business rules to the e-mail message to determine from selected ones of the business rules a set of actions to be applied to the e-mail message; and

a distribution mechanism coupled to receive the set of actions from the rule engine and apply at least one action thereof to the e-mail message to control delivery of the e-mail message.

6. The post office of claim 1, further comprising:

a primary message store, coupled to the receipt engine, for receiving and non-persistently storing e-mail messages; and

a secondary message store, accessible to the distribution engine, for receiving therefrom, and persistently storing an e-mail message in response to the rule engine determining that the e-mail message satisfied a business rule requiring the e-mail message to be reviewed by a recipient other than a recipient specified by a sender of the e-mail message.

7. The post office of claim 1, further comprising:

a primary message store, coupled to the receipt engine, for receiving and non-persistently storing e-mail messages; and

a secondary message store, coupled to the distribution engine, for receiving therefrom, and persistently storing an e-mail message in response to the rule engine specifying the action that the e-mail message be reviewed by an administrator recipient prior to delivery to a specified recipient.

8. The post office of claim 1, wherein an e-mail message includes at least one specified recipient, and the distribution engine delivers the e-mail message to a non-specified recipient prior to delivery to a specified recipient.

9. The post office of claim 1, wherein:

the rule engine specifies an action of deleting the e-mail message; and

the distribution engine automatically deletes the e-mail message, without delivering the e-mail message to any of its specified recipients.

10. The post office of claim 1, wherein:

the rule engine specifies an action of copying the e-mail message; and

the distribution engine automatically copies the e-mail message, and delivers the copy of the e-mail message to a recipient other than a specified recipient.

11. The post office of claim 1, wherein:

the rule engine specifies an action of returning the e-mail message; and

the distribution engine automatically returns the e-mail message to a sender, and does not deliver.the e-mail message to any of its specified recipients.

12. The post office of claim 1, wherein:

the rule engine specifies an action of deferring the e-mail message; and

the distribution engine persistently stores the e-mail message in a storage area for subsequent review by an administrator, and does not deliver the e-mail message to any of the specified recipients.

13. The post office of claim 1, wherein each action has a priority, and the distribution engine executes a highest priority action for each e-mail message.

14. The post office of claim 13, wherein a highest priority is assigned to an action of gating an e-mail message to a recipient other than a specified recipient.

15. An e-mail system comprising:

a first post office operating on a first computer;

a second post office, operating on a second computer, the second post office comprising:

a receipt mechanism for receiving e-mail messages from a plurality of clients, each e-mail message having at least one specified recipient;

a database of business rules, each business rule specifying an action for controlling delivery of an e-mail message;

a rule engine coupled to receive an e-mail message from the receipt mechanism and selectively applying the business rules to the e-mail message to determine a set of actions to be applied to the e-mail message to control delivery of the e-mail message, the rule engine specifying for at least one e-mail message an action of deferring delivery of the message to its specified recipients by delivering the message to an administrator associated with the first post office;

a distribution mechanism coupled to receive the at least one action from the rule engine and which in response to the rule engine applying the action of deferring delivery of the e-mail message, the distribution mechanism automatically combines the e-mail message with a new distribution list specifying at least the first post office for receiving the e-mail message for review by the administrator associated with the first post office, and a rule history specifying at least one business rule determined to be applicable to the e-mail message by at least one rule engine, and automatically delivers the e-mail message to the first post office on the distribution list instead of a specified recipient of the e-mail message.

16. The e-mail system of claim 15, further comprising:

an administration application, communicatively coupled to the first post office, for reviewing e-mail messages delivered to the administrator.

17. A process for controlling the delivery of e-mail message in a business, comprising:

providing to a post office a set of business rules derived from business communication policies, each business rule defining an action applied to an e-mail message based on the attribute of the message;

receiving messages at the post office;

to at least one message received at the post office, applying the business rules to the attributes of the message to determine at least one action of deferring delivery to be applied to the message;

automatically combining the e-mail message with a new distribution list specifying at least one destination post office for receiving the e-mail message for review by an administrator associated with the destination post office and a rule history specifying at least one business rule determined to be applicable to the e-mail message; and

automatically delivering the e-mail message to a destination post office on the distribution list instead of a specified recipient of the e-mail message.

18. A computer implemented process for deferring the delivery of an e-mail message, comprising:

storing a database including an organizational hierarcy of a business, the hierarchy including a plurality of roles, each role associated with a user;

storing a database of business rules, each business rule specifying an action for controlling the delivery of an e-mail message as a function of an attribute of the e-mail message, wherein at least one business rule defines an action for deferring delivery of an e-mail message based upon a role of a recipient user in the organizational hierarchy;

receiving the e-mail message at a post office, the e-mail having at least one specified recipient, the at least one specified recipient having a role; and

applying the business rules to the e-mail message, including responsive to the role of the at least one specified recipient deferring the e-mail message by delivering the e-mail message to an administrator to review the e-mail message prior to any delivery of the e-mail message to the at least one specified recipient.

19. The process of claim 18, further comprising:

receiving a command from the administrator specifying an action to be applied to the e-mail message, the command selected from one of a group consisting of:

deleting the e-mail message instead of delivering it to its specified recipients;

copying the e-mail message and delivering the copy to a non-specified recipient; and

returning the e-mail message to its sender without delivering it to its specified recipients.

20. A computer implemented process for deferring the delivery of an e-mail message, comprising:

storing a database of business rules, each business rule specifying an action for controlling the delivery of an e-mail message as a function of an attribute of the e-mail message;

receiving the e-mail message at a post office, the e-mail message having at least one specified recipient;

deferring delivery of the e-mail message, by:

automatically combining the e-mail message with a new distribution list specifying at least one new destination post office for receiving the e-mail message for review by an administrator associated with the destination post office and a rule history specifying at least one business rule determined to be applicable to the e-mail message; and

automatically delivering the e-mail message to a first destination post office on the distribution list instead of a specified recipient of the e-mail message;

persistently storing the e-mail message at the first destination post office until the e-mail message is reviewed;

automatically reviewing the e-mail message after a specified time interval to determine an action to be applied to the e-mail message; and

automatically applying the action to the e-mail message.

21. A process for deferring the delivery of selected e-mail messages, comprising:

storing a database of business rules, each business rule specifying an action for controlling the delivery of an e-mail message as a function of an attribute of the e-mail message;

receiving a plurality of e-mail messages at a first post office, each e-mail message having at least one specified recipient;

selecting at least one e-mail message from the plurality of e-mail messages by applying at least one business rule to the e-mail message;

delivering each non-selected e-mail message to its specified recipients; and

deferring the selected e-mail message by:

automatically combining the selected e-mail message with a new distribution list specifying at least one

new destination post office for receiving the e-mail message for review by an administrator associated with the destination post office and a rule history specifying at least one business rule determined to be applicable to the e-mail message;

automatically delivering the selected e-mail message to a destination post office on the distribution list instead of a specified recipient of the e-mail message;

persistently storing the selected e-mail message in a storage area of the destination post office until the selected e-mail message is reviewed prior to any further delivery of the e-mail message to its specified recipients or to another destination post office on the distribution list.

**22**. A computer implemented process for reviewing an e-mail message, comprising:

receiving the e-mail message at a first post office, the e-mail message having at least one specified recipient;

deferring the e-mail message by:

automatically combining the selected e-mail message with a new distribution list specifying at least one second post office for receiving the e-mail message for review by an administrator associated with the second post office and a rule history specifying at least one business rule determined to be applicable to the e-mail message; and

automatically delivering the selected e-mail message to an administrator at the second post office on the distribution list instead of a specified recipient of the e-mail message;

persistently storing the e-mail message at the second post office until the e-mail message is reviewed;

automatically reviewing the e-mail message after a specified time interval to determine an action to be applied to the e-mail message; and

automatically applying the action to the e-mail message.

**23**. A computer implemented process for reviewing an e-mail message, each e-mail message having at least one specified recipient, the process comprising:

storing a database of business rules, each business rule specifying an action for controlling the delivery of an e-mail message as a function of an attribute of the e-mail message;

automatically combining e-mail message with a distribution list specifying at least one destination post office for receiving the e-mail message for review by an administrator associated with the destination post office, and a rule history specifying at least one business rule determined to be applicable to the e-mail message;

automatically delivering the selected e-mail message to an administrator at a destination post office on the distribution list instead of a specified recipient of the e-mail message;

persistently storing the e-mail message delivered to the administrator in a message store until the e-mail message is reviewed, each e-mail message in the message store having an expiration date;

receiving for at least one of the persistently stored e-mail message, a command from the administrator prior to the expiration date of the e-mail message, the command indicating an action to be applied to the e-mail message, and applying the action to the e-mail message; and

for each e-mail message for which a command is not received from the administrator prior to the expiration date of the e-mail message, automatically determining an action to be applied to the message by applying at least one business rule to the e-mail message, and applying the action to the e-mail message.

**24**. A post office for receiving and redistributing data objects on a computer network, the post office comprising:

a receipt mechanism that receives a data object from a sender, the data object having at least one specified recipient;

a database of business rules, each business rule specifying an action for controlling the delivery of a data object as a function of an attribute of the data object;

a rule engine coupled to receive a data object from the receipt mechanism and coupled to the database to selectively apply the business rules to the data object to determine from selected ones of the business rules a set of actions to be applied to the data object; and

a distribution mechanism coupled to receive the set of actions from the rule engine and apply at least one action thereof to the data object to control delivery of the data object and which in response to the rule engine applying an action of deferring delivery of the data object, the distribution engine automatically combines the data object with a new distribution list specifying at least one new destination post office for receiving the data object for review by an administrator associated with the destination post office and a rule history specifying at least one business rule determined to be applicable to the data object by at least one rule engine, and automatically delivers the data object to a first destination post office on the distribution list instead of a specified recipient of the data object.

**25**. A computer implemented process for deferring the delivery of a data object, comprising:

storing a database including an organizational hierarchy of a business, the hierarchy including a plurality of roles, each role associated with a user;

storing a database of business rules, each business rule specifying an action for controlling the delivery of a data object as a function of an attribute of the data object, wherein at least one business rule defines an action for deferring delivery of a data object based upon a role of a recipient user in the organizational hierarchy;

receiving the data object at a post office, the data object having at least one specified recipient, the at least one specified recipient having a role;

applying the business rules to the data object, including responsive to the role of the recipient deferring delivery of the data object by delivering the data object to a recipient other than a specified recipient;

persistently storing the data object until the data object is reviewed;

automatically reviewing the data object after a specified time interval to determine an action to be applied to the data object; and

automatically applying the action to the data object.

**26**. A process for deferring the delivery of selected data objects, comprising:

storing a database of business rules, each business rule specifying an action for controlling the delivery of a data object as a function of an attribute of the data object;

receiving a plurality of data objects at a first post office, each data object having at least one specified recipient;

selecting at least one data object from the plurality of data objects by applying at least one business rule to the data object;

delivering each non-selected data object to its specified recipients;

deferring the selected data object by:

automatically combining the selected data object with a new distribution list specifying at least one new destination post office for receiving the data object for review by an administrator associated with the destination post office; and a rule history specifying at least one business rule determined to be applicable to the data object;

automatically delivering the selected data object to a destination post office on the distribution list instead of a specified recipient of the data object; and

persistently storing the data object in a storage area until the data object is reviewed prior to any further delivery of the data object to its specified recipients.

* * * * *

US006226666B1

(12) **United States Patent**
**Chang et al.**

(10) **Patent No.: US 6,226,666 B1**
(45) **Date of Patent: *May 1, 2001**

(54) **AGENT-BASED MANAGEMENT SYSTEM HAVING AN OPEN LAYERED ARCHITECTURE FOR SYNCHRONOUS AND/OR ASYNCHRONOUS MESSAGING HANDLING**

(75) Inventors: **Daniel T. Chang**; **Neelakantan Sundaresan**, both of San Jose, CA (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

( * ) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/884,457**

(22) Filed: **Jun. 27, 1997**

(51) **Int. Cl.**[7] ..... **G06F 15/16**; G06F 15/167

(52) **U.S. Cl.** ..... **709/202**; 709/201; 709/206; 709/315; 379/100.08; 379/93.24

(58) **Field of Search** ..... 709/200–205, 709/206–209, 210–219, 220–229, 230–239, 315; 370/395, 402, 455, 241, 471, 392; 379/89, 94, 100.08; 283/61; 713/201; 705/400; 232/17; 717/1; 712/17

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | | |
|---|---|---|---|---|
| 5,327,558 | * | 7/1994 | Burke et al. | 710/8 |
| 5,627,764 | * | 5/1997 | Schutzman et al. | 395/200.37 |
| 5,634,127 | * | 5/1997 | Cloud et al. | 395/680 |
| 5,655,081 | * | 8/1997 | Bonnell et al. | 709/202 |

(List continued on next page.)

OTHER PUBLICATIONS

First International Workshop on Mobile Agents 97 (MA'97) URL=http:www.informatik.uni—stuttgart.de/ipvr/vs/ws/ma97/ma97.html.

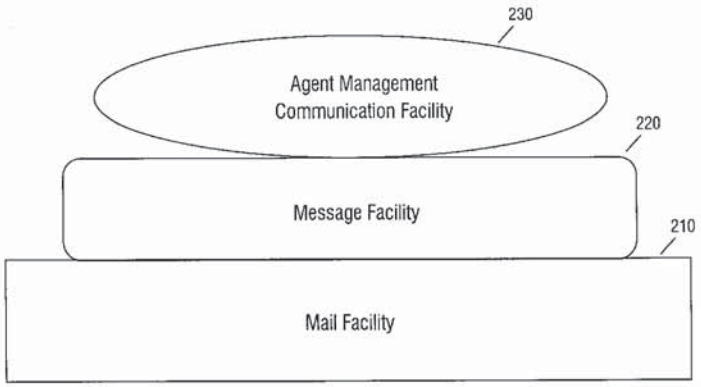
(List continued on next page.)

*Primary Examiner*—Zarni Maung
*Assistant Examiner*—Beatriz Prieto
(74) *Attorney, Agent, or Firm*—Prentiss Wayne Johnson

(57) **ABSTRACT**

A communication infrastructure providing communication between agents, between agents and agent-hosting servers, and between agent-hosting servers. The communication infrastructure consists of three layers (from bottom to top): Mail Facility Layer, Message Facility Layer, and Agent Management Communication Facility Layer. The Mail Facility Layer is the lowest layer providing a general, semantics-free mail paradigm for asynchronous communication between distributed objects, whether they are local or remote to each other. The Mail Facility Layer provides a level of abstraction in terms of mail, virtual mailbox, post office, and mail queue, and hides the details of implementation and actual transport. It is designed to provide location transparency and to be implementable using various transport protocols. The next Message Facility Layer provides a typed messaging paradigm for asynchronous and synchronous message passing between distributed objects. The Message Facility Layer uses the Mail Facility Layer for sending messages and for getting responses to requests sent. It allows for the association of typed message handlers with typed messages such that the format and semantics of messages are encapsulated through their types, are extensible, and can be processed by the associated message handlers. The Agent Management Communication Facility Layer is the highest layer providing the services for inter-agent communication between agents, agent-agent-server communication between an agent and an agent server, and inter-agent-server communication between agent servers for managing agents such as locating an agent, dispatching an agent, retrieving an agent, etc. The key abstractions provided in this layer include agent manager, agent, and agent identifier. It uses the Message

**12 Claims, 16 Drawing Sheets**



U.S. PATENT DOCUMENTS

| | | | | |
|---|---|---|---|---|
| 5,680,551 | * | 10/1997 | Martino | 395/200.56 |
| 5,689,550 | * | 11/1997 | Garson et al. | 379/88.18 |
| 5,715,474 | * | 2/1998 | Burke et al. | 710/6 |
| 5,757,669 | * | 5/1998 | Christie et al. | 395/200.35 |

OTHER PUBLICATIONS

"Mobile Agent Computing", A White Paper, Mitsubishi Electric ITA, Feb. 28, 1997, http://www.meitca.com/HSL/Projects/Concordia.

Voyager, ObjectSpace. 1997.http://www.objectspace.com/Voyager/voyager1.html.

"Voyager Core Package Technical Overview", ObjectSpace, Mar., 1997.

"JavaSpace Specification", Revision 0.3, Sun Microsystems, Inc. Mar. 1997.

N. Carriero and D. Gelernter, "Linda in Context", Communications of the ACM., 32(4), pp. 444–458, Apr., 1989.

J. Waldo, G. Wyant, A. Wollrath and S. Kendall, "A Note on Distributed Computing", Sun Microsystems Laboratories technical report SMLI TR–94–29, Nov. 1994.

D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, "Itinerant Agents for Mobile Computing", IBM Research Report, RC 20010, IBM Research Division, Mar. 1995.

C. Harrison, D. Chess, and A. Kershenbaum, "Mobile Agents: Are they a good idea?", IBM Research Report, IBM Research Division, Mar. 1995.

Aglets Workbench, IBM, URL=http://www.trl.ibm.co.jp/aglets.

D. T. Chang and D. B. Lange, "Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW", in Proceedings of the OOPSLA96 Workshop: Toward the Integration of WWW and Distributed Object Technology, Oct., 1996.

Concordia, Mitsubishi Electric ITA, URL=http://www.meitca.com/HSL/Projects/Concordia.

"Concordia: An Infrastructure for Collaborating Mobile Agents" Mitsubishi Electric ITA, First International Workshop on Mobile Agents 97 MA'97), Apr., 1997.

"The Common Object Request Broker: Architecture and Specification", Revision 2.0, OMG, Jul. 1995.

InfoSleuth Project, URL=http://www.mcc.com/projects/infosleuth.

The Java Development Kit (JDK), URL http://java.sun.com/products/jdk.

JKQML, IBM, URL=http://www.alphaworks.ibm.com/formula/jkqml/.

Y. Labrou, "Semantics for an Agent Communication Language", Ph.D. Thesis, CSEE Department, University of Maryland, Baltimore Maryland 21228–5398, Sep., 1996. URL=http://www.cs.umbc.edu/kqml.

Visual Warehouse, IBM, URL=http://www.software.ibm.com/data/warehouse/vw.

Wise (Wonderful Indexing and Searching Environment), IBM, URL=http://wise.watson.ibm.com.

Ref: Newton's Telecom Dictionary, Newton, H., Flatiron Publishing, 14th Expanded and Updated Ed., Mar., 1998, see encapsulation, Mar. 1998.*

A Gateway between MHS (X.400) and SMTP, Tang, D., Anzenberger, M., Markovitz P., Wallace M., National Bureau of Standards Institute for Computer Science and Technology, Mar. 1998.*

* cited by examiner

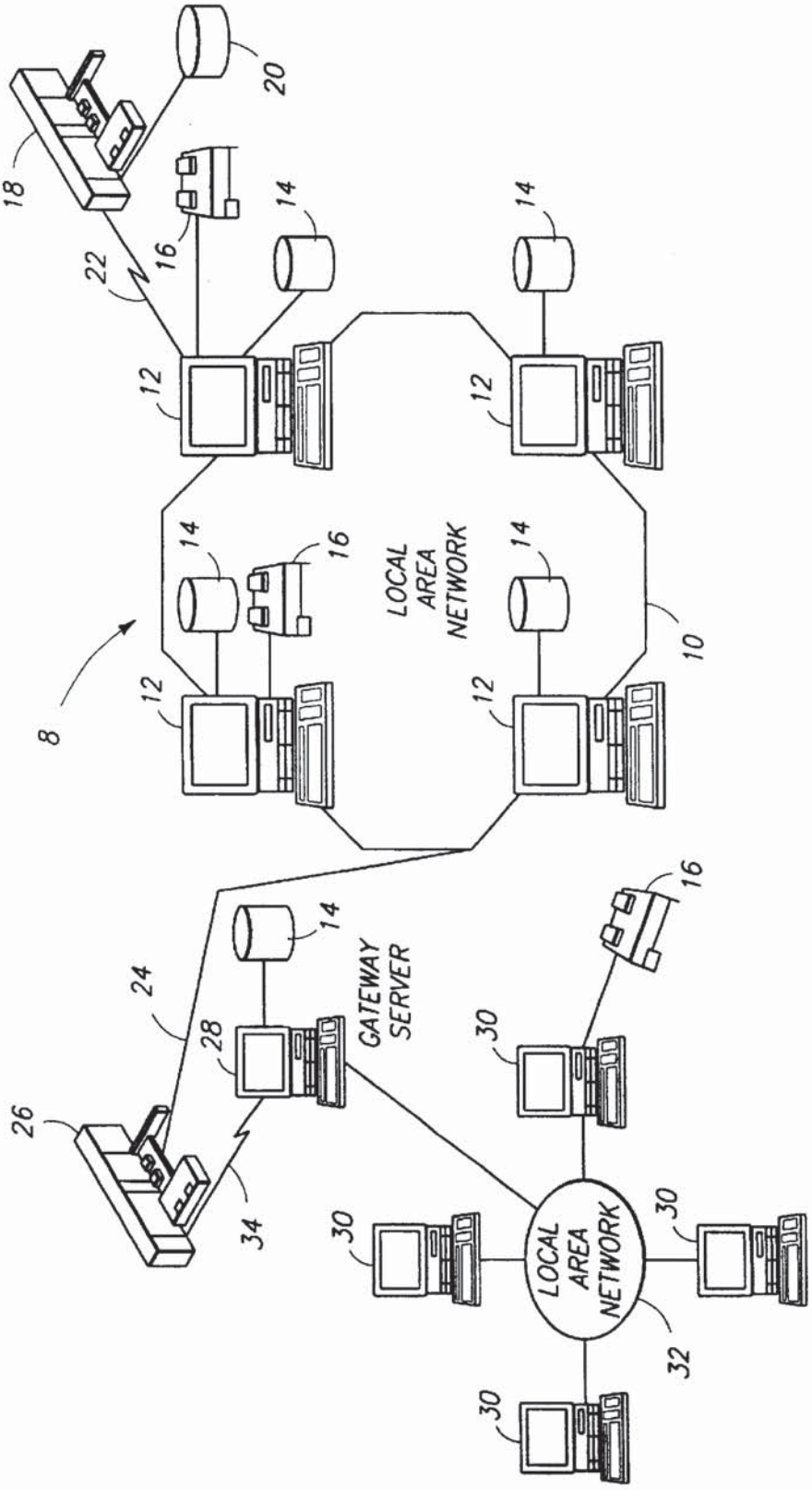

20
18
14
16
22
14
12
12
14
16
LOCAL AREA NETWORK
14
8
12
12
10
16
14
GATEWAY SERVER
24
28
30
26
34
30
LOCAL AREA NETWORK
30
32
30

Fig. 1

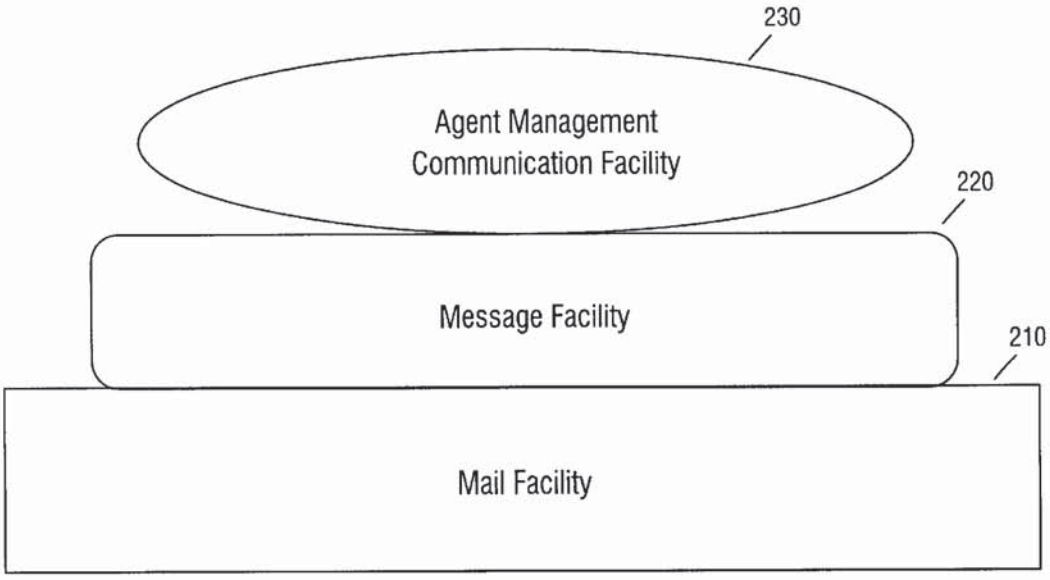

230
Agent Management
Communication Facility
220
Message Facility
210
Mail Facility

*Fig. 2*

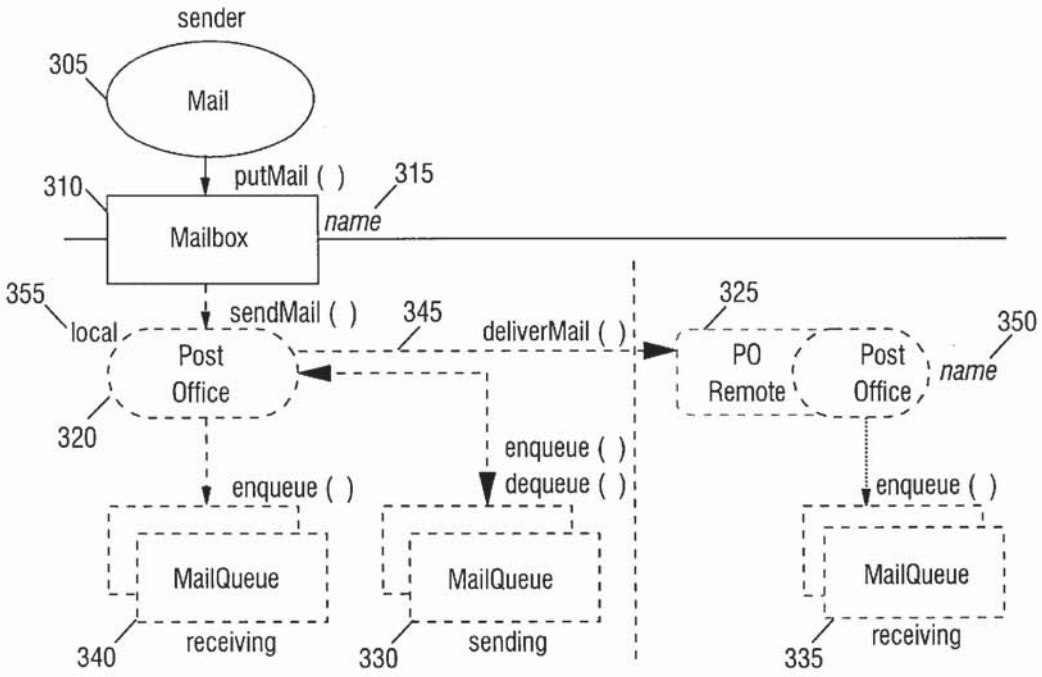

sender
305
Mail
putMail ( )
315
310
Mailbox
name
355
local
sendMail ( )
345
deliverMail ( )
325
350
PO
Remote
Post
Office
name
Post
Office
320
enqueue ( )
dequeue ( )
enqueue ( )
enqueue ( )
MailQueue
receiving
340
MailQueue
sending
330
MailQueue
receiving
335

*Fig. 3*

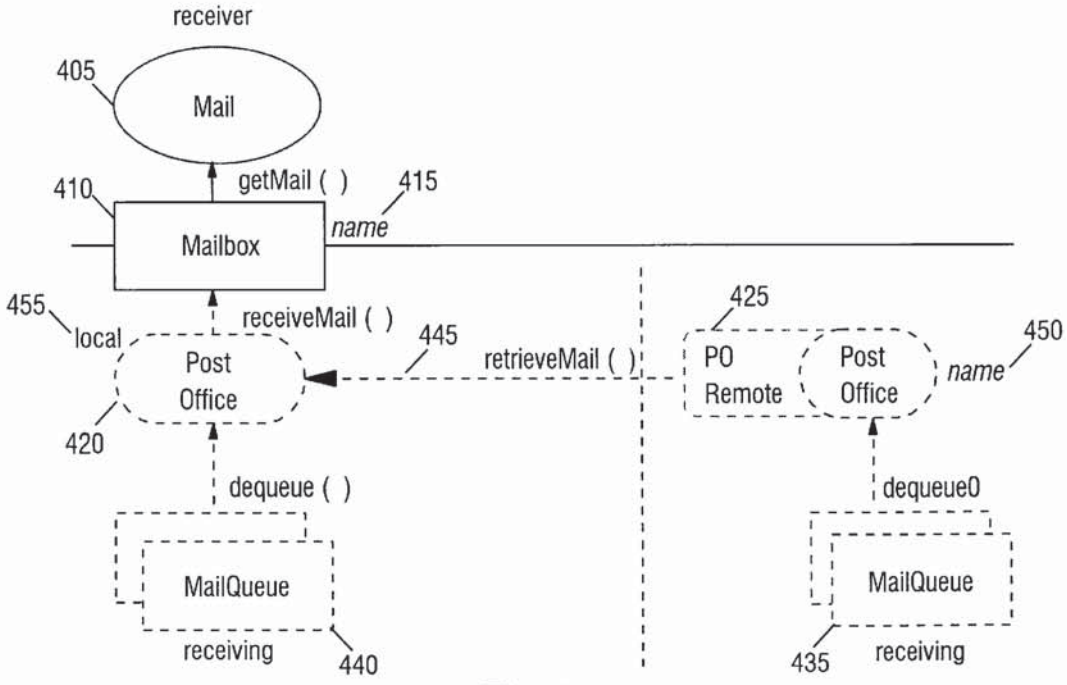

receiver
405
Mail
getMail ( )
415
410
Mailbox
name
455
local
receiveMail ( )
445
retrieveMail ( )
425
450
PO
Remote
Post
Office
name
Post
Office
420
dequeue ( )
dequeue0
MailQueue
receiving
440
MailQueue
receiving
435

*Fig. 4*

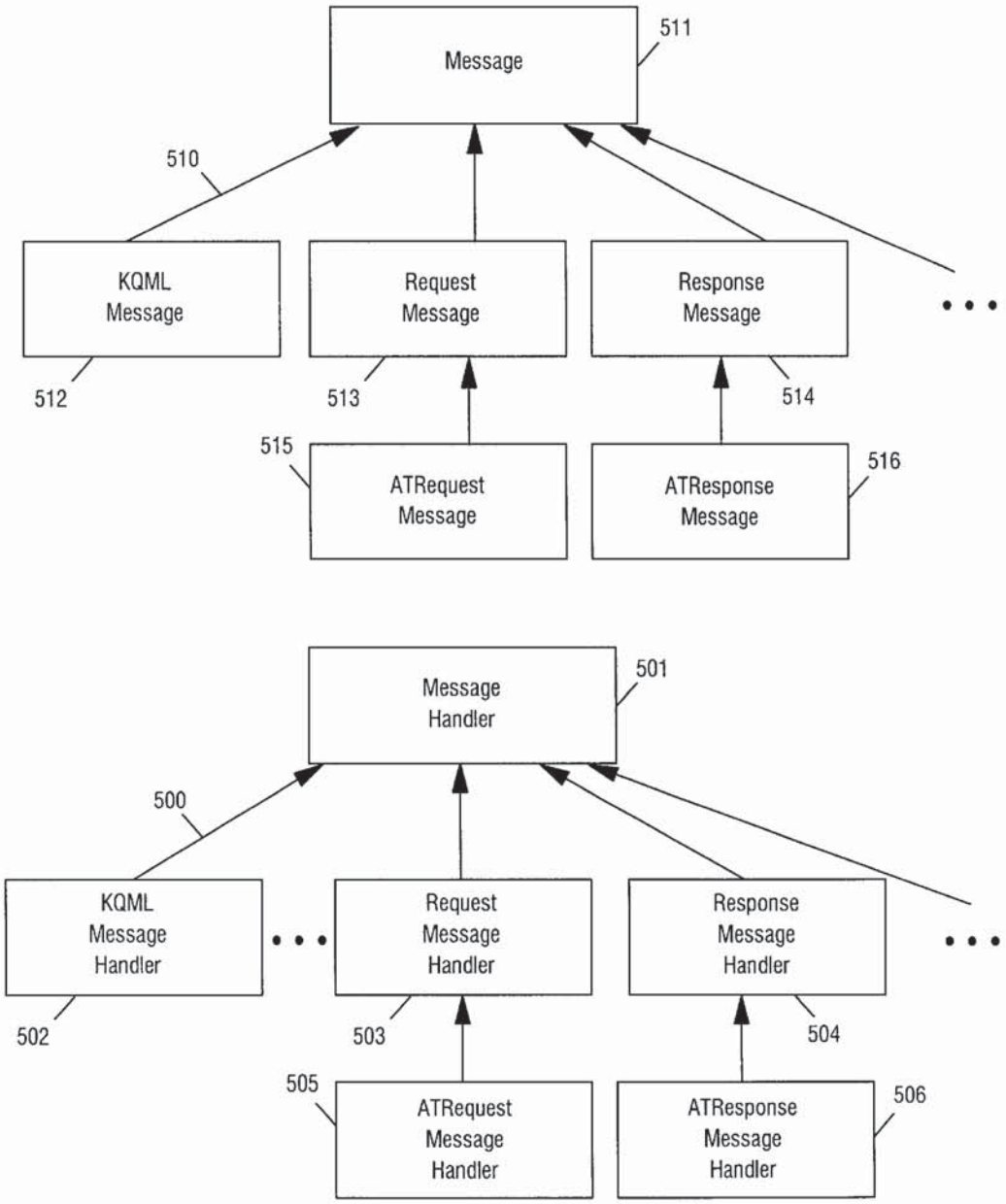

Message
511
510
KQML Message
Request Message
Response Message
512
513
514
515
ATRequest Message
ATResponse Message
516
Message Handler
501
500
KQML Message Handler
Request Message Handler
Response Message Handler
502
503
504
505
ATRequest Message Handler
ATResponse Message Handler
506

*Fig. 5*

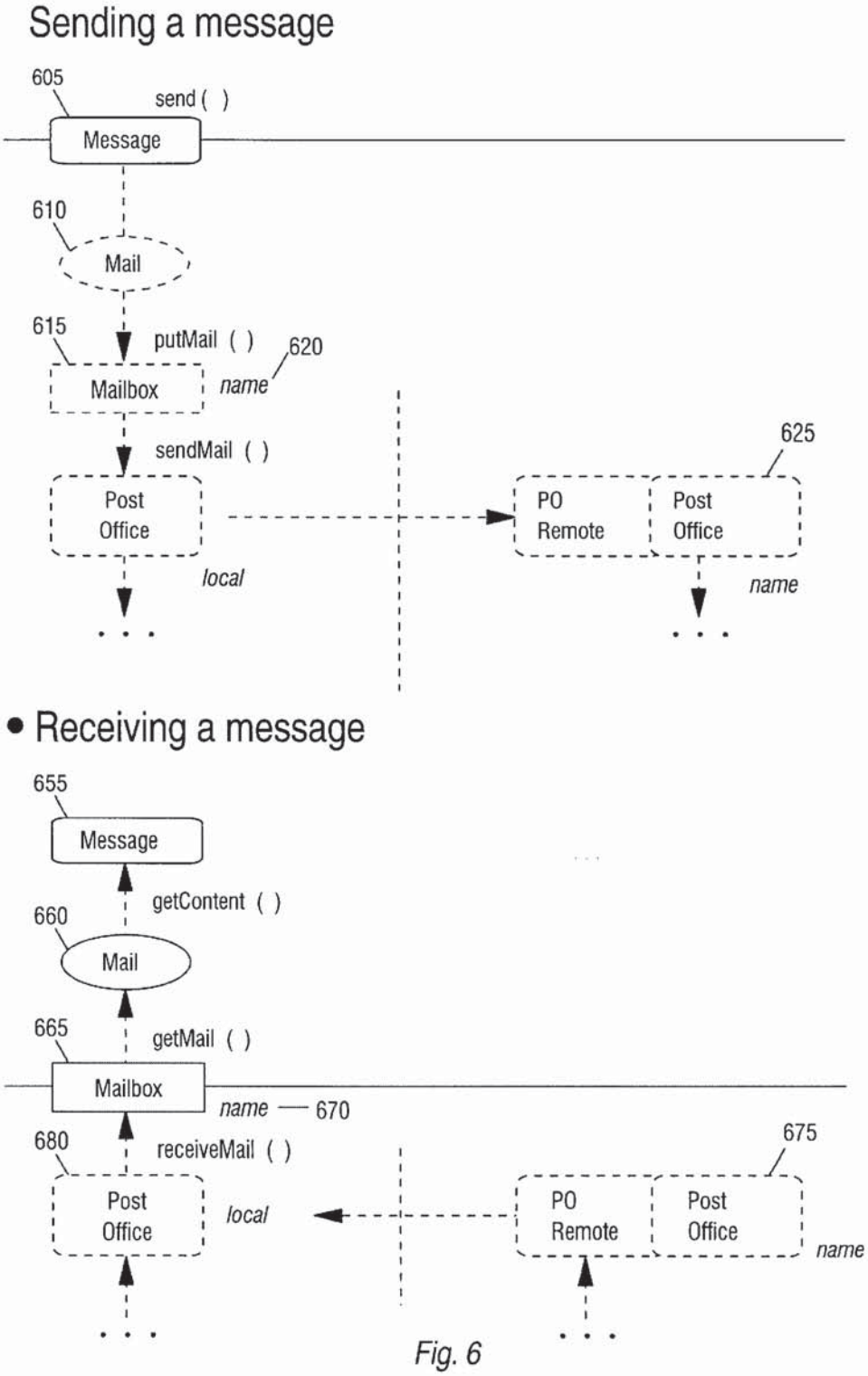

Sending a message
605
send ( )
Message
610
Mail
615
putMail ( )
620
Mailbox
name
sendMail ( )
625
Post Office
local
PO Remote
Post Office
name
• Receiving a message
655
Message
getContent ( )
660
Mail
665
getMail ( )
Mailbox
name
670
680
receiveMail ( )
675
Post Office
local
PO Remote
Post Office
name

Fig. 6

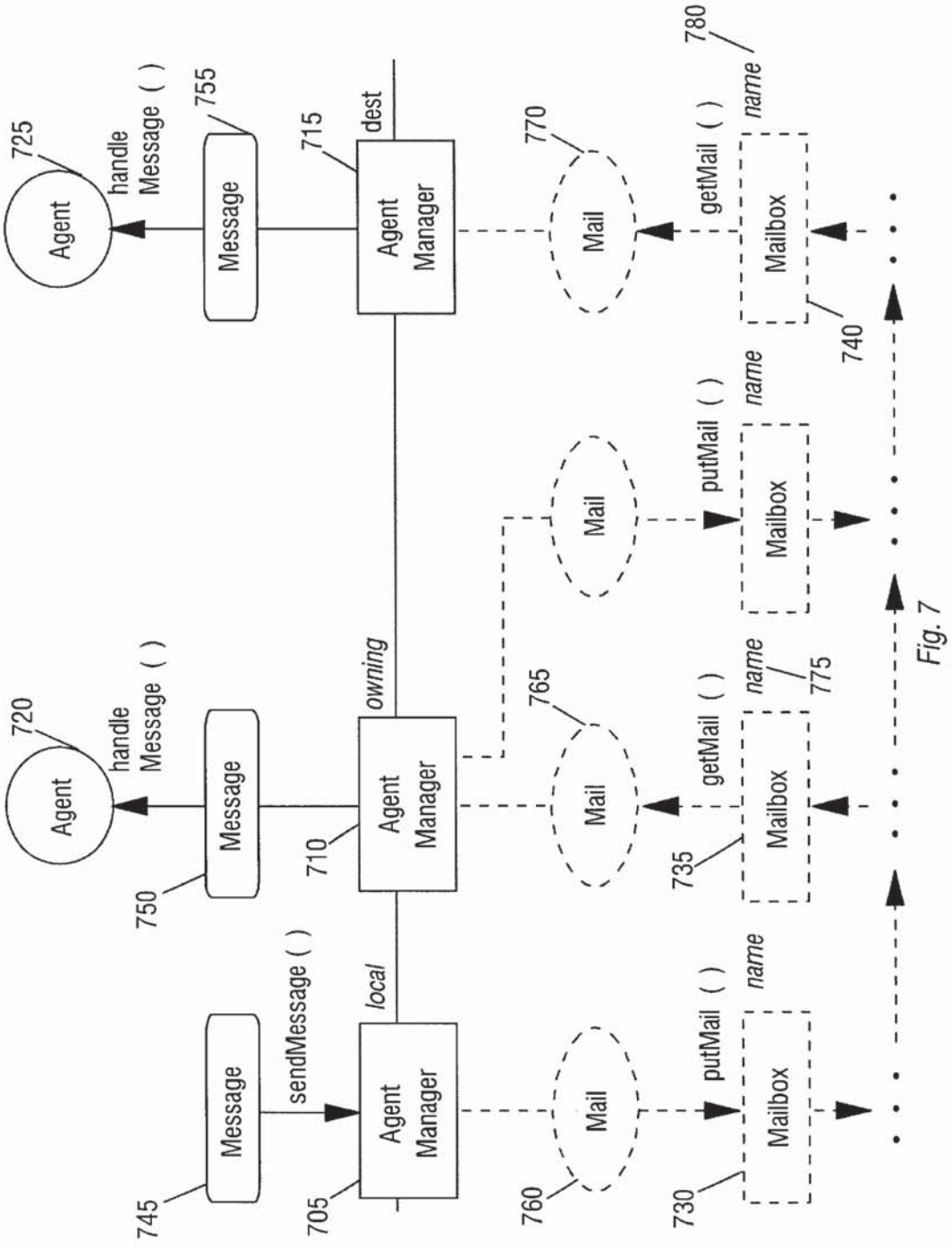

Agent
handle Message ( )
Message
Agent Manager
dest
owning
local
sendMessage ( )
Mail
getMail ( )
putMail ( )
Mailbox
name
725
755
715
770
780
720
750
710
765
735
775
740
745
705
760
730

Fig. 7

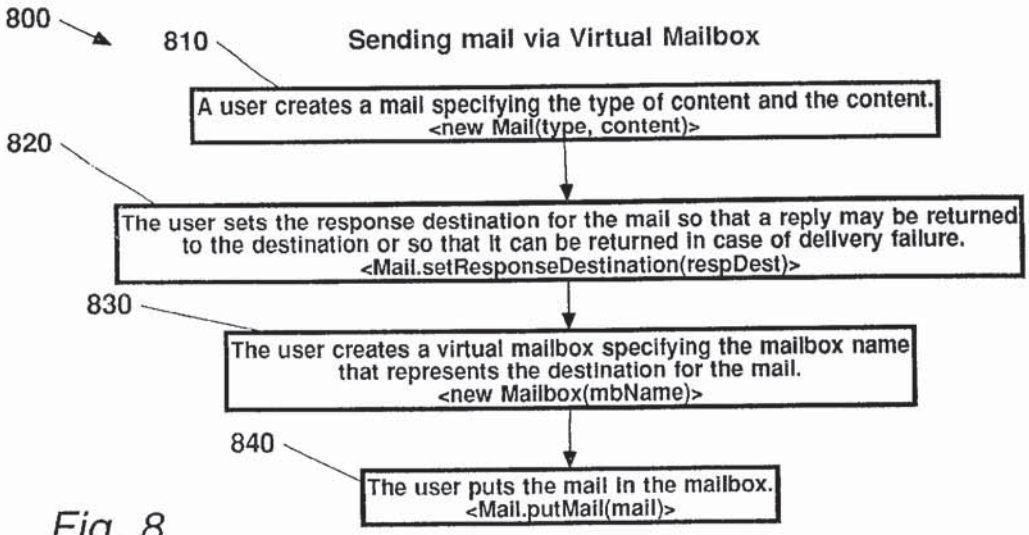

800
810
Sending mail via Virtual Mailbox
A user creates a mail specifying the type of content and the content.
<new Mail(type, content)>
820
The user sets the response destination for the mail so that a reply may be returned to the destination or so that it can be returned in case of delivery failure.
<Mail.setResponseDestination(respDest)>
830
The user creates a virtual mailbox specifying the mailbox name that represents the destination for the mail.
<new Mailbox(mbName)>
840
The user puts the mail in the mailbox.
<Mail.putMail(mail)>

Fig. 8

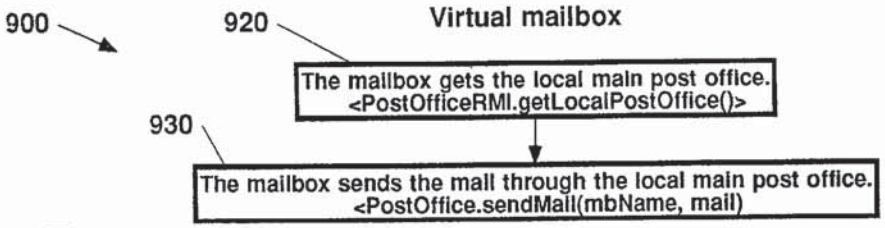

900
920
Virtual mailbox
The mailbox gets the local main post office.
<PostOfficeRMI.getLocalPostOffice()>
930
The mailbox sends the mail through the local main post office.
<PostOffice.sendMail(mbName, mail)

Fig. 9

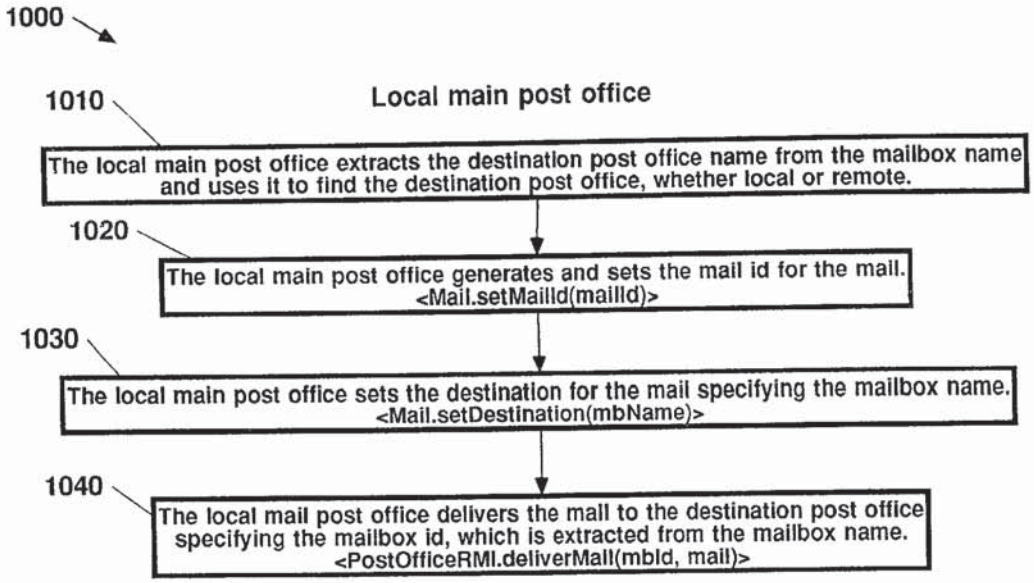

1000
1010
Local main post office
The local main post office extracts the destination post office name from the mailbox name and uses it to find the destination post office, whether local or remote.
1020
The local main post office generates and sets the mail id for the mail.
<Mail.setMailId(mailId)>
1030
The local main post office sets the destination for the mail specifying the mailbox name.
<Mail.setDestination(mbName)>
1040
The local mail post office delivers the mail to the destination post office specifying the mailbox id, which is extracted from the mailbox name.
<PostOfficeRMI.deliverMail(mbId, mail)>

*Fig. 10*

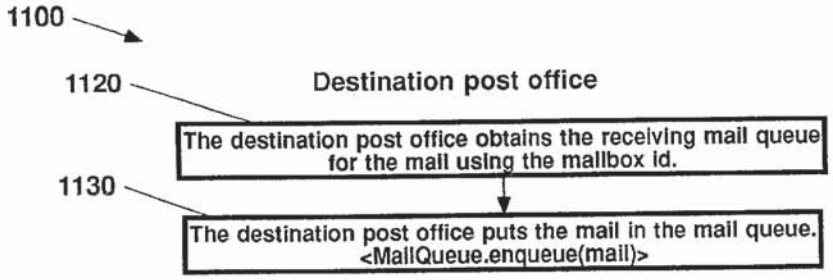

1100
1120
Destination post office
The destination post office obtains the receiving mail queue for the mail using the mailbox id.
1130
The destination post office puts the mail in the mail queue.
<MailQueue.enqueue(mail)>

*Fig. 11*

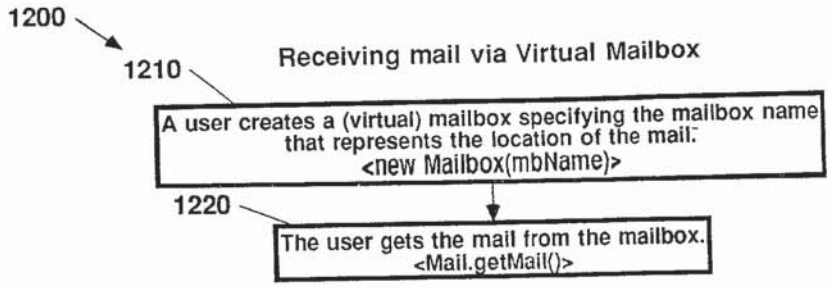



Fig. 12

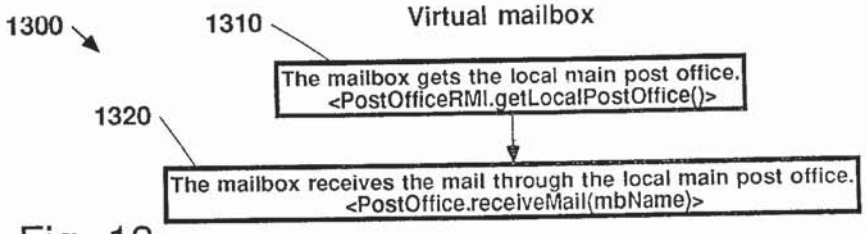



Fig. 13

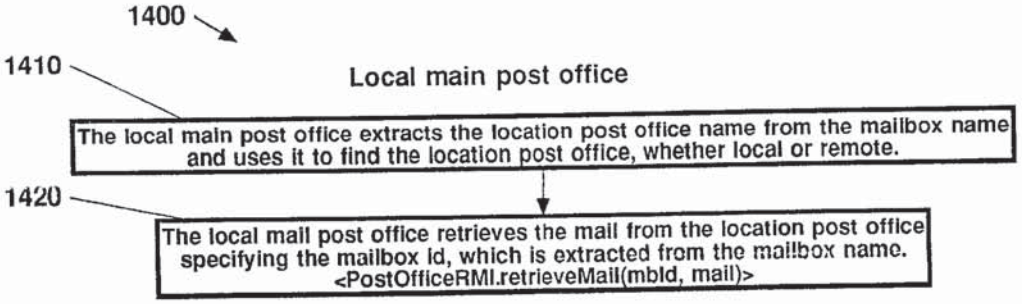



Fig. 14

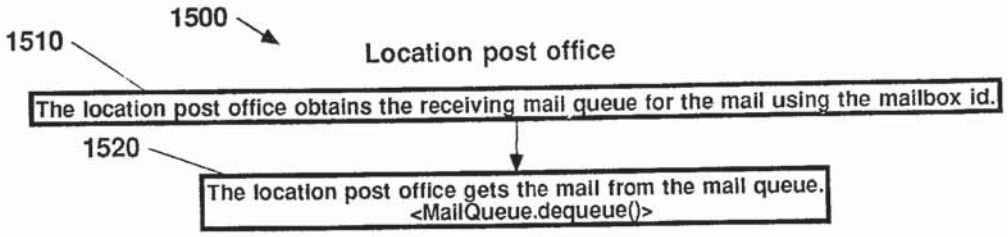



Fig. 15

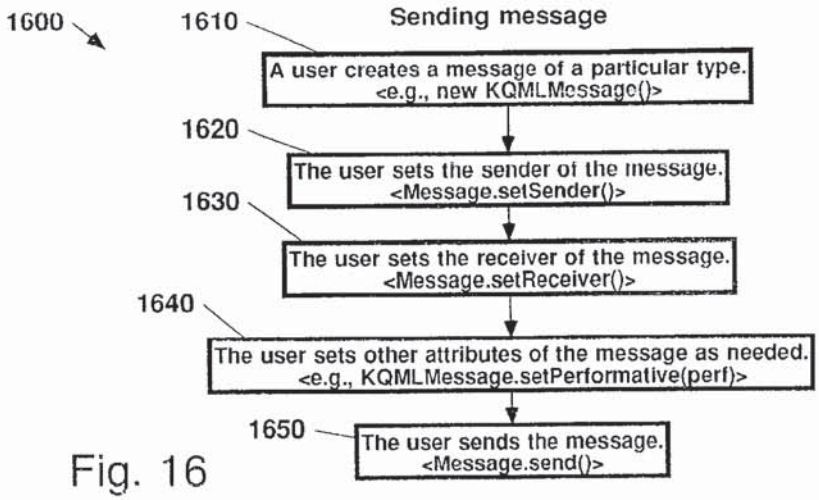



Fig. 16

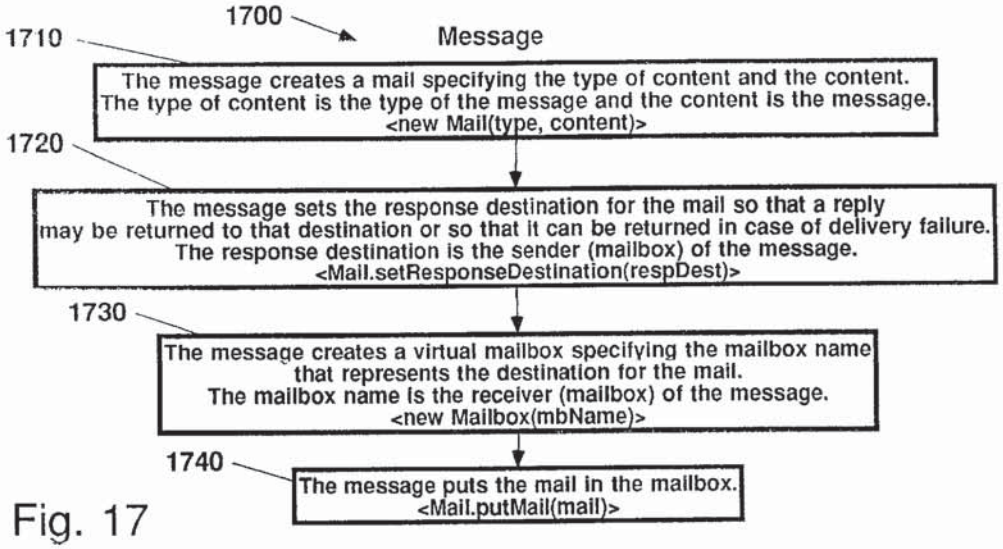



Fig. 17

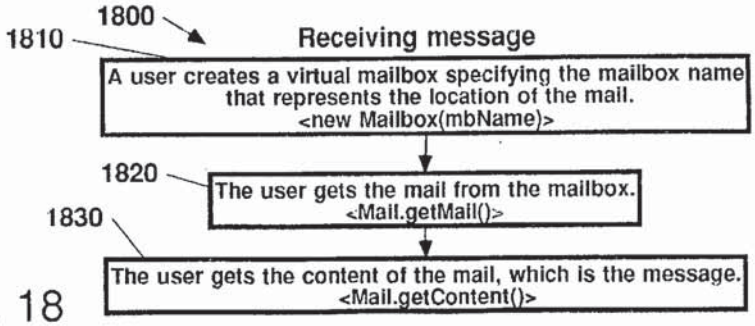



Fig. 18

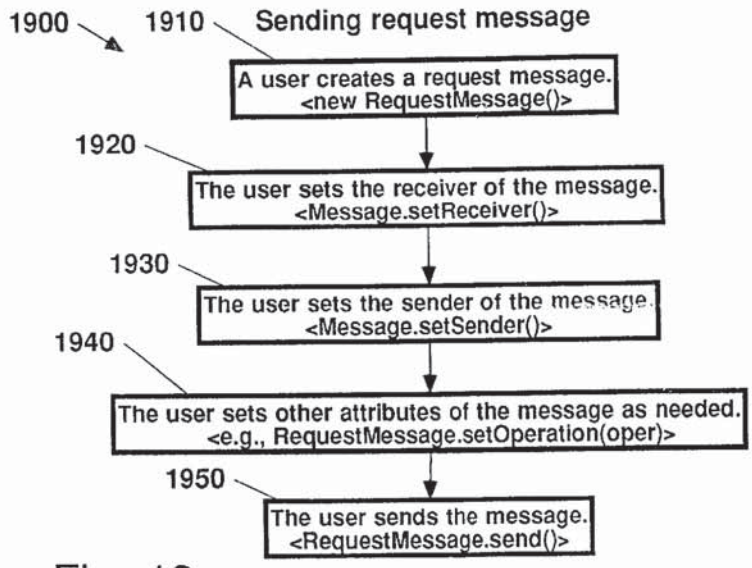

1900
1910
Sending request message
A user creates a request message.
<new RequestMessage()>
1920
The user sets the receiver of the message.
<Message.setReceiver()>
1930
The user sets the sender of the message.
<Message.setSender()>
1940
The user sets other attributes of the message as needed.
<e.g., RequestMessage.setOperation(oper)>
1950
The user sends the message.
<RequestMessage.send()>

Fig. 19

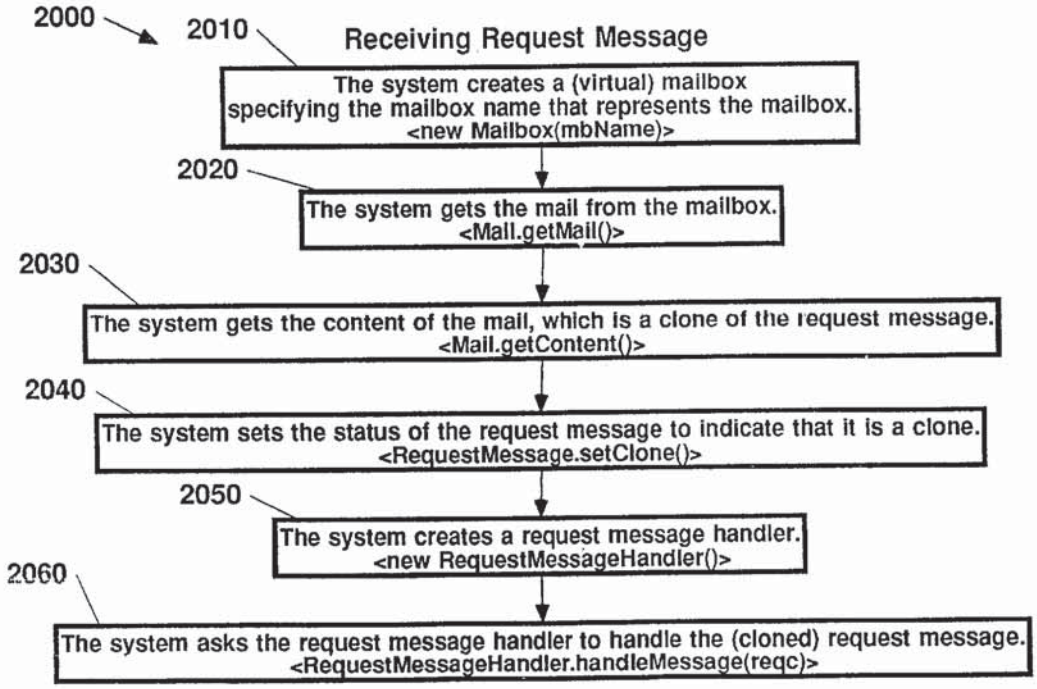

2000
2010
Receiving Request Message
The system creates a (virtual) mailbox
specifying the mailbox name that represents the mailbox.
<new Mailbox(mbName)>
2020
The system gets the mail from the mailbox.
<Mail.getMail()>
2030
The system gets the content of the mail, which is a clone of the request message.
<Mail.getContent()>
2040
The system sets the status of the request message to indicate that it is a clone.
<RequestMessage.setClone()>
2050
The system creates a request message handler.
<new RequestMessageHandler()>
2060
The system asks the request message handler to handle the (cloned) request message.
<RequestMessageHandler.handleMessage(reqc)>

Fig. 20

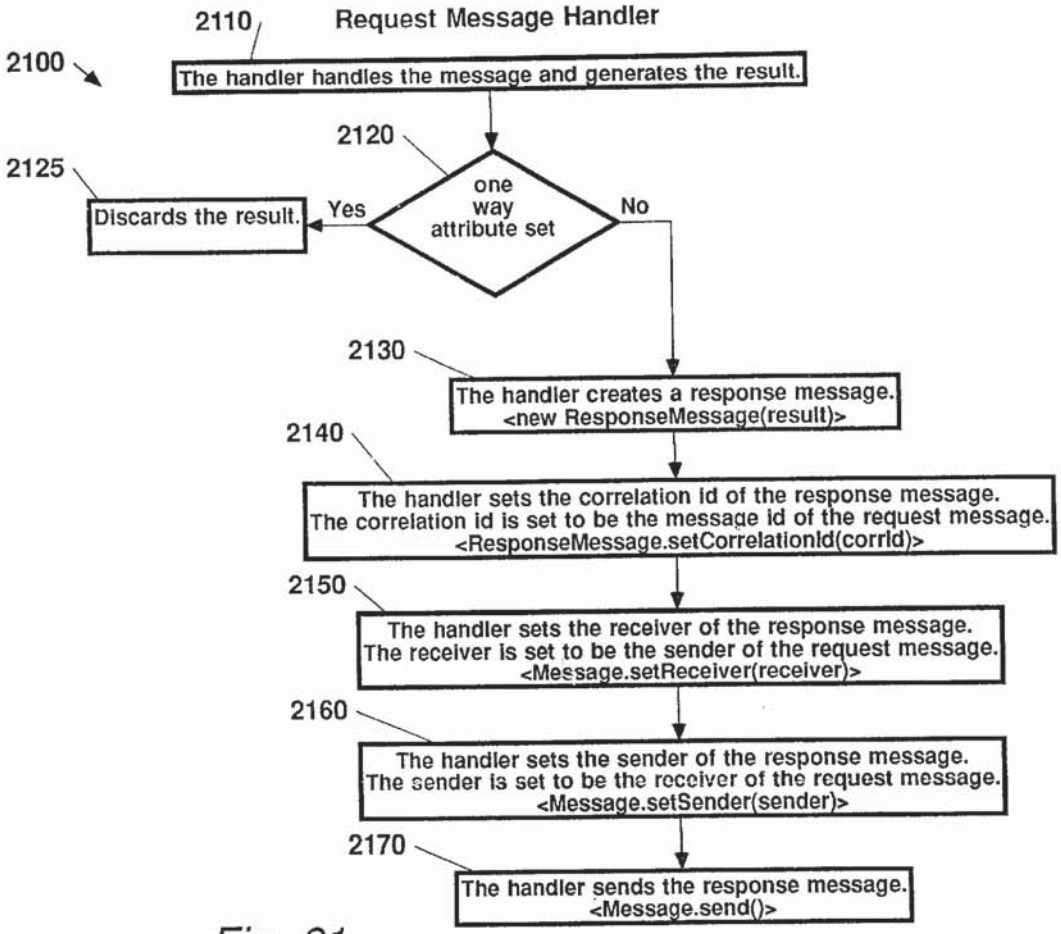

2110
Request Message Handler
2100
The handler handles the message and generates the result.
2120
2125
one way attribute set
Yes
No
Discards the result.
2130
The handler creates a response message.
<new ResponseMessage(result)>
2140
The handler sets the correlation id of the response message.
The correlation id is set to be the message id of the request message.
<ResponseMessage.setCorrelationId(corrId)>
2150
The handler sets the receiver of the response message.
The receiver is set to be the sender of the request message.
<Message.setReceiver(receiver)>
2160
The handler sets the sender of the response message.
The sender is set to be the receiver of the request message.
<Message.setSender(sender)>
2170
The handler sends the response message.
<Message.send()>

*Fig. 21*

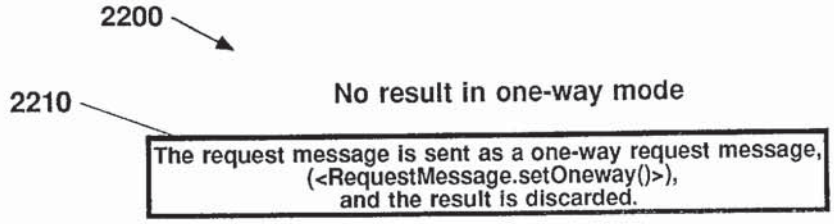



Fig. 22

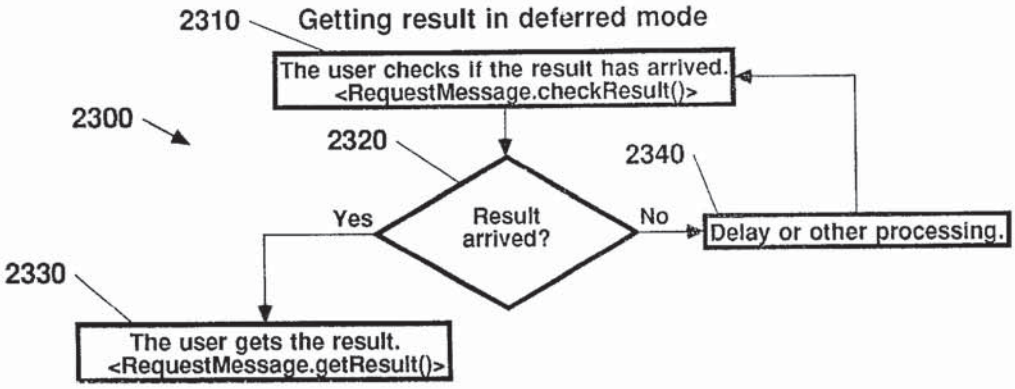



Fig. 23

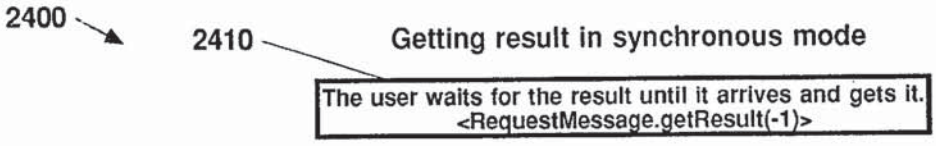



Fig. 24

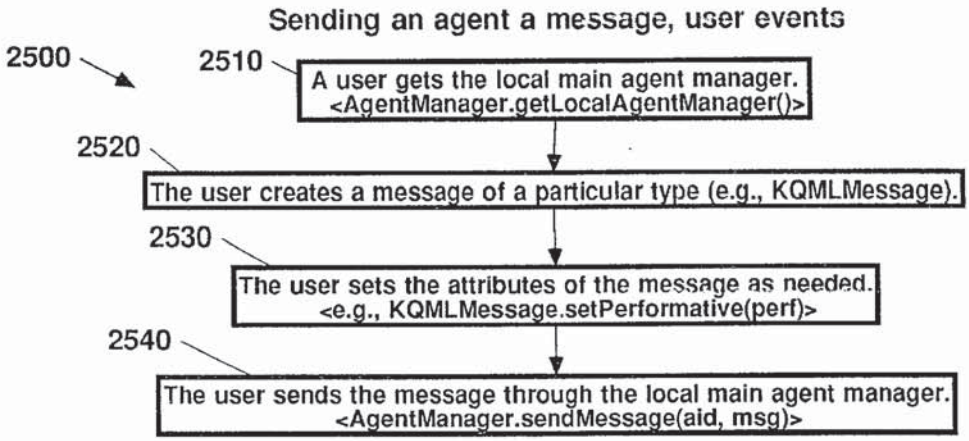

Sending an agent a message, user events
2500
2510
A user gets the local main agent manager.
<AgentManager.getLocalAgentManager()>
2520
The user creates a message of a particular type (e.g., KQMLMessage).
2530
The user sets the attributes of the message as needed.
<e.g., KQMLMessage.setPerformative(perf)>
2540
The user sends the message through the local main agent manager.
<AgentManager.sendMessage(aid, msg)>

Fig. 25

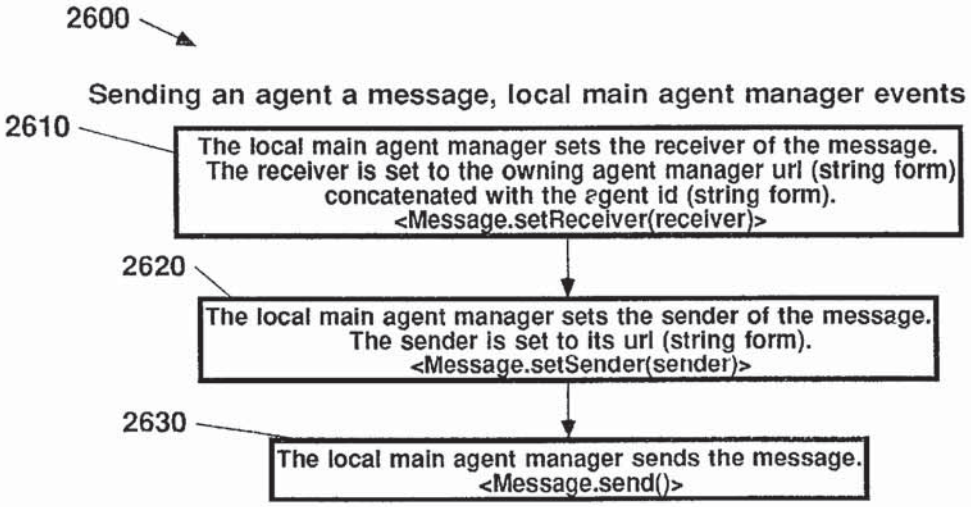

2600
Sending an agent a message, local main agent manager events
2610
The local main agent manager sets the receiver of the message.
The receiver is set to the owning agent manager url (string form)
concatenated with the agent id (string form).
<Message.setReceiver(receiver)>
2620
The local main agent manager sets the sender of the message.
The sender is set to its url (string form).
<Message.setSender(sender)>
2630
The local main agent manager sends the message.
<Message.send()>

Fig. 26

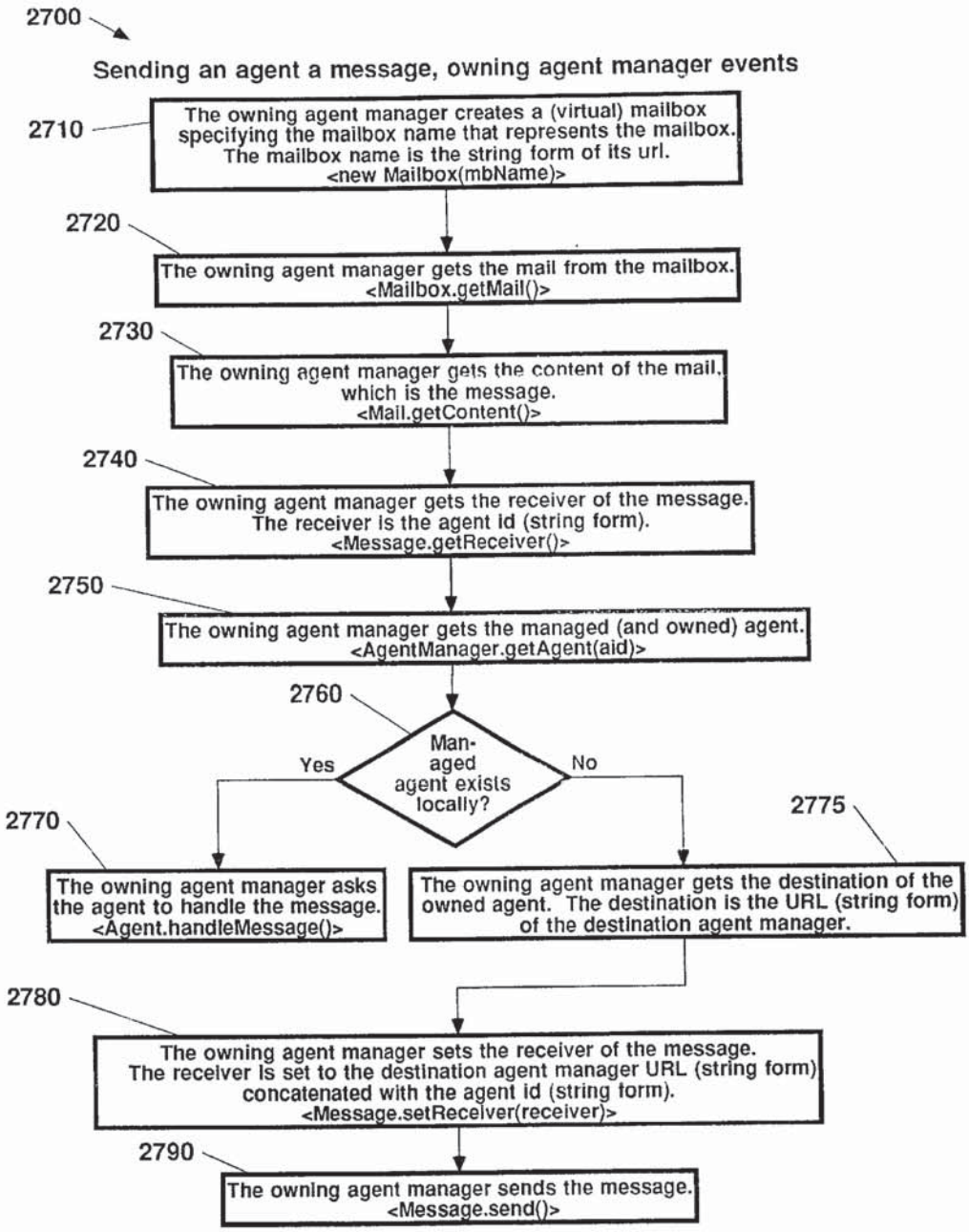

2700
Sending an agent a message, owning agent manager events
2710
The owning agent manager creates a (virtual) mailbox specifying the mailbox name that represents the mailbox. The mailbox name is the string form of its url.
<new Mailbox(mbName)>
2720
The owning agent manager gets the mail from the mailbox.
<Mailbox.getMail()>
2730
The owning agent manager gets the content of the mail, which is the message.
<Mail.getContent()>
2740
The owning agent manager gets the receiver of the message. The receiver is the agent id (string form).
<Message.getReceiver()>
2750
The owning agent manager gets the managed (and owned) agent.
<AgentManager.getAgent(aid)>
2760
Man-aged agent exists locally?
Yes
No
2770
The owning agent manager asks the agent to handle the message.
<Agent.handleMessage()>
2775
The owning agent manager gets the destination of the owned agent. The destination is the URL (string form) of the destination agent manager.
2780
The owning agent manager sets the receiver of the message. The receiver is set to the destination agent manager URL (string form) concatenated with the agent id (string form).
<Message.setReceiver(receiver)>
2790
The owning agent manager sends the message.
<Message.send()>

Fig. 27

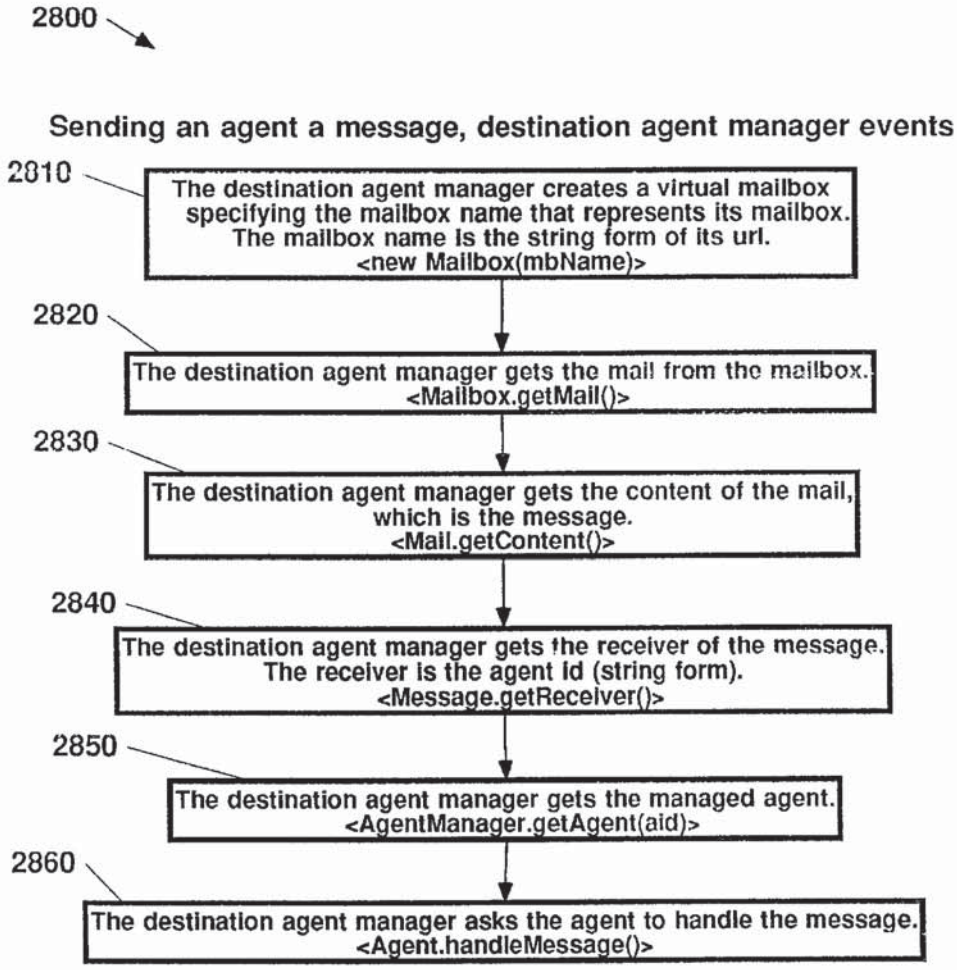

2800
Sending an agent a message, destination agent manager events
2810
The destination agent manager creates a virtual mailbox specifying the mailbox name that represents its mailbox. The mailbox name is the string form of its url.
<new Mailbox(mbName)>
2820
The destination agent manager gets the mail from the mailbox.
<Mailbox.getMail()>
2830
The destination agent manager gets the content of the mail, which is the message.
<Mail.getContent()>
2840
The destination agent manager gets the receiver of the message. The receiver is the agent id (string form).
<Message.getReceiver()>
2850
The destination agent manager gets the managed agent.
<AgentManager.getAgent(aid)>
2860
The destination agent manager asks the agent to handle the message.
<Agent.handleMessage()>

*Fig. 28*

# AGENT-BASED MANAGEMENT SYSTEM HAVING AN OPEN LAYERED ARCHITECTURE FOR SYNCHRONOUS AND/OR ASYNCHRONOUS MESSAGING HANDLING



## BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates in general to agent computer programs, including mobile agents, intelligent agents, collaborating agents, internet agents, and task-specific agents, and more particularly to a communication infrastructure for communication between agents, between agents and agent-hosting servers, and between agent-hosting servers.

2. Description of the Related Art

Java virtual machines are rapidly becoming available on all kinds of computing platforms, from portables to desktops to workstations to mainframes. For the first time in computing history, there may soon be available a virtual, homogeneous platform for distributed and parallel computing on a global scale. The basic elements of this computing platform are distributed Java objects. Prior to JDK 1.1 [The Java Development Kit (JDK), URL=http://java.sun.com/products/jdk], one had to use the low-level socket-based class library (java.net package) for communication between distributed Java objects. With JDK 1.1, one can now use Java RMI (Remote Method Invocation) for direct method invocation between Java distributed objects. Java RMI raises the level of communication to that of objects, and it can pass objects by value using Java object Serialization. However, Java RMI is stationary (remote objects), rigid (predefined methods calls), point-to-point, and connection-oriented. Therefore, it is still too low-level and inflexible for direct use in many applications, such as agent-based applications.

The term "agent" has been used to mean different things in different contexts [D. Chess, B. Grosof, C. Harrison, D. Levine, C. Paris, and G. Tsudik, "Itinerant Agents for Mobile Computing", IBM Research Report, RC 20010, IBM Research Division, March 1995; C. Harrison, D. Chess, and A. Kershenbaum, "Mobile Agents: Are they a good idea?", IBM Research Report, IBM Research Division, March 1995.]—from intelligent agents to internet agents to mobile agents to task-specific agents to user agents, just to name a few. A key, distinct characteristic of agents, from our perspective, is that agents are autonomous. An agent has its own identity, thread of execution, and lifecycle. It is this characteristic that makes the agent system, and specifically Java agent system, a unique, flexible and powerful paradigm for distributed and parallel computing [D. T. Chang and D. B. Lange, "Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW", in Proceedings of the OOPSLA96 Workshop: Toward the Integration of WWW and Distributed Object Technology, October, 1996; MA'97 (First International Workshop on Mobile Agents 97), URL=http:www.informatik.uni-stuttgart.deripvr/vs/ws/ma971ma97.html].

Given that Java agents are autonomous and can be executing independently on various Java virtual machines throughout a vast computer network, what makes them useful and powerful in carrying out parallel and distributed computing is that they must be able to communicate with each other in a dynamic and flexible fashion: the mechanism must allow agents to communicate when one of the agents moves to a different address space (mobile agents), when they must communicate at a higher level than methods calls (intelligent agents), when they need to communicate as a group (collaborating agents), and when a part of computer network is down or one of the agents is not available (disconnected operation).



Most of the currently available Java agent systems have focused their support on agent mobility. They provide limited support for inter-agent communication. Among these, Voyager [Voyager, ObjectSpace, URL=http://www.objectspace.com/Voyager/voyager.html; "Voyager Core Package Technical Overview", ObjectSpace, March, 1997], Concordia [Concordia, Mitsubishi Electric ITA, URL=http://www.meitca.com/HSL/Projects/Concordia; "Concordia: An Infrastructure for Collaborating Mobile Agents", Mitsubishi Electric ITA, in First International Workshop on Mobile Agents 97 (MA'97), April, 1997; "Mobile Agent Computing", A White Paper, Mitsubishi Electric ITA, Feb. 28, 1997], and Aglets [Aglets Workbench, IBM, URL=http://www.trl.ibm.co.jp/aglets] are the best known.

Voyager defines the notion of a virtual object, which is basically a proxy to a remote object. In Voyager any object can be virtualized using a program called vc, which is a utility for converting regular classes to virtual classes. Messages are sent—via method calls—to remote objects through their local virtual references. Voyager messages can be sent in a synchronous, deferred, or asynchronous (one-way) mode. Object mobility is achieved through sending a "move" message to a remote object.

Concordia supports two types of asynchronous distributed events for inter-agent communication: selected events and group-oriented events. In the select-event messaging, an agent registers the type of events it would like to receive with an event manager. When the event manager receives an event of the registered type it forwards the event to the registered agent. Concordia also supports group-oriented events. An agent can join a group of agents. When one of the agents initiates an event, the event is forwarded to all the agents in the group. Agent mobility is achieved through the use of itineraries, which involves message passing between collaborating Concordia servers.

In Aglets, agents can communicate with each other by sending messages through their proxies. The messages can be sent in a synchronous or deferred mode. Agent mobility is achieved by directly dispatching an agent (through its proxy) or through the use of itineraries. This involves message passing between collaborating agent contexts using the agent transfer protocol.

CORBA [The Common Object Request Broker: Architecture and Specification, Revision 2.0, OMG, July 1995] provides an architecture for stationary objects to communicate with each other in a distributed and heterogeneous environment. It defines a framework for remote method invocation using the IIOP (Internet Inter-ORB Protocol). Under the cover this involves sending request messages and receiving response messages between collaborating hosts.

KQML is a language of communication for intelligent agents. KQML is based on using primitives called performatives. Performatives define permissible actions or operations that agents use for communication. A performative has

a name (which specifies what the performative means) and the following fields: sender, receiver, language (language of actual communication: prolog, SQL etc.), ontology (term definitions for the content), correlation id, and content.

JavaSpace ["JavaSpace Specification", Revision 0.3, Sun Microsystems, Inc. March 1997] is a Java adaptation for the internet of the pattern-matching shared memory paradigm provided by Linda [. Carriero and D. Gelemter, "Linda in Context", Communications of the ACM., 32(4), pp. 444–458, April, 1989].

Conventional methods have failed to provide a uniform, flexible and robust underlying communication infrastructure for agent systems for communication between agents, between agents and agent-hosting servers, and between agent-hosting servers. Thus, there is a clearly felt need for a method of, system for, and computer program product for, providing a flexible and robust underlying communication infrastructure for agent systems for communication between agents, between agents and agent-hosting servers, and between agent-hosting servers.

## SUMMARY OF THE INVENTION

A communication infrastructure providing communication between agents, between agents and agent-hosting servers, and between agent-hosting servers. The infrastructure meets technical requirements for flexibility and robustness: extensible types of messages, asynchronous and synchronous message passing, queuing, disconnected operation, inter-agent communication, and inter-agent-server communication. The communication infrastructure consists of three layers (from bottom to top): Mail Facility Layer, Message Facility Layer, and Agent Management Communication Facility Layer. The communication infrastructure has an open architecture in that a lower layer is designed to be more general than upper layers and can be used independent of the upper layers. Each upper layer, however, is designed to use and depends on the lower layers. The Mail Facility Layer is the lowest layer providing a general, semantics-free mail paradigm for asynchronous communication between distributed objects, whether they are local or remote to each other. The Mail Facility Layer provides a level of abstraction in terms of mail, virtual mailbox, post office, and mail queue, and hides the details of implementation and actual transport. It is designed to provide location transparency and to be implementable using various transport protocols. The next Message Facility Layer provides a typed messaging paradigm for asynchronous and synchronous message passing between distributed objects, whether they are local or remote to each other. The Message Facility Layer uses the Mail Facility Layer for sending messages and, where appropriate, for getting responses to requests sent. It allows for the association of typed message handlers with typed messages such that the format and semantics of messages are encapsulated through their types, are extensible, and can be processed by the associated message handlers. The Agent Management Communication Facility Layer is the highest layer providing the services for inter-agent communication between agents, agent-agent-server communication between an agent and an agent server, and inter-agent-server communication between agent servers for managing agents such as locating an agent, dispatching an agent, retrieving an agent, etc. The key abstractions provided in this layer include agent manager, agent, and agent identifier. It uses the Message Facility Layer and Mail Facility Layer to carry out the communication.

The present invention has the advantage of providing a flexible and robust underlying communication infrastructure for agent systems for communication between agents, between agents and agent-hosting servers, and between agent-hosting servers.

The present invention has the further advantage of providing extensible types of messages.

The present invention has the further advantage of providing asynchronous and synchronous message passing.

The present invention has the further advantage of providing queuing of message passing.

The present invention has the further advantage of providing disconnected operation for inter-agent communication and inter-agent-server communication.

The present invention has the further advantage of allowing implementations of various messaging paradigms to support distributed objects and agent mobility.

T She present invention has the further advantage of allowing implementations of various abstractions to be easily be built on top of the Mail Facility Layer, and facilitating multiple protocol implementations through the Mail Facility Layer.

## BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and the advantages thereof, reference is now made to the Description of the Preferred Embodiment in conjunction with the attached Drawings, in which:

FIG. **1** is a block diagram of a distributed computer system used in performing the method of the present invention, forming part of the apparatus of the present invention, and which may use the article of manufacture comprising a computer-readable storage medium having a computer program embodied in said medium which may cause the computer system to practice the present invention;

FIG. **2** is a block diagram of an agent communication infrastructure in accordance with the present invention;

FIG. **3** illustrates sending a mail through a virtual mailbox using the Mail Facility Layer of the present invention;

FIG. **4** illustrates receiving a mail through a virtual mailbox using the Mail Facility Layer of the present invention;

FIG. **5** illustrates a type hierarchy of messages and a type hierarchy of message handlers in accordance with the Message Facility Layer of the present invention;

FIG. **6** illustrates sending and receiving of a message using the Message Facility Layer of the present invention;

FIG. **7** illustrates agent communication in accordance with the Agent Management Communication Facility Layer of the present invention;

FIG. **8** is a flowchart illustrating the operations preferred in carrying out the send mail portion of the present invention;

FIG. **9** is a flowchart illustrating the operations preferred in carrying out the virtual mailbox portion of the present invention when sending mail;

FIG. **10** is a flowchart illustrating the operations preferred in carrying out the local main post office portion of the present invention when sending mail;

FIG. **11** is a flowchart illustrating the operations preferred in carrying out the destination post office portion of the present invention;

FIG. **12** is a flowchart illustrating the operations preferred in carrying out the receive mail portion of the present invention;

FIG. **13** is a flowchart illustrating the operations preferred in carrying out the virtual mail box portion of the present invention when receiving mail;

FIG. **14** is a flowchart illustrating the operations preferred in carrying out the local main post office portion of the present invention when receiving mail;

FIG. **15** is a flowchart illustrating the operations preferred in carrying out the location post office portion of the present invention;

FIG. **16** is a flowchart illustrating the operations preferred in carrying out the send message portion of the present invention;

FIG. **17** is a flowchart illustrating the operations preferred in carrying out the message portion of the present invention;

FIG. **18** is a flowchart illustrating the operations preferred in carrying out the receive message portion of the present invention;

FIG. **19** is a flowchart illustrating the operations preferred in carrying out the Request/Response Messaging portion of the present invention when sending a request message;

FIG. **20** is a flowchart illustrating the operations preferred in carrying out the Request/Response Messaging portion of the present invention when receiving a request message;

FIG. **21** is a flowchart illustrating the operations preferred in carrying out the Request/Response Messaging portion of the present invention when handling a request message;

FIG. **22** is a flowchart illustrating the operations preferred in carrying out the one-way mode of the Request/Response Messaging portion of the present invention;

FIG. **23** is a flowchart illustrating the operations preferred in carrying out the deferred mode of the Request/Response Messaging portion of the present invention;

FIG. **24** is a flowchart illustrating the operations preferred in carrying out the synchronous mode of the Request/Response Messaging portion of the present invention;

FIG. **25** is a flowchart illustrating the operations preferred in carrying out the agent communication portion of the present invention when a user sends an agent a message;

FIG. **26** is a flowchart illustrating the operations preferred in carrying out the agent communication portion of the present invention when a local main agent manager facilitates an agent message;

FIG. **27** is a flowchart illustrating the operations preferred in carrying out the agent communication portion of the present invention when an owning agent manager facilitates an agent message; and

FIG. **28** is a flowchart illustrating the operations preferred in carrying out the agent communication portion of the present invention when a destination agent manager facilitates an agent message.

## DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures and in particular with reference to FIG. **1**, there is depicted a pictorial representation of a distributed computer system **8** which may be utilized to implement the method of, system for, and article of manufacture of the present invention. As may be seen, distributed computer system **8** may include a plurality of networks **10** and **32**, which may be Local Area Networks (LAN), intranet networks, or internet networks, each of which preferably includes a plurality of individual computers **12** and **30**, respectively. Of course, those skilled in the art will appreciate that a plurality of Intelligent Work Stations (IWS) coupled to a host processor may be utilized for each such network.

As is common in such data processing systems, each individual computer may be coupled to a storage device **14** and/or a printer/output device **16**. One or more such storage devices **14** may be utilized, in accordance with the present invention, to store the various computer programs which may be accessed and executed by a user within the distributed computer system **8**, in accordance with the present invention. In a manner well known in the prior art, each such computer program may be stored within a storage device **14**.

Still referring to FIG. **1**, it may be seen that distributed computer system **8** may also include multiple mainframe computers, such as mainframe computer **18**, which may be preferably coupled to Local Area Network **10** by means of communication link **22**. Mainframe computer **18** may also be coupled to a storage device **20** which may serve as remote storage for Local Area Network **10** which may be coupled via communications controller **26** and communications link **34** to a gateway server **28**. Gateway server **28** is preferably an individual computer or Intelligent Work Station which serves to link Local Area Network **32** to Local Area Network **10**.

As discussed above with respect to Local Area Network **32** and Local Area Network **10**, a plurality of server computer programs may be stored within storage device **20** and executed by mainframe computer **18**. Similarly, a plurality of client computer programs may be stored within storage devices **14** and executed by individual computers **12** such that distributed client/server computer programs are provided. Of course, those skilled in the art will appreciate that the mainframe computer **18** may be located a great geographical distance from Local Area Network **10**, and similarly, Local Area Network **10** may be located a substantial distance from Local Area Network **32**. That is, Local Area Network **32** may be located in California while Local Area Network **10** may be located within Texas and mainframe computer **18** may be located in New York.

As will be appreciated upon reference to the foregoing, it is often desirable for a user within one portion of distributed data processing system **8** to execute agent computer programs on one or more portions of data processing system S. For example, the user may execute a client computer program on computer **12** which dispatches a mobile agent to execute on mainframe **18**. After obtaining services or information from mainframe **18**, the mobile agent may transfer to computer **30** to obtain further services or information from computer **30**. Finally, the mobile agent may be retrieved from computer **30** back to computer **12**.

System Architecture

The communication infrastructure consists of three layers (from bottom to top):

Mail Facility Layer **210**;

Message Facility Layer **220**; and

Agent Management Communication Facility Layer **230**.

The communication infrastructure has an open architecture in that a lower layer is designed to be more general than upper layers and can be used independent of the upper layers. Each upper layer, however, is designed to use and depends on the lower layers. FIG. **2** illustrates how the Agent Management Communication Facility Layer **230** uses and depends upon the lower Message Facility Layer **220** and Mail Facility Layer **210**. FIG. **2** also illustrates how the Message Facility Layer **220** uses and depends upon the lower Mail Facility Layer **210**. However, FIG. **2** also illustrates how the lower Message Facility Layer **220** and Mail Facility Layer **210** may be used independently of the upper Agent Management Communication Facility Layer **230**, and

how the lower Mail Facility Layer **210** may be used independently of the upper Message Facility Layer **220**.

The Mail Facility Layer **210** is the lowest layer and the foundation of the communication infrastructure. It provides a general, semantics-free mail paradigm for asynchronous communication between distributed objects such as Java objects, whether they are local or remote to each other. The Mail Facility Layer **210** provides a level of abstraction in terms of mail, virtual mailbox, post office, and mail queue, and hides the details of implementation and actual transport. It is designed to provide location transparency and to be implementable using various transport protocols. Table 2 comprises a package index of the preferred embodiment of the present invention; Table 4 comprises class definitions of the mail package of the preferred embodiment of the present invention; Table 5 comprises class definitions of the mail implementation package of the preferred embodiment of the present invention; Table 8 comprises a class hierarchy of the preferred embodiment of the present invention; and Table 9 comprises an index of all fields and methods of the preferred embodiment of the present invention.

The next layer in the communication infrastructure is the Message Facility Layer **220**. It provides a typed messaging paradigm for asynchronous and synchronous message passing between Java objects, whether they are local or remote to each other. The Message Facility Layer **220** uses the Mail Facility Layer **210** for sending messages and, where appropriate, for getting responses to requests sent. It allows for the association of typed message handlers with typed messages such that the format and semantics of messages are encapsulated through their types, are extensible, and can be processed by the associated message handlers. Table 6 comprises class definitions of the message package of the preferred embodiment of the present invention, and Table 7 comprises class definitions of the message handler package of the preferred embodiment of the present invention.

The Agent Management Communication Facility Layer **230** is the highest layer of the communication infrastructure. It provides the services for inter-agent communication between agents, agent-agent-server communication between an agent and an agent server, and inter-agent-server communication between agent servers for managing agents such as locating an agent, dispatching an agent, retracting an agent, etc. The key abstractions provided in this layer include agent manager, agent, and agent identifier. It uses the Message Facility Layer **220** and Mail Facility Layer **210** to carry out the communication. Table **3** comprises class definitions of the agent package of the preferred embodiment of the present invention.

Mail Facility

The Mail Facility Layer **210** provides an asynchronous mail delivery service. Java objects, whether they are local or remote to each other, can use the facility to communicate with each other in an asynchronous manner. A key concept and innovation is that of the virtual mailbox. To send a mail **305**, one needs to simply open a virtual mailbox **310** with the name **315** that represents the destination for the mail **305** and put the mail in the virtual mailbox **310**. The Mail Facility Layer **210** will do the rest and deliver the mail **305** to the physical destination represented by the name **315**, be it local or remote. This is shown in FIG. **3**, where Mail **305**, Mailbox **310**, PostOffice **320**, PORemote **325**, and MailQueue **330** are Java classes or interfaces.

To receive a mail **405**, one again needs to simply open a virtual mailbox **410** with the name **415** that represents the location for the mail and get the mail **405** from the virtual mailbox **410**. The Mail Facility Layer **210** will do the rest and retrieve the mail **405** from the physical location represented by the name **415**, be it local or remote, as illustrated in FIG. **4**.

The Mail Facility Layer **210** uses the abstraction of post office (**320**, **325**, **420**, and **425**) and mail queue (**330**, **335**, **340**, **435**, and **440**) to encapsulate and hide the details of implementation and actual transport. The post office can be implemented using various transport protocols (**345** and **445**) and can support multiple transport protocols at the same time. The mail queue (**330**, **335**, **340**, **435**, and **440**) provides the store and forward, and persistence capabilities to support asynchronous and disconnected operations.

Each mail (**305** and **405**) is designed to have an unique identifier, a correlation identifier, which can be used to correlate related mail, e.g., between responses and requests. To allow for content of various types to be sent and received by mail, each mail has a type specification which is extensible and which does not require name registration. An example of this is given in the discussion of the Message Facility Layer **220**. A mail can be given a priority to facilitate its processing by the receiver.

As mentioned before, the virtual mailbox (**310** and **410**) provides location transparency when sending and receiving mail. Each virtual mailbox **310** is associated with a name **315** which represents some destination or location, local or remote. The name **315** ties it with a certain post office **325**, which is the logical home of the virtual mailbox **310**. When sending a mail **305**, one simply puts it in a virtual mailbox **310** with the appropriate name **315**. When receiving a mail **405**, one gets it from a virtual mailbox **410** with the appropriate name **415**. If needed, one can specify the type or correlation identifier of the mail to be received.

The post office (**320**, **325**, **420**, and **425**) does the actual sending and receiving of mail. Each post office (**320**, **325**, **420**, and **425**) is associated with a name (**355**, **350**, **455**, and **450** respectively) which represents its location and identification, and which may include the specification of the mail transport protocol (**345**, **445**) (M, IIOP, or MQSeries, for example) to be used by the post office. If the protocol is not specified, the default protocol is implied. A mail **305** can be sent through any known local post office **320**, which in turn will deliver the mail **305** to the appropriate destination post office **325**. A mail **405** can be received through any known local post office **420**, which in turn will retrieve the mail from the appropriate location post office **425**, which is the physical location of the mail to be received. If needed, one can specify the type or correlation identifier of the mail to be received.

The post office **320** stores mail in mail queues (**330** and **340**). It maintains various receiving mail queues **340** for the virtual mailboxes that it owns. Additionally it maintains some sending mail queues **330** which allow it to handle disconnected operation and to optimize mail delivery. The quality of service provided by a mail queue depends on its implementation (e.g., in memory, using files, using databases).

From the above discussion it can be seen that the Mail Facility Layer **210** is a general purpose, asynchronous mail delivery service for Java objects. It forms a flexible and robust foundation for the communication infrastructure. It provides virtual mailboxes for one to send and receive mail in a location transparent manner. The mail can contain various types of content with extensible type specifications. And it utilizes mail queues to provide store and forward, and persistence capabilities.

Message Facility

The Message Facility Layer **220** provides a typed message paradigm for asynchronous and synchronous message

passing between Java objects, whether they are local or remote to each other. It allows for the association of typed message handlers (**501**, **502**, **503**, **504**, **505**, and **506**) with typed messages (**511**, **512**, **513**, **514**, **515**, and **516** respectively) such that the format and semantics of messages are encapsulated through their types and can be processed by the associated message handlers. As such, both messages and their associated message handlers can be easily extended and doing so without the need of a naming authority. The message type hierarchy **510** and the associated message handler type hierarchy **500** are illustrated in FIG. **5**, where Message **511** , MessageHandler **501**, etc. are Java interfaces or classes, and arrows are used to indicate inheritance.

An important consideration in the design of the Message Facility Layer **220** is that, in general, agents need to communicate with each other using many different types of messages: event messages ["Concordia: An Infrastructure for Collaborating Mobile Agents", Mitsubishi Electric ITA, in First International Workshop on Mobile Agents 97 (MA'97), April, 1997; "Mobile Agent Computing", A White Paper, Mitsubishi Electric ITA, Feb. 28, 1997], KQML messages [InfoSleuth Project, URL=http://www.mcc.com/projects/infosleuth; JKQML, IBM, URL=http://objects.yamato.ibm.com/JKQML/index-e.html; Y. Labrou, "Semantics for an Agent Communication Language", Ph.D. thesis, CSEE department, University of Maryland, Baltimore Md. 21228-5398, September, 1996. URL=http://www.cs.umbc.edu/kqml], method invocation messages [The Common Object Request Broker: Architecture and Specification, Revision 2.0, OMG, July 1995], request/response messages [Aglets Workbench, IBM, URL=http://www.trl.ibm.co.jp/aglets], etc. A similar consideration is that agent-hosting servers also need to communicate with each other using many different types of messages in order to manage agents: agent transfer messages [Aglets Workbench, IBM, URL=http://www.trl.ibm.cojp/aglets], agent query messages, statistics gathering messages, etc. Therefore, a key requirement in the design of the Message Facility Layer **220** is that it must support multiple types of messages and message handlers, and that it must support their extension and identification with ease. These are accomplished through the use of Java interface/class hierarchies and design patterns.

The Message Facility Layer **220** allows for asynchronous and synchronous message passing. Certain types of messages, such as event messages and KQML messages, are asynchronous in nature and are sent one-way. Other messages, such as method invocation messages and agent transfer messages, involve responses and can be sent in one of three different modes: synchronous (waiting for responses) **2400** of FIG. **24**, deferred (getting responses at a later time) **2300** of FIG. **23**, or one-way (asynchronous, discarding the responses) **2200** of FIG. **22**.

A key design decision and innovation is to integrate the allowed message passing mode with the type of messages rather than treat it as orthogonal to the type of messages. This is based on the observation that for messages such as event messages or KQML messages, it does not make sense to send them in a synchronous or deferred mode. This also allows message definers the freedom to choose the appropriate mode(s) for their types of messages. Table 1 shows exemplary message types in accordance with the preferred embodiment of the present invention.

TABLE 1

| | Mode | | | |
|---|---|---|---|---|
| Message Type | Asynchronous One-Way | | Synchronous | Deferred |
| Event | X | | | |
| KQML | X | | | |
| Request/Response | X | | X | X |
| Agent Transfer | | | X | X |

To send a message **605** of FIG. **6** via the Message Facility Layer **220**, a user first creates a message **605** and then calls send() which creates a mail **610** specifying the type of content and the content wherein the type of content is the type of the message and the content is the message **605**. The message **605** also creates a virtual mailbox **615** specifying the virtual mailbox name **620** that represents the destination for the mail **625**, which is the receiver virtual mailbox of the message **605**. Thereafter, the message **605** puts the mail **610** in the virtual mailbox **615** which causes the Mail Facility Layer **210** to deliver the message **605** to the destination **625**.

To receive a message via the Message Facility Layer **220**, a user creates a virtual mailbox **665** specifying the virtual mailbox name **670** that represents the physical location **675** of the mail. The user then gets the mail **660**, whose content is the message **655**, from the virtual mailbox **665** which causes the Mail Facility Layer **210** to get the mail **660** from the location post office **675** through the local main post office **680** into the virtual mailbox **665**.

In summary, the Message Facility Layer **220** is a general purpose, message passing service for Java objects and uses the Mail Facility Layer **210** for actual message delivery. It serves as a flexible middle layer for the communication infrastructure. It provides an extensible framework for handling typed messages and associated handlers, and it allows for message passing, where appropriate, in asynchronous, synchronous, or deferred mode.

Agent Management Communication Facility

The Agent Management Communication Facility Layer **230** is the highest layer of the communication infrastructure. It is designed to provide a uniform scheme for handling inter-agent communication, whether the agents involved are stationary or mobile, and inter-agent-server communication. It uses the Message Facility Layer **220** and Mail Facility Layer **210** to carry out the communication.

A key abstraction provided is that of an agent manager (**705**, **710**, and **715**). An agent manager manages a group of agents, stationary or mobile, and is responsible for working with other agent managers to locate an agent, send a message to an agent, dispatch an agent, retrieve an agent, etc. Each agent is autonomous and has an agent identifier which uniquely identifies it regardless whether it moves or not. The message passing between agents is illustrated in FIG. **7**, where AgentManager (**705**, **710**, and **715**) and Agent (**720** and **725**) are Java classes.

Inter-agent Communication

Communication between mobile agents is done through the collaboration of agent managers. A mobile agent's owning agent manager is aware of an agent's whereabouts at all times and can cause appropriate message forwarding to the current location of an agent. A local agent manager manages agents located at the local agent manager's location or system. If there are one or more local agent managers at a location, then a local main agent manager is the default agent manager. If a mobile agent moves to a location other than that of its owning agent manager, then the managing agent manager at that current location of the agent is known as a destination agent manager.

Each agent manager (**705**, **710**, and **715**) owns one or more virtual mailboxes (**730**, **735**, **740**, and **790**) and uses them to exchange messages (**745**, **750**, and **755**) via mail (**760**, **765**, **770**, and **785**). For example, if an agent manager **705** is asked to send a message **745** to an agent **720**, it will generate a mail **760** (with the message **745** encapsulated as its content) and put in the owning agent manager's virtual mailbox **730**. The owning agent manager **710** then will, at an appropriate time, receive the message **750**, thus causing a mail **765** to be retrieved from its virtual mailbox **735**. If the agent **720** is owned and managed by owning agent manager **710**, then the owning agent manager **710** will locate the agent **720** and request the agent **720** to handle the message **750**.

If the agent **725** is owned, but not managed by, agent manager **710**, then the message needs to be redirected to the managing destination agent manager **715**. In this situation, owning agent manager **710** generates a mail **785** (with the message **750** encapsulated as its content) and puts the mail **785** in the destination agent manager's **715** virtual mailbox **790**. The destination agent manager **715** then will, at an appropriate time, receive the message **755**, thus causing a mail **770** to be retrieved from its virtual mailbox **740**. The destination agent manager **715** will then locate the agent **725** and request the agent **725** to handle the message **755**.

From the above discussion it can be seen that the present invention provides a uniform facility for communication between agents, whether they are stationary or mobile, and for communication between agent managers for the purpose of managing agents, e.g., transferring an agent to a new location.

Referring next to FIG. **8** through FIG. **28**, flowcharts illustrating operations preferred in carrying out the present invention are shown. In the flowcharts, the graphical conventions of a diamond for a test or decision and a rectangle for a process or function are used. These conventions are well understood by those skilled in the art, and the flowcharts are sufficient to enable one of ordinary skill to write code in any suitable computer programming language.

Mail Facility Layer Preferred Embodiment

Referring first to FIG. **8** through FIG. **15**, the operations preferred in carrying out the Mail Facility Layer **210** of the present invention are illustrated. FIG. **8** through FIG. **11** illustrate the operations preferred in sending mail, and FIG. **12** through FIG. **15** illustrate the operations preferred in receiving mail. To send mail, a user creates a mail **305** specifying the type of content and the content by use of API (Application Program Interface) <new Mail(type, content)>(process block **810** of FIG. **8**). The user may also set a response destination for the mail **305** so that a reply may be returned to that destination or so that the mail **305** can be returned in case of delivery failure, <Mail.setResponseDestination(respDest)>(process block **820**). The user then creates a virtual destination mailbox **310** by specifying a virtual mailbox name **315** that represents a destination for the mail **325**, <new Mailbox(mbName)> (process block **830**). The virtual mailbox name **315** comprises a post office name **350** and a mailbox id. The post office name **350** comprises a protocol, host, port, and post office id. This supports multiple protocols (e.g., RMI, which is the default), and the protocol determines the type of post office to be used for sending/receiving mail (e.g., an RMI post office is used in process block **920**). Thereafter, the user puts the mail **305** in the virtual mailbox **310**, <Mail.putMail (mail)>(process block **840**).

Referring next to FIG. **9**, the operations preferred in carrying out the virtual mailbox (**310** and **900**) portion of the Mail Facility Layer **210** of the present invention are illustrated. After the mail **305** has been put into the virtual mailbox **310** by process block **840**, process block **920** of FIG. **9** causes the virtual mailbox **310** to get the local main post office **320**, <PostOfficeRMI.getLocalPostOffice()>, and process block **930** sends the mail **305** through the local main post office **320**, <PostOffice.sendMail(mbName, mail)>.

Referring now to FIG. **10**, the operations preferred in carrying out the local main post office (**320** and **1000**) portion of the Mail Facility Layer **210** of the present invention are illustrated. The local main post office **320** extracts the destination post office name **350** from the virtual mailbox name **315** and uses it to find the destination post office **325**, whether local or remote (process block **1010**). If the destination post office is not available, then the local main post office **320** may put the mail **305** in the sending mail queue **330** and repeat this step later. This supports disconnected operation. Thereafter, the local main post office **320** generates and sets the mail id for the mail, <Mail.setMailId(mailId)>(process block **1020**), and sets the destination for the mail specifying the virtual mailbox name **315**, <Mail. setDestination(mbName)>(process block **1030**). Process block **1040** then causes the local mail post office **320** to deliver the mail **305** to the destination post office **325** specifying the mailbox id, which is extracted from the virtual mailbox name, <PostOfficeRMI.deliverMail(mbId, mail)>.

Referring now to FIG. **11**, the operations preferred in carrying out the destination post office (**325** and **1100**) portion of the Mail Facility Layer **210** of the present invention are illustrated. After the mail **305** is received at the destination post office **325**, the destination post office **325** obtains the receiving mail queue **335** for the mail **305** using the mailbox id (process block **1120**), and then the destination post office **325** puts the mail **305** in the mail queue **335**, <MailQueue.enqueue(mail)>(process block **1130**). Different types of mail queues may provide different quality of service. For example, in-memory mail queue <MemMailQueue>provides fast access, whereas database mail queue <SQLMailQueue>provides persistent storage of mail. The use of these is determined by the post office and is transparent to the user.

Referring now to FIG. **12** through FIG. **15**, the operations preferred in receiving mail are illustrated. Referring first to FIG. **12**, the user operations preferred in receiving mail via a virtual mailbox **410** are illustrated. A user creates a virtual mailbox **410** specifying the virtual mailbox name **415** that represents the physical location of the mail located at post office **425**, <new Mailbox(mbName)>(process block **1210**). Thereafter, the user gets the mail **405** from the virtual mailbox **410**, <Mail.getMail()>(process block **1220**). Alternatively, the user may get the mail with a specific type of content or a specific correlation identifier. The correlation identifier or correlation id associates two or more pieces of mail, for example a mail object and a reply to that mail object.

Referring next to FIG. **13**, the operations preferred in the virtual mailbox **410** when receiving mail are illustrated. In response to the user getting the mail **405** from the virtual mailbox **410** (process block **1220**), the virtual mailbox **410** gets the local main post office **420**, <PostOfficeRMI.getLocalPostOffice()>(process block **1310**), and thereafter the virtual mailbox **410** receives the mail **405** through the local main post office **420**, <PostOffice.receiveMail(mbName)>(process block **1320**). Alternatively, the virtual mailbox may receive the mail with a specific type of content or a specific correlation id.

Referring next to FIG. **14**, the operations preferred in the local main post office **420** when receiving mail are illustrated. In response to the virtual mailbox **410** getting the local main post office **420** (process block **1320**), the local main post office **420** extracts the location post office name **450** from the virtual mailbox name **415** and uses it to find the location post office **425**, whether local or remote (process block **1410**). The location post office is the physical location of the mail to be received. If the location post office **425** is not available, the local main post office **420** will repeat process block **1410** later, thus supporting disconnected operation. Thereafter, the local mail post office **420** retrieves the mail **405** from the location post office **425** specifying the mailbox id, which is extracted from the virtual mailbox name **415**, <PostOfficeRMI.retrieveMail(mbId, mail)> (process block **1420**).

Referring next to FIG. **15**, the operations preferred in the location post office **425** when receiving mail are illustrated. In response to the local mail post office **420** retrieving the mail **405** from the location post office **425** (process block **1420**), the location post office **425** obtains the receiving mail queue **435** for the mail **405** using the mailbox id (process block **1510**), and then gets the mail **405** from the mail queue **435**, <MailQueue.dequeue()>(process block **1520**). Alternatively, the location post office **425** may get the mail with a specific type of content or a specific correlation id. The mail obtained by process block **1520** is returned to process block **1420** which causes the mail to be returned to process block **1320** which causes the mail to be returned to process block **1220**.

Message Facility Layer Preferred Embodiment

Referring now to FIG. **16** through FIG. **18**, the operations preferred in carrying out the Message Facility Layer **220** of the present invention are illustrated. FIG. **16** illustrates sending a message, FIG. **17** illustrates the sending of a message via the Mail Facility Layer **210**, and FIG. **18** illustrates the receiving of a message.

Referring first to FIG. **16** illustrating the operations preferred in sending a message, a user creates a message **605** of a particular type, <e.g., new KQMLMessage()>(process block **1610**). The user sets the sender of the message, <Message.setSender()>(process block **1620**); sets the receiver of the message, <Message.setReceiver()>(process block **1630**), and sets other attributes of the message as needed, <e.g., KQMLMessage.setPerformative(pero)> (process block **1640**). The user then sends the message, <Message. send()>(process block **1650**).

Referring next to FIG. **17** illustrating the operations preferred in the sending of a message **605** via the Mail Facility Layer **210**, the message **605** creates a mail **610** specifying the type of content and the content wherein the type of content is the type of the message and the content is the message **605**, <new Mail(type, content)>(process block **1710**). The message **605** may also set a response destination for the mail **610** so that a reply may be returned to the destination or so that it can be returned in case of delivery failure, <Mail.setResponseDestination(respDest)>(process block **1720**). The response destination is the sender virtual mailbox of the message. The message **605** also creates a virtual mailbox **615** specifying the virtual mailbox name **620** that represents the destination for the mail **625**, <new Mailbox(mbName)>(process block **1730**), which is the receiver virtual mailbox of the message **605**. Thereafter, the message **605** puts the mail **610** in the virtual mailbox **615**, <Mail.putMail(mail)>(process block **1740**), which invokes the Virtual Mailbox processing **900** of the Mail Facility Layer **210** starting with process block **920** of FIG. **9** wherein the message **605** is the user of the Mail Facility Layer **210**.

Referring next to FIG. **18** illustrating the operations preferred in the receiving of a message via the Message Facility Layer **220**, a user creates a virtual mailbox **665** specifying the virtual mailbox name **670** that represents the physical location **675** of the mail, <new Mailbox (mbName)> (process block **1810**). The user then gets the mail **660** from the virtual mailbox **665**, <Mail.getMail()> (process block **1820**), which invokes the Virtual Mailbox processing **1300** of the Mail Facility Layer **210** starting with process block **1310** of FIG. **13** wherein the message **655** is the user of the Mail Facility Layer **210**. After process block **1320** has received the mail **660** into the virtual mailbox **665** from the local main post office **680**, the user gets the content of the mail **660**, which is the message **655**, <Mail.getContent()>(process block **1830**). Alternatively, process block **1820** may allow the user to optionally get the mail with a specific type of content which represents the type of message to be received.

Request/Response Messaging Preferred Embodiment

Referring next to FIG. **19** through FIG. **24**, the operations preferred in carrying out the Request/Response Messaging of the present invention are illustrated. FIG. **19** illustrates the operations preferred in sending a request message; FIG. **20** illustrates the operations preferred in receiving a request message; FIG. **21** illustrates the operations preferred in carrying out the Request Message Handler; FIG. **22** illustrates the operations preferred in a one-way mode of Request/Response Messaging; FIG. **23** illustrates the operations preferred in a deferred mode of Request/Response Messaging; and FIG. **24** illustrates the operations preferred in a synchronous mode of Request/Response Messaging.

Referring next to FIG. **19** illustrating the operations preferred in the sending of a request message, a user first creates a request message, <new RequestMessage()> (process block **1910**). The user sets the receiver of the message, <Message.setReceiver()>(process block **1920**); the sender of the message, <Message. setsender()>(process block **1930**); and other attributes of the message as needed, <e.g., RequestMessage.setOperation(oper)>(process block **1940**). Thereafter, the user sends the message, <RequestMessage.send()>(process block **1950**) which invokes process block **1650** of FIG. **16** and processing continues by the Message Facility Layer **220**.

Referring next to FIG. **20** illustrating the operations preferred in the receiving of a request message, the system (e.g., an agent manager) first creates a virtual mailbox specifying the virtual mailbox name that represents the virtual mailbox, <new Mailbox(mbName)>(process block **2010**). The system then gets the mail from the virtual mailbox, <Mail.getMail()>(process block **2020**), using the virtual mailbox processing **1300** of FIG. **13**, and the content of the mail, which is a clone of the request message, <Mail.getContent(>(process block **2030**). Thereafter, the system sets the status of the request message to indicate that it is a clone, <RequestMessage.setClone()>(process block **2040**). The system then creates a request message handler, <new RequestMessageHandler()>(process block **2050**), and the system asks the request message handler to handle the (cloned) request message, <RequestMessageHandler.handleMessage(reqc)>(process block **2060**).

Referring next to FIG. **21** illustrating the operations preferred in the carrying out of the Request Message Handler, the handler handles the message and generates a result (process block **2110**). Thereafter, the Request Message Handler determines if a one-way attribute is set (<RequestMessage.isOneway()>) (decision block **2120**),

and if the one-way attribute is set, then process block **2125** discards the result. Returning now to decision block **2120**, if the one-way attribute is not set, then the Request Message Handler creates a response message, <new ResponseMessage(result)>(process block **2130**). Thereafter, the Request Message Handler sets a correlation identifier or correlation id of the response message to be the message identifier or message id of the request message, <ResponseMessage.setCorrelationId(corrId)>(process block **2140**). The Request Message Handler also sets the receiver of the response message to be the sender of the request message, <Message.setReceiver(receiver)>(process block **2150**), and sets the sender of the response message to be the receiver of the request message, <Message.setSender (sender)>(process block **2160**). Thereafter, the Request Message Handler sends the response message, <Message.send()>(process block **2170**), invoking process block **1650** of the sending message portion **1600** of the Message Facility Layer **220**.

Referring next to FIG. **22**, FIG. **23**, and FIG. **24**, three modes of sending the request message and getting the result are illustrated: one-way in FIG. **22**, deferred in FIG. **23**, or synchronous in FIG. **24**.

Referring first to FIG. **22** illustrating the operations preferred in a one-way mode of Request/Response Messaging, if the request message mode is set to one-way before sending (<RequestMessage.setOneway()>), then the request message is sent as a one-way request message, and the result is discarded (process block **2210**).

Referring next to FIG. **23** illustrating the operations preferred in a deferred mode of Request/Response Messaging, the user first checks if the result has arrived, <RequestMessage.checkResult()>(process block **2310**). Thereafter, the user determines if a the result has arrived (decision block **2320**), and if the result has arrived, then the user gets the result (process block **2330**). Returning now to decision block **2320**, if the result has not arrived, then the user can repeat process block **2310** and decision block **2320** at a later time.

Referring next to FIG. **24** illustrating the operations preferred in a synchronous mode of Request/Response Messaging, the user waits for the result until the result arrives, and gets it, <RequestMessage.getResult(−1)> (process block **2410**).

Agent Communication Facility Layer Preferred Embodiment

Referring next to FIG. **25** through FIG. **28**, the operations preferred in carrying out the Agent Communication Facility Layer **230** of the present invention are illustrated. FIG. **25** illustrates the operations preferred in carrying out the agent communication portion of the present invention when a user sends an agent a message; FIG. **26** illustrates the operations preferred in carrying out the agent communication portion of the present invention when a local agent manager facilitates an agent message; FIG. **27** illustrates the operations preferred in carrying out the agent communication portion of the present invention when an owning agent manager facilitates an agent message; and FIG. **28** illustrates the operations preferred in carrying out the agent communication portion of the present invention when a destination agent manager facilitates an agent message.

Referring next to FIG. **25** illustrating the operations preferred in carrying out the agent communication facility layer portion of the present invention when a user sends an agent a message, a user first gets the local main agent manager **705**, <AgentManager.getLocalAgentManager()> (process block **2510**), wherein the local main agent manager is the default agent manager for managing agents located at the local main agent manager's location or system. Thereafter, the user creates a message **745** of a particular type (e.g., KQMLMessage) (process block **2520**), and sets the attributes of the message as needed, <e.g., KQMLMessage.setPerformative(perf)>(process block **2530**). The user then sends the message through the local main agent manager **705**, <AgentManager.sendMessage (aid, msg)>(process block **2540**).

Referring next to FIG. **26** illustrating the operations preferred in carrying out the agent communication facility layer portion of the present invention when a local agent manager facilitates an agent message, the local main agent manager **705** first sets the receiver of the message to the owning agent manager **710** URL (Uniform Resource Locator) in string form concatenated with the agent id in string form, <Message.setReceiver(receiver)>(process block **2610**). The owning agent manager is aware of an agent's whereabouts at all times and can cause appropriate message forwarding to the current location of an agent. The local main agent manager **705** also sets the sender of the message to the its URL (string form), <Message. setSender (sender)>(process block **2620**), and then sends the message, <Message.send()>(process block **2630**), invoking process block **1650** of the sending message portion **1600** of the Message Facility Layer **220**.

Referring next to FIG. **27** illustrating the operations preferred in carrying out the agent communication facility layer portion of the present invention when an owning agent manager facilitates an agent message, the owning agent manager **710** first creates a virtual mailbox **735** specifying the virtual mailbox name **775** that represents it's virtual mailbox wherein the virtual mailbox name is a string form of its URL, <new Mailbox(mbName)>(process block **2710**). Thereafter, the owning agent manager **710** gets the mail **765** from the virtual mailbox **735**, <Mailbox.getMail(>(process block **2720**), and gets the content of the mail **765**, which is the message **750**, <Mail.getContent()>(process block **2730**). The owning agent manager **710** also gets the receiver **720** of the message wherein the receiver is the agent id in string form, <Message.getReceiver)>(process block **2740**), and gets the managed and owned agent **720**, <AgentManager.getAgent(aid)>(process block **2750**). Thereafter, the owning agent manager **710** determines if an agent **720** exists locally (decision block **2760**), and if the agent **720** exists locally, then the owning agent manager **710** asks the managed agent **720** (managed agent if local) to handle the message **750**, <Agent.handleMessage()>(process block **2770**). Returning now to decision block **2760**, if the agent does not exist locally (if the agent has moved to a location managed by a different agent manager **715**), then the owning agent manager **710** gets the destination **715** of the owned agent **725** wherein the destination is the URL in string form of the destination agent manager **715** (process block **2775**). This is the situation where the agent **725** has moved away from it's owning agent manager **710**. Whenever it does so, the owning agent manager **710** is informed of it's new destination and the destination agent manager **715**. Thereafter, the owning agent manager **710** sets the receiver of the message wherein the receiver is set to the destination agent manager **715** URL in string form concatenated with the agent id in string form, <Message.setReceiver(receiver>(process block **2780**). The owning agent manager **710** then sends the message **755**, <Message.send()>(process block **2790**).

Referring next to FIG. **28** illustrating the operations preferred in carrying out the agent communication portion of

the present invention when a destination agent manager facilitates an agent message, the destination agent manager creates a virtual mailbox specifying the virtual mailbox name that represents it's virtual mailbox wherein the virtual mailbox name is a string form of it's URL, <new Mailbox (mbName)>(process block **2810**). The destination agent manager is the managing agent manager at the current location of the agent if the agent moves to a location other than that of its owning agent manager. Thereafter, the destination agent manager gets the mail from the virtual mailbox, <Mailbox.getMail()>(process block **2820**); gets the content of the mail which is the message, <Mail.getContent()>(process block **2830**); and gets the receiver of the message wherein the receiver is the agent id in string form, <Message.getReceiver()>(process block **2840**). The destination agent manager then gets the managed and owned agent, <AgentManager.getAgent(aid)>(process block **2850**), and asks the managed agent to handle the message, <Agent.handleMessage()>(process block **2860**).

Although the present invention has been particularly shown and described with reference to a preferred embodiment, it will be understood by those skilled in the art that various changes in form and detail may be made without departing from the spirit and the scope of the invention.

TABLE 2

API User's Guide Class Hierarchy Index
Package Index
Other Packages
package com.ibm.jma.agent
package com.ibm.jma.mail
package com.ibm.jma.mail.input
package com.ibm.jma.message
package com.ibm.jma.message.handler

TABLE 3

All Packages Class Hierarchy Index
package com.ibm.jma.agent
Class Index
Agent
AgentID
AgentManager
Exception Index
AgentException
All Packages Class Hierarchy This Package Previous Next Index
Class com.ibm.jma.agent.Agent
java.lang.Object
. . . com.ibm.jma.agent.Agent
public abstract class Agent
extends Object
implements Cloneable, MessageHandler
This is the abstract, Toot class of all agents, stationary or mobile. Each agent has a globally unique identifier and is managed by an agent manager. An agent can communicate with other agents using messages.
See Also:
AgentManager, AgentID, Message
Constructor Index
Agent()
Method Index
getID()
Gets the identifier of this agent.
GetManager()
Gets the (current) agent manager of this agent.
getMessageTypes()
Gets the message types which can be handled by this agent.
handleMessage(Message)
Handles the specified message.
init(AgentManager, AgentID, Object)
Constructors
Agent
protected Agent()
Methods
init
protected void init(AgentManager am, AgentID aid, Object init)
getID
public final AgentID getID()
Gets the identifier of this agent
getManager
public final AgentManager getManager()
Gets the (current) agent manager of this agent.
handleMessage
public boolean handleMessage(Message msg)
Handles the specified message.
getMessageTypes
public String getMessagetypes()
Gets the message types which can be handled by this agent
Returns:
message types concatenated in a string and separated by spaces.
All Packages Class Hierarchy This Package Previous Next Index
All Package Class Hierarchy This Package Previous Next Index
Class com.ibm.jma.agent.AgentID
java.lang.object
. . . com.ibm.jma.agent.AgentID
public final class AgentID
extends Object
implements Serializable
An AgentID object encapsulates an agent's identifier.
Constructor Index
AgentID(byte[])
Constructs an agent identifier from the specified byte array representation.
AgentID(String)
Constructs an agent identifier from the specified string representation.
Method Index
equals(Object)
Test if the specified object is an agent identifier and is equal to this agent identifier.
getID()
Gets the byte array representation of this agent identifier.
hashCode()
Returns the hash code for this agent identifier.
toString()
Gets the string representation of this agent identifier.
Constructors
AgentID
public AgentID(byte bid[])
Constructs an agent identifier from the specified byte array representation.

AgentID
public AgentID(String sid)
Constructs an agent identifier from the specified string representation.
Methods
getID
public byte[] get()
Gets the byte array representation of this agent identifier.
toString
public String toString()
Gets the string representation of this agent identifier.
Overrides:
toString in class Object
equals
public boolean equals(Object obj)
Test if the specified object is an agent identifier and is equal to this agent identifier.
Overrides:
equals in class Object
hashCode
public int hashcode()
Returns the hash code for this agent identifier.
Overrides:
hashCode in class Object
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class com.ibm.jma.agent.AgentManager
java.lang.Object
. . . com.ibm.jma.agent.AgentManager
public final class AgentManager
extends Object
implements MessageHandler
An agent manager manages agents, stationary or mobile, including their communication, mobility, etc. Each agent manager may collaborate with other agent managers to accomplish its tasks.
See Also:
Agent, AgentID, Message
Field Index
DEFAULT_AM_NAME
DEFAULT_PORT_NUMBER
DEFAULT_PROTOCOL
Method Index
createAgent(URL, String, Object)
Creates an agent with the specified codebase, class name, and initialization.
dispatchAgent(agent, String)
Dispatches an agent to the specified destination.
getAgent(AgentID)
Gets the agent managed by this agent manager with the specified agent identifier.
getAgents()
Gets all agents managed by this agent manager.
getDestination(Agent ID)
Gets the destination of the agent owned by this agent manager.
getLocalAgentManager()
Gets the local agent manager.
getMessageTypes()
Gets the message types which can be handled by this agent manager.
getName()
Gets the name of this agent manager.
getURL()
Gets the url of this agent manager.
handleMessage(Message)
Handles the specified message.
retrieveAgent(AgentID)
Retrieves the agent with the specified agent identifier.
sendMessage(AgentID, Message)
Sends the specified message to the agent with the specified identifier.
Fields
DEFAULT_AM_NAME
public static final String DEFAULT_AM_NAME
DEFAULT_PROTOCOL
protected static final String DEFAULT_PROTOCOL
DEFAULT_PORT_NUMBER
protected static final int DEFAULT_PORT_NUMBER
Methods
getLocalAgentManager
public static AgentManager getLocalAgentManager()
Gets the local agent manager
getURL
public URL getURL()
Gets the url of this agent manager.
getName
public String getName()
Gets the name of this agent manager.
handleMessage
public boolean handleMessage(Message msg)
Handles the specified message.
Returns:
true if the specified message can be handled.
getMessageTypes
public String getMessageTypes()
Gets the message types which can be handled by this agent manager.
Returns:
message types concatenated in a string and separated by spaces
createAgent
public Agent createAgent(URL codebase, String classname, Object init) throws agentException
Creates an agent with the specified codebase, class name, and initialization. The newly created agent is owned by this agent manage.
send Message
public void sendMessage(AgentID aid, Message; msg)
Sends the specified message to the agent with the specified identifier.
dispatchAgent
public void dispatchAgent(Agent a, String dest)
Dispatches an agent to the specified destination.
retrieveAgent
public Agent retrieveAgent(AgentID aid)
Retrieves the agent with the specified agent identifier.
getAgent
public Agent getAgent(AgentID aid)
Gets the agent managed by this agent manager with the specified agent identifier.
getAgents

public Enumeration getAgents()
Gets all agents managed by this agent manager.
getDestination
public String getDestination(AgentID aid)
Gets the destination of the agent owned by this agent manager.
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Precarious Next Index
Class com.ibm.jma.agent.AgentException
java.lang.Object
. . . java.lang.Throwable
. . . java.lang.Exception
. . . com.ibm.jma.agent.AgentException
public class AgentException
extends Exception
Constructor Index
AgentException()
public AgentException(String)
Constructors
AgentException
public AgentException()
AgentException
public AgentException(String s)
All Packages Class Hierarchy This Package Previous Next Index

TABLE 4

All Packages Class Hierarchy Index
package com.ibm.jma.mail
Interface Index
MailQueue
PostOffice
Class Index
Mail
Mailbox
URL
All Packages Class Hierarchy This Package Previous Index
Interface com.ibm.jma.mail.MailQueue
public interface MailQueue
This interface is implemented by all mail queues.
See Also:
Mail
Method Index
close()
Closes this mail queue to disallow processing of mail.
dequeue()
Removes a mail from this mail queue.
dequeue(byte[])
Removes a mail with the specified correlation id from this mail queue.
dequeue(String)
Removes a mail with the specified content type from this mail queue.
enqueue(Mail)
Adds a mail to this mail queue.
getName()
Returns the name of this mail queue.
isEmpty()
Tests if this mail queue has no mail.
open()
Opens this mail queue for processing of mail.
size()
Returns the number of mail in this mail queue.
Methods
getName
public abstract string getName()
Returns the name of this mail queue.
enqueue
public abstract void enqueue(Mail mail)
Adds a mail to this mail queue.
Parameters:
mail—the mail to be added
dequeue
public abstract Mail dequeue()
Removes a mail from this mail queue.
Returns:
a mail
dequeue
public abstract Mail dequeue(String type)
Removes a mail with the specified content type from this mail queue.
Returns:
a mail with the specified content type
dequeue
public abstract Mail dequeue(byte corrId[])
Removes a mail with the specified correlation id from this mail queue.
Returns:
a mail with the specified correlation id
isEmpty
public abstract boolean isEmpty()
Tests if this mail queue has no mail.
size
public abstract int size()
Returns the number of mail in this mail queue.
open
public abstract void open()
Opens this mail queue for processing of mail.
close
public abstract void close()
Closes this mail queue to disallow processing of mail.
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Interface com.ibm.jma.mail.PostOffice
public interface PostOffice
The PostOffice interface is used to represent post office facility in Jamaica. It provides APIs for sending and receiving mails between post offices and mail boxes.
See Also:
PostOfficeRMI
Method Index
receiveMail(String)
This method receives mail from a mailbox with name "mbName"
receiveMail(String, byte[])
This method receives mail that has a correlation id "corrId" from the mail box of name "mbName"
receiveMail(String, byte[], long)
This method receives mail with correlation id "corrId" with timed wait of "waitTime" milliseconds from

mailbox "mbName" when it times out a null mail object is returned

receiveMail(String, long)

This method receives mail with timed wait of "waitTime" milliseconds from a mailbox "mbName".

receiveMail(String, String)

This method receives mail of "type" type from a mailbox with name "mbName"

receiveMail(String, String, long)

This method receives mail of type "type" with timed wait of "waitTime" milliseconds from mailbox "mbName" when it times out a null mail object is returned

sendMail(String, Mail)

This method sends a mail "mail" to a mailbox with name "mbName"

Methods

sendMail

public abstract byte[] sendMail(String mbName, Mail mail)

This method sends a mail "mail" to a mailbox with name "mbName"

Parameters:

mbName—Name of the mailbox to which the mail has to be sent

mail—The mail object that has to be sent

Returns:

A byte array that identifies the mail

receiveMail

public abstract Mail receiveMail(String mbName)

This method receives mail from a mailbox with name "mbName"

Parameters:

mbName—Name of the mailbox from which to receive mail

Returns:

The received mail object

receiveMail

public abstract Mail receiveMail(String mbName, String type)

This method receives mail of type "type" from a mailbox with name "mbName"

Parameters:

mbName—Name of the mailbox from which to receive mail

type—Type of the mail that is to be received

Returns:

The received mail object

receiveMail

public abstract Mail receiveMail(String mbName, byte corrId[])

This method receives mail that has a correlation id "corrId" from the mail box of name "mbName"

Parameters:

mbName—Name of the mailbox from which to receive mail

corrId—Correlation id of the mail that is to be received

Returns:

The received mail object

receiveMail

public abstract Mail receiveMail(String mbName, long waitTime)

This method receives mail with timed wait of "waitTime," milliseconds from a mailbox "mbName".

Parameters:

mbName—Name of the mailbox from which to receive

waitTime—The number of milliseconds to wait

receiveMail

public abstract Mail receiveMail(String mbName, String type, long waitTime)

This method receives mail of type "type" with tie wait of "waitTime" milliseconds from mailbox "mbName" when it times out a null mail object is returned

Parameters:

mbName—Name of the mail box from which to receive

type—Type of the mail to be received

waitTime—Number of milliseconds to wait

Returns:

The received mail object

receiveMail

public abstract Mail receiveMail(String mbName, byte corrId[], long waitTime)

This method receives mail with correlation id "corrId" with timed wait of "waitTime" milliseconds from mailbox "mbName" when it times out a null mail object is returned

Parameters:

mbName—Name of the mail box from which to receive

corrId—Correlation Id of the mail to be received

waitTime—Number of milliseconds to wait

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

Class com.ibm.jma.mail.Mail

java.lang.object

. . . com.ibm.jma.mail.Mail

public class Mail

extends Object

implements Serializable

A Mail object is used to transport typed content. Each mail has a type and a content. The format and semantics of the content depends on the content type.

constructor Index

Mail()

Mail(Mail)

Mail(String, Object)

Constructs a mail with the specified content type and content.

Methods Index

getContent()

Returns the content of this mail.

getCorrelationID()

Returns the correlation id.

getDestination()

Returns the destination of this mail.

getMailID()

Returns the mail id of this mail.

getPriority()

Returns the priority of this mail

getResponseDestination()

Returns the response destination of this mail.

getType()

Returns the content type of this mail.

setCorrelationId(byte[])

Sets the correlation id for the mail that this mail corresponds to.

setDestination(String)
Sets the destination of this mail

setMailId(byte[])
Sets the mail id of this mail.

setPriority(int)
Sets the priority of this mail.

setResponseDestination(String)
Sets the response destination of this mail.

toString()
Returns a string representing this mail.

Constructors

Mail
public Mail(String type, Object content)
Contructs a mail with the specified content type and content.

Parameters:
type—the content type of this mail
content—the content of this mail

Mail
protected Mail()

Mail
protected Mail(Mail m)

Methods

toString
public String toString()
Returns a string representing this mail.

Overrides:
toString in class Object

getType
public String getType()
Returns the content type of this mail.

getContent
public Object getContent()
Returns the content of this mail.

setMailId
public void setMailId(byte id[])
Sets the mail id of this mail.

getMailId
public byte[] getMailId()
Returns the mail id of this mail.

setCorrelationId
public void setCorrelationId(byte id[])
Sets the correlation id for the mail that this mail corresponds to. Required for a response mail.

getcorrelationId
public byte[] getCorrelationId()
Returns the correlation id.

setPriority
public void setPriority(int priority)
Sets the priority of this rail. The priority ranges from 1 to 10. The default is 5.

getPriority
public int getPriority()
Returns the priority of this mail

setDestination
public void setDestination(String dest)
Sets the destination of this mail

getDestination
public String getDestination()
Returns the destination of this mail.

setResponseDestination
public void setResponseDestination(String respDest)
Sets the response destination of this mail.

getResponseDestination
public String getResponseDestination()
Returns the response destination of this mail

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

Class com.ibm.jma.mail.Mailbox
java.lang.object
. . . com.ibm.jma.mail.Mailbox
public class Mailbox
extends Object

A Mailbox object is used to send (put) and receive (get) mail. A mailbox is virtual. To send a mail, one simply opens a mailbox with the name that represents the destination for the mail and puts the mail in the mailbox. To receive a mail, one again simply opens a mailbox with the name that represents the location for the mail and gets the mail from the mailbox.

See Also:
Mail

Constructor Index

Mailbox(String)
Constructs (opens) a mailbox with the specified name.

Method Index

getMail()
Gets (receives) a mail from this mailbox.

getMail(byte[])
Gets (receives) a mail with the specified correlation id from this mailbox.

(byteMail[], long)
Gets (receives) a mail with the specified correlation id from this mailbox , waiting if the mail arrives within the specified wait time.

getMail(long)
Gets (receives) a mail from this mailbox, waiting if the mail arrives with the specified wait time.

getMail(String)
Gets (receives) a mail with the specified content type from this mailbox.

getMail(String, long)
Gets (receives) a mail with the specified content type from this mailbox, waiting if the mail arrives within the specified wait time.

getName()
Returns the name of this mailbox.

getMail(Mail)
Puts a mail in this mailbox (i.e., sends a mail to this mailbox).

Constructors

Mailbox
public Mailbox(String name)
Constructs (opens) a mailbox with the specified name.

Methods

getName
public String getName()
Returns the name of this mailbox.

putMail
public byte[] putMail(Mail mail)
Puts a mail in this mailbox (i.e., sends a mail to this mailbox).

Parameters:
mail—the mail to be sent
Returns:
the mail id of the mail sent
getMail
public Mail getMail()
Gets (receives) a mail from this mailbox. If there is no mail, it returns null.
Returns:
a mail
getMail
public Mail getMail(long waitTime)
Gets (receives) a mail from this mailbox, waiting if the mail arrives within the specified wait time. If the wait time is set to −1, it waits forever until the mail arrives.
Returns:
a mail
getMail
public Mail getMail(String type)
Gets (receives) a mail with the specified content type from this mailbox. If there is no such mail, it returns null.
Returns:
a mail with the specified content type
getMail
public Mail getMail(String type, long waitTime)
Gets (receives) a mail with the specified content type from this mailbox, waiting if the mail arrives within the specified wait time. If the wait time is set to −1, it waits forever until the mail arrives.
Parameters:
type—the content type of the mail to be received
waitTime—the time (in msecs) to wait for the mail to arrive
Returns:
a mail with the specified content type
getMail
public Mail getMail(byte corrId[])
Gets (receives) a mail with the specified correlation id from this mailbox. If there is no such mail, it returns null.
Returns:
a mail with the specified correlation id
getMail
public Mail getMail(byte corrId[], long waitTime)
Gets (receives) a mail with the specified correlation id from this mailbox waiting if the mail arrives within the specified wait time. If the wait time is set to −1, it waits forever until the mail arrives.
Parameters:
corrId—the correlation id of the mail to be received
waitTime—the time (in msecs) to wait for the mail to arrive
Returns:
a mail with the specified correlation id
All Packages Class Hierarchy This Package Previous Next Index
All Packages class Hierarchy This Package Previous Next Index
Class com.ibm.jma.mail.URL
java.lang.Object
. . . com.ibm.ima.mail.URL
public final class URL
extends Object
implements Serializable
Constructor Index
URL(String)
URL(String, String, int, String)
Methods Index
equals(URL)
getFile()
getHost()
getName()
getPort()
getProtocol()
getRef()
setRef(String)
toString()
Constructurs
URL
public URL(String spec) throws MalformedURLException
URL
public URL(String protocol, String hostName, int portNumber, String filename) throws MalformedURLException
Methods
getProtocol
public String getProtocol()
getHost
public String getHost()
getPort
public int getPort()
getFile
public String getFile()
setRef
public void setRef(String ref)
getRef
public String getRef()
equals
public boolean equals(TR obj)
toString
public String toString()
Overrides:
toString in class Object
getName
public String getName()
All Packages Class Hierarchy This Package Previous Next Index

TABLE 5

All Packages Class Hierarchy Index
package com.ibm.jma.mail.impl
Interface Index
PostOfficeRemoteRMI
Class Index
MemMailQueue
POServre
PostOfficeRMI
SQL MailQueue
Exception Index
NoSuchMailException
NotImplementedException
All Packages Class Hierarchy This Package Previous Next Index

Interface
com.ibm.jma.mail.impl.PostOfficeRemoteRMI
public interface PostOfficeRemoteRMI
extends Remote
Methods Index
deliverMail(String, Mail)
a deliverMail(String, Mail[])
retrieveMail(String, byte[], long)
retrieveMail(String, long)
retrieveMail(String, String, long)
Methods
getName
public abstract String getName() throws RemoteException
deliverMail
public abstract void deliverMail(String mbId, Mail mail) throws RemoteException
deliverMail
public abstract void deliverMail(String mbId, Mail mails[]) throws RemoteException
retrieveMail
public abstract Mail retrieveMail(string mbId, long waitTime) throws RemoteException
retrieveMail
public abstract Mail retrieveMail(String mbId, String type, long waitTime) throws RemoteException
retrieveMail
public abstract Mail retrieveMail(String mbId, byte corrId[], long waitTime) throws RemoteException
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.mail.impl.MemMailQueue
java.lang.object
. . . com.ibm.jma.mail.impl.MemMailQueue
public class MemMailQueue
extends Object
implements Mail Queue
Constructor Index
MemMailQueue()
Method Index
close()
dequeue()
dequeue(byte[])
dequeue(String)
enqueue(Mail)
getName()
isEmpty()
open()
size()
Constructors
MemMailQueue
public MemMailQueue()
Methods
getName
public String getName()
isEmpty
public synchronized boolean isEmpty()
size
public int size()
enqueue
public synchronized void enqueue(Mail mail)
dequeue
public synchronized Mail dequeue()
dequeue
public synchronized Mail dequeue(byte corrId[])
dequeue
public synchronized Mail dequeue(string mailtype)
open
public void open()
close
public void close()
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class com.ibmjma.mail.impl.POServer
java.lang.object
. . . java.awt.Component
. . . java.awt.Container
. . . java.awt.Window
. . . java.awt.Frame
. . . com.ibm.jma.mail.impl.POServer
public class POServer
extends Frame
implements ItemListener
Constructor Index
POServer(String)
Method Index
action(Event, Object)
ItemStateChanged(ItemEvent)
main(String[])
minimumSize()
preferredSize()
Constructors
POServer
public POServer(String n)
Methods
preferredSize
public Dimension preferredSize()
Overrides:
preferredSize in class Container
minimumSize
public Dimension minimumsize()
Overrides:
minimumSize in class Container
itemStateChanged
public void itemStateChanged(ItemEvent evt)
action
public boolean action(Event evt, Object arg)
Overrides:
action in class Component
main
public static void main(String args[])
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.mail.impl.PostOfficeRMI
java.lang-Object
. . . java.rmi.server.RemoteObject

. . . java.rmi.server.RemoteServer
. . . java.rmi.server.UnicastRemoteObject
. . . com.ibm.jma.mail.impl.PostofficeRMI
public class PostOfficeRMI
extends UnicastRemoteObject
implements PostOfficeRemoteRM, PostOffice
Method Index
deliverMail(String, Mail)
deliverMail(String, Mail[])
getLocalPostOffice()
getName()
receiveMail(String)
receiveMail(String, byte[])
receiveMail(String, byte[], long)
receive Mail(String, long)
receiveMail(String, String)
receiveMail(String, String, long)
retrieveMail(String, byte[], long)
retrieveMail(String, long)
retrieveMail(String, String, long)
sendMail(String, Mail)
Methods
getLocalPostOffice
public static synchronized PostOfficeRMI getLocalPostOffice() throws Rer
getName
public String getName() throws RemoteException
deliverMail
public void deliverMail(String mbId, Mail mail) throws RemoteException
deliverMail
public void deliverMail(String mbId, Mail maillist[]) throws RemoteException
retrieveMail
public Mail retrieveMail(String mbId, long waitTime) throws RemoteException, Illegal
retrieveMail
public Mail retrieveMail(String mbId, String type, long waitTime) throws RemoteException, Illegal
retrieveMail
public Mail retrieveMail(String mbId, byte corrId[], long waitTime) throws RemoteException, Illegal
sendMail
public byte[] sendMail(String mbName, Mail mail)
receiveMail
public Mail receiveMail(String mbName)
receiveMail
public Mail receiveMail(String mbName, long waitTime)
receiveMail
public Mail receiveMail(String mbName, String type)
receiveMail
public Mail receiveMail(String mbName, String type, long waitTime)
receiveMail
public Mail receiveMail(String mbName, byte corrId [])
receiveMail
public Mail receiveMail(String mbName, byte corrId[], long waitTime)
All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.mail.impl.SQLMailQueue
java.lang.object
. . . com.ibm.jma.mail.impl.SQLMailQueue
public class SQLMailQueue
Constructor Index
SQL MailQueue(String)
Method Index
close()
dequeue()
dequeue(byte[])
dequeue(String)
enqueue(Mail)
getName()
isEmpty()
open()
size()
Constructors
SQL MailQueue
public SQLMailQueue(String name)
Methods
getName
public String getName()
enqueue
public synchronized void enqueue(Mail mail)
dequeue
public synchronized Mail dequeue()
dequeue
public synchronized Mail dequeue(byte corrId[])
dequeue
public synchronized Mail dequeue(String mailtype)
isEmpty
public synchronized boolean isEmpty()
size
public int size()
open
public void open()
close
public void close()
All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.mail.impl.NoSuchMailException
java.lang.Object
java.lang.Throwable
java.lang.Exception
com.ibm.jma.mail.impl.NoSuchMailException
public final class NoSuchMailException
extends Exception
Constructor Index
NoSuchMailException()
NOSuchMailException(byte[])
Constructors
NoSuchMailException
public NoSuchMailException()
NoSuchMailException
public NoSuchMailException (byte corrId[])
All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.mail.impl.NotImplementedException
java.lang.Object
. . . java.lang.Throwable
. . . java.lang.Exception
. . . com.ibm.jma.mail.impl.NotImplementedException
public final class NotImplementedException
extends Exception
constructor Index
NotImplementedException()
public Not ImplementedException(Object)
Constructors
NotImplementedException
public NotImplementedException()
NotImplementedException
public NotImplementedException(Object obj)
All Packages Class Hierarchy This Package Previous Next Index

TABLE 6

All Packages Class Hierarchy Index
package com.ibm.jma.message
Class Index
ATRequestMessage
ATResponseMessage
KOMLMessage
Message
RequestMessage
ResponseMessage
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.message.ATRequestMessage
java.lang.Object
. . . com.ibm.jma.message.Message
. . . com.ibm.jma.message.RequestMessage
. . . com.ibm.jma.message.ATRequestMessage
public class ATRequestMessage
extends RequestMessage
An ATRequestMessage object is used to send agent transfer requests.
See Also:
RequestMessage
Constructor Index
ATRequestMessage
Constructors
ATRequestMessage
public ATRequestMessage()
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.message.ATResponseMessage
java.lang.object
. . . com.ibm.jma.message.Message
. . . com.ibm.jma.message.ResponseMessage
. . . com.ibm.jma.message.ATResponseMessage
public class ATResponseMessage
extends ResponseMessage
A ATResponseMessage object is used to send back the result of an agent request message.
See Also:
RequestMessage
Constructor Index
ATResponseMessage(Object)
Constructs an agent transfer response message with the specified result.
Method Index
getParameters()
Gets the parameters of the result.
setParameters(Hashtable)
Sets the parameters of the result
Constructors
ATResponseMessage
public ATResponseMessage(Object result)
Constructs an agent transfer response message with the specified result.
Methods
setParameters
public void setparameters(Hashtable params)
Sets the parameters of the result
getParameters
public Hashtable getparameters()
Gets the parameters of the result.
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.message.KQMLMessage
Java.lang.object
. . . com.ibm.jma.message.Message
. . . com.ibm.jma.message.KQMLMessage
public class KQMLMessage
extends Message
A KQMLMessage object is used to send messages following the KQML format and protocol.
Constructor Index
KQMLMessage()
Constructs a KQML message.
Method Index
getContent()
Gets the content
getInReplyTo()
Gets the identifier that this message is replying to.
GetInReplyWith()
Gets the identifier that this message is to be relied with.
getLanguage()
Gets the content language.
getOntology()
Gets the content ontology.
getPerformative()
Gets the performative.
setContent(Object)
Sets the content.
setInReplyTo(String)
Sets the identifier that this message is replying to.
setLanguage(String)
Sets the content language.
setOntology(String)
Sets the content ontology.
setPerformative(String)

Sets the performative.

setReplyWith(String)

Sets the identifier that this message is to be relied with.

Constructors

KQMLMessage

public KQMLMessage()

Constructs a KQML message.

Methods

setperformative

public void setPerformative(String perf)

Sets the performative.

getPerformative

public String getperformative()

Gets the performative.

setInReplyTo

public void setInReplyTo(String int)

Sets the identifier that this message is replying to.

getInReplyTo

public String getInReplyTo()

Gets the identifier that this message is replying to.

setReplyWith

public void setReplyWith(String rw)

Sets the identifier that this message is to be relied with.

getInReplyWith

public String getInReplyWith()

Gets the identifier that this message is to be relied with.

setLanguage

public void setLanguage(String lang)

Sets the content language.

getLanguage

public String getLanguage()

Gets the content language.

setOntology

public void setOntology(String onto)

Sets the content ontology.

getOntology

public String getOntology()

Gets the content ontology.

setContent

public void setContent(Object content)

Sets the content.

getContent

public Object getContent()

Gets the content.

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

Class com.ibm.jma.message.Message

java.lang.object

. . . com.ibm.jma.message.Message

public abstract class Message

extends Object

implements Serializable

This is the abstract, root class of all types of messages. A Message object is used to encapsulate the information which is to be sent from a sender to a receiver. Its class name represents its type, which determines its format and semantics. Each type of messages may be associated with a corresponding type of message handlers which are designed to handle the messages.

See Also:

MessageHandler

Field Index

mailID

Constructor Index

Message()

Method Index

getReceiver()

Gets the receiver mailbox of this message.

getReceiverMB()

Gets the sender of this message.

getSender()

Gets the sender mailbox of this message.

getSenderMB()

Gets the sender mailbox of this message

send()

Sends this message to the receiver.

setReceiver(String)

Sets the receiver of this message.

setSender(String)

Sets the sender of this message.

Fields

mailId

protected byte mailId[]

Constructors

Message

public Message()

setsender

public void setSender(String sender)

Sets the sender of this message.

getSender

public String getSender()

Gets the sender of this message.

getSenderMB

public String getSenderMB()

Gets the sender mailbox of this message.

setReceiver

public void setReceiver(String receiver)

Sets the receiver of this message.

getReceiver

public String getReceiver()

Gets the receiver of this message.

getReceiverMB

public String getReceiverMB()

Gets the receiver mailbox of this message.

send

public void send()

Sends this message to the receiver. This operation uses the Jamaica Mail Facility (com.ibm.jma.mail) to accomplish its task.

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

Class

com.ibm.jma.message.RequestMessage

java.lang.Object

com.ibm.jma.message.Message

. . . com.ibm.jma.message.RequestMessage

public class RequestMessage

extends Message

A RequestMessage object is used to send messages which return a result. A request message can be sent in three different modes: one-way (asynchronous, discarding the result), synchronous (blocking until the result arrives), or

deferred (obtaining the result at a later time). The result of a request message is contained in a response message.

See Also:

Constructs a request message.

Constructor Index

RequestMessage()
Constructs a request message

Method Index

checkResult()
Tests if the result has arrived.

getMessageID()
Gets the message id.

getOperation()
Gets the operation to be performed by the receiver of this message.

getParameters()
Gets the parameters of the operation.

getResult()
Gets the result.

getResult()
Gets the result if it arrives within the specified wait time.

isClone()
Tests if this message is a clone.

isOneway()
Tests if the messaging mode is one-way.

isRead()
Tests if the result has been read.

isSend()
Tests if this message has been sent

send()
Sends this message.

setClone()
Indicates that this message is a clone.

setMessageID(byte[])
Sets the message id.

setOneway()
Sets the messaging mode to one-way.

setOperation(String)
Sets the operation to be performed by the receiver of this message.

setParameters(Hashtable)
Sets the parameters of the operation.

Constructors

RequestMessage
public RequestMessage()
Constructs a request message.

Methods

setMessageId
protected void setMessageID(byte msgId[])
Sets the message id.

getMessageId
public byte[] getMessageId()
Gets the message id.

setOneway
public void setoneway()
Sets the messaging mode to one-way. The result, if any, will be discarded.

isOneway
public boolean isoneway()
Tests if the messaging mode is one-way.

setOperation
public void setoperation(String oper)
Sets the operation to be performed by the receiver of this message.

getOperation
public String getoperation()
Gets the operation to be performed by the receiver of this message.

setparameters
public void setParameters(Hashtable params)
Sets the parameters of the operation.

getParameters
public Hashtable getparameters()
Gets the parameters of the operation.

send
public void send()
Sends this message.

Overrides:
send in class Message

isSent
public boolean isSent()
Tests if this message has been sent

setClone
public void setclone()
Indicates that this message is a clone.

isClone
public boolean isclone()
Tests if this message is a clone.

checkResult
public boolean checkResult()
Tests if the result has arrived.

getResult
public Object getResult()
Gets the result. If the result has not arrived, returns null.

getResult
public Object getResult(long waitTime)
Gets the result if it arrives within the specified wait time. If the wait time is set to −1, waits forever until the result arrives.

Returns:
the result

isRead
public boolean isread()
Tests if the result has been read.

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

Class
com.ibm.jma.message.ResponseMessage
java.lang.Object
com.ibm.jma.message.Message
com.ibm.jma.message.ResponseMessage
public class ResponseMessage
extends Message

A ResponseMessage object is used to send back the result of a request message.

See Also:
RequestMessage

Constructor Index

ResponseMessage(Object)
Constructs a response message with the specified result.

Method Index

getCorrelationId()

Gets the correlation id.

getResult()

Gets the result that this message contains.

setCorrelationId(byte[])

Sets the correlation id for the corresponding request message.

Constructors

ResponseMessage

public ResponseMessage(Object result)

Constructs a response message with the specified result.

Methods

setCorrelationId

public void setCorrelationId(byte corrId[])

Sets the correlation id for the corresponding request message.

getCorrelationId

public byte[] getCorrelationId()

Gets the correlation id.

getResult

public Object getResult()

Gets the result that this message contains.

All Packages Class Hierarchy This Package Previous Next Index

TABLE 7

All Packages Class Hierarchy Index

package com.ibm.jma.message.handler

Interface Index

MessageHandler

Class Index

ATRequestMessageHandler

ATResponseMessageHandler

KQMLMessageHandler

KSQLMessageHandler

RequestMessageHandler

ResponseMessagHandler

All Package Class Hierarchy This Package Previous Next Index

Interface

com.ibm.jma.message.handler.MessageHandler

public interface MessageHandler

This interface is implemented by all message handlers. Each type of message handlers is designed to handle an associated type(s) of messages.

See Also:

Message

Method Index

getMessageTypes()

Gets the message types which can be handled by this message handler.

handleMessage(Message)

Handles the specified message.

Methods

handleMessage

public abstract boolean handleMessage(Message msg)

Handles the specified message.

Returns:

true if the specified message can be handled.

getMessageType

public abstract String getMessageTypes()

Gets the message types which can be handled by this message handler.

Returns:

message types concatenated in a string and separated by spaces

All Packages Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

Class

com.ibm.jma.message.handler.ATRequestMessage Handler

java.lang.object

. . . com.ibm.jma.message.handler.ATRequestMessage Handler

public class ATRequestMessageHandler

extends RequestMessageHandler

The handler for handling agent transfer request messages.

See Also:

Message

Constructor Index

ATRequestMessageHandler()

Method Index

getMessageTypes()

Gets the message types which can be handled by this message handler.

handleMessage(Message)

Handles the specified message.

Constructors

ATRequestMessageHandler

public ATRequestMessageHandler()

handleMessage

public boolean handleMessage(Message msg)

Handles the specified message.

Returns:

true if the specified message can be handled.

Overrides:

handleMessage in class RequestMessageHandler

getMessageTypes

public String getMessageTypes()

Gets the message types which can be handled by this message handler.

Returns:

message types concatenated in a string and separated by spaces

Overrides:

getMessageTypes in class RequestMessageHandler

All Package Class Hierarchy This Package Previous Next Index

All Packages Class Hierarchy This Package Previous Next Index

Class

com.ibm.j ma. message.handler.ATResponseMessageHandler

java.lang.object

. . . com.ibm.jma.message.handler.ResponseMessageHandler

. . . com.ibm.jma.message.handler.ATResponseMessageHandler

public class ATResponseMessageHandler

extends ResponseMessagHandler

The handler for handling agent transfer response messages.

See Also:

Message

Constructor Index

ATResponseMessageHandler()
Method Index
getMessageTypes()
Gets the message can be handled by this message handler.
handleMessage(Message)
Handles the specified message.
Constructors
ATResponseMessageHandler
public ATResponseMessageHandler()
Methods
handleMessage
public boolean handleMessage(Message msg)
Handles the specified message.
Returns:
true if the specified message can be handled.
Overrides:
handleMessage in class ResponseMessageHandler
getMessageTypes
public String getMessageTypes()
Gets the message types which can be handled by this message handler.
Returns:
message types concatenated in a string and separated by spaces
Overrides:
getMessageTypes in class ResponseMessageHandler
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.message.handler.KQMLMessageHandler
java.lang.object
. . . com.ibm.jma.message.handler.KQMLMessageHandler
public class KQMLMessageHandler
extends Object
implements MessageHandler
The handler for handling KQML messages.
See Also:
KQMLMessage, KSQLMessageHandler
Constructor Index
KQMLMessageHandler()
Method Index
getMessageTypes()
Gets the message types which can be handled by this message handler.
handleMessage(Message)
Handles the specified message.
Constructors
KQMLMessageHandler
public KQMLMessageHandler()
Methods
handleMessage
public boolean handleMessage(Message msg)
Handles the specified message.
Returns:
true if the specified message can be handled.
getMessageTypes
public String getMessageTypes()
Gets the message types which can be handled by this message handler.
Returns:
message types concatenated in a string and separated by spaces
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.message.handler.KSQLMessageHandler
java.lang.Object
. . . com.ibm.jma.message.handler.RSQLMessageHandler
public class KSQLMessageHandler
extends KQMLMessageHandler
The handler for handling KQML messages whose content language is SQL.
See Also:
KQMLMessage, KQMLMessageHandler
Constructor Index
KSQLMessageHandler()
Method Index
handleMessage(Message)
Handles the specified message.
Constructors
KSQLMessageHandler
public KSQLMessageHandler()
Methods
handleMessage
public boolean handleMessage(Message msg)
Handles the specified message.
Returns:
true if the specified message can be handled.
Overrides:
handleMessage in class KQMLMessageHandler
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.message.handler.RequestMessageHandler
java.lang.Object
. . . com.ibm.jma.message.handler.RequestMessageHandler
public class RequestMessageHandler
extends Object
implements MessageHandler
The handler for handling request messages.
See Also:
RequestMessage
Constructor Index
RequestMessageHandler()
Method Index
getMessageTypes()
Gets the message types which can be handled by this message handler.
handleMessage(Message)
Handles the specified message.
Constructors
RequestMessageHandler
public RequestMessageHandler()
Methods
handleMessage
public boolean handleMessage(Message reqc))
Handles the specified message.
Returns:
true if the specified message can be handled.

getMessageTypes
public String getMessageTypes()
Gets the message types which can be handled by this message handler.
Returns:
message types concatenated in a string and separated by spaces
All Packages Class Hierarchy This Package Previous Next Index
All Packages Class Hierarchy This Package Previous Next Index
Class
com.ibm.jma.message.handler.ResponseMessageHandler
java.lang.object
. . . com.ibm.jma.message.handler.ResponseMessageHandler
public class ResponseMessageHandler
extends Object
implements MessageHandler
The handler for handling response messages.
See Also:
Message
Constructor Index
ResponseMessagHandler()
Method Index
getMessageTypes()
Gets the message types which can be handled by this message handler.
Handles the specified message.
ResponseMessageHandler
public ResponseMessageHandler()
handleMessage
public boolean handleMessage(Message msg)
Handles the specified message.
Returns:
true if the specified message can be handled.
getMessageTypes
public String getMessageTypes()
Gets the message types which can be handled by this message handler.
Returns:
message types concatenated in a string and separated by spaces
All Packages Class Hierarchy This Package Previous Next Index

TABLE 8

All Packages Index
Class Hierarchy
class java.lang.Object
class com.ibm.jma.agent.Agent (implements java.lang.Cloneable, com.ibm.jma.message.handler.MessageHandler)
class com.ibm.jma.agent.AgentManager (implements java.io.Serializable)
class com.ibm.jma.agent.AgentManager (implements com.ibm.jma.message.handler.MessageHandler)
class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
class java.awt.Container
class java.awt.Window
class java.awt.Frame (implements java.awt.MenuContainer)
class com.ibm.jma.mail.impl.POServer (implements java.awt.event.ItemListener)
class com.ibm.jma.message.handler.KQMLMessageHandler (implements com.ibm.jma.message.handler.MessageHandler)
class com.ibm.jma.message.handler.KSQLMessageHandler
class com.ibm.jma.mail.Mail (implements java.io.Serializable)
interface com.ibm.jma.mail.MailQueue
class com.ibm.jma.mail.Mailbox
class com.ibm.jma.mail.impl.MemMailQueue (implements com.ibm.jma.mail.MailQueue)
class com.ibm.jma.message.Message (implements java.io.Serializable)
class com.ibm.jma.message.KQMLMessage
class com.ibm.jma.message.RequestMessage
class com.ibm.jma.message.ATRequestMessage
class com.ibm.jma.message.ResponseMessage
class com.ibm.jma.message.ATResponseMessage
interface com.ibm.jma.message.handler.MessageHandler
interface com.ibm.jma.mail.PostOffice
interface com.ibm.jma.mail.impl.PostOfficeRemoteRMI (extends java.rmi.Remote)
class java.rmi.server.RemoteObject (implements java.rmi.Remote, java.io.Serializable)
class java.rmi.server.RemoteServer
class javaimi.server.UnicastRemoteObject
class com.ibm.jma.mail.impl.PostOfficeRMI (implements com.ibm.jma.mail.impl.PostOfficeRemoteRMI, com.ibm.jma.mail.PostOffice)
class com.ibm.jma.message.handler.RequestMessageHandler (implements com.ibm.jma.message.handler.MessageHandler)
class com.ibm.jma.message.handler.ATRequestMessageHandler
class com.ibm.jma.message.handler.ResponseMessageHandler (implements com.ibm.jma.message.handier.MessageHandler)
class com.ibm.jma.message.handler.ATResponseMessageHandler
class com.ibm.jma.mail.impl.SOLMailQueue (implements com.ibm.jma.mail.MailQueue)
class java.lang.Throwable (implements java.io.Serializable)
class java.lang.Exception
class com.ibm.jma.agent.AgentException
class com.ibm.jma.mail.impl.NoSuchMailException
class com.ibm.jma.mail.impl.NoImplementedException
class com.ibm.jma.mail.URL (implements java.io.Serializable)

TABLE 9

All Packages Class Hierarchy A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Index of all Fields and Methods
A
action(Event, Object). Method in class com.ibm.jma.mail.impl.POServer
Agent(). Constructor for class com.ibm.jma.agent.Agent
AgentException(). Constructor for class com.ibm.jma.agent.AgentException
AgentException(String). Constructor for class com.ibm.jma.agent.AgentException
AgentID(byte[]). Constructor for class com.ibm.jma.agent.AgentID

Constructs an agent identifier from the specified byte array representation.

AgentID(String). Constructor for class com.ibm.jma.agent.AgentID
Constructs an agent identifier from the specified string representation.

ATRequestMessage(). Constructor for class com.ibm.jma.message.ATRequestMessage

ATRequestMessageHandler(). Constructor for class com.ibm.jma.message.handler. ATRequestMessageHandler

ATResponseMessage(Object). Constructor for class com.ibm.jma.message.ATResponseMessage
Constructs an agent transfer response message with the specified result.

ATResponseMessageHandler(). Constructor for class com.ibm.jma.message.handler.ATResponseMessageHandler

C

checkResult(). Method in class com.ibm.jma.message.RequestMessage
Tests if the result has arrived.

close(). Method in interface com.ibm.jma.mail.MailQueue
Closes this mail queue to disallow processing of mail.

close(). Method in class com.ibm.jma.mail.impl.MemMailQueue

close(). Method in class com.ibm.jma.mail.impl.SQLMailQueue

createAgent(URL, String, Object). Method in class com.ibm.jma.agent.AgentManager
Creates an agent with the specified codebase, class name, and initialization.

D

DEFAULT_AM_NAME. Static variable in class com.ibm.jma.agent.AgentManager

DEFAULT_PORT_NUMBER. Static variable in class com.ibm.jma.agent.AgentManager

DEFAULT_PROTOCOL. Static variable in class com.ibm.jma.agent.AgentManager

deliverMail(String, Mail). Method in interface com.ibm.jma.mail.impl.PostOfficeRemoteRMI

deliverMail(String, Mail). Method in class com.ibm.jma.mail.impl.PostOfficeRMI

deliverMail (String, Mail[]). Method in interface com.ibm.jma.mail.impl.PostOfficeRemoteRMI

deliverMail(String, Mail[]). Method in class com.ibm.jma.mail.impl.PostOfficeRMI

dequeue(). Method in interface com.ibm.jma.mail.MailQueue
Removes a mail from this mail queue.

dequeue(). Method in class com.ibm.jma.mail.impl.MailQueue

dequeue(). Method in class com.ibm.jma.mail.impl.MailQueue

dequeue(byte[]). Method in interface com.ibm.jma.mail.MailQueue
Removes a mail with the specified correlation id from this mail queue.

dequeue(byte[]). Method in class com.ibm.jma.mail.impl.MemMailQuene

dequeue(byte[]). Method in class com.ibm.jma.mail.impl.SQLMailQueue

dequeue(String). Method in interface com.ibm.jma.mail.MailQueue
Removes a mail with the specified content type from this mail queue.

dequeue(String). Method in class com.ibm.jma.mail.impl.MemMailQueue

dequeue(String). Method in class com.ibm.jma.mail.impl.SQLMailQueue

dispatchAgent(Agent, String). Method in class com.ibm.jma.agent.AgentManager
Dispatches an agent to the specified destination.

E

enqueue(Mail). Method in interface com.ibm.jma.mail.MailQueue
Adds a mail to this mail queue.

enqueue(Mail). Method in class com.ibm.jma.mail.impl.MemMailQueue

enqueue(Mail). Method in class com.ibm.jma.mail.impl.SQLMailQueue

equals(Object). Method in class com.ibm.jma.agent.AgentID
Test if the specified object is an agent identifier and is equal to this agent identifier.

equals(URL). Method in class com.ibm.jma.mail.URL

G

getAgent(AgentID). Method in class com.ibm.jma.agent.AgentManager
Gets the agent managed by this agent manager with the specified agent identifier.

getAgents(). Method in class com.ibm.jma.agent.AgentManager
Gets all agents managed by this agent manager.

getContent(). Method in class com.ibm.jma.message.KQMLMessage
Gets the content.

getContent(). Method in class com.ibm.jma.mail.Mail
Returns the content of this mail.

getCorrelationID(). Method in class com.ibm.jma.mail.Mail
Returns the correlation id.

getcorrelationId() Method in class com.ibm.jma.message.ResponseMessage
Gets the correlation id.

getDestination(). Method in class com.ibm.jma.mail.Mail
Returns the destination of this mail.

getDestination(AgentID). Method in class com.ibm.jma.agent.AgentManager
Gets the destination of the agent owned by this agent manager.

getFile(). Method in class com.ibm.jma.mail.URL

getHost(). Method in class com.ibm.jma.mail.URL

getID(). Method in class com.ibm.jma.agent.Agent
Gets the identifier of this agent.

getID(). Method in class com.ibm.jma.agent.AgentID
Gets the byte array representation of this agent identifier.

getInReplyTo.() Method in class com.ibm.jma.message.KQMLMessage
Gets the identifier that this message is replying to.

getInReplyWith(). Method in class com.ibm.jma.message.KQMLMessage
Gets the identifier that this message is to be relied with.

getLanguage(). Method in class com.ibm.jma.message.KQMLMessage