receiver already has an instance for the subagent (it merely did not have the transmitter's id for it), or this is the first time the receiver has heard of this subagent and a new subagent instance is created.

[0123] If the receiver has no local entry for the instance, its next step depends on the class of the instance. If it is a subagent the receiver requests the global identifier as above. If it is an interned string, then the receiver asks the sender for the characters in the string, and either uses or creates a local copy. In all other cases the receiver can create a remote reference to the instance without any further communication.

(4) Delayed Messages and Pending Instances

[0124] As explained above, a message may be received which contains references to an unknown subagent or interned string. Such messages are preferably delayed until the relevant information arrives from the sender. Other messages that refer to the same unknown instance may arrive after a request for information has been sent and before the reply is received. These messages must also be delayed.

[0125] Information about received-but-unknown subagents and interned strings are stored in the "pending" vector in subagent instances. If a uid is not found in the decode vector it is looked up in the pending vector. If found there, a request for the instance's data has already been sent, but the current message must be delayed until the information arrives.

(i) Garbage Collection

[0126] Since objects within an agent are distributed among a collection of machines, a global, asynchronous garbage collection strategy is preferable. A scheme of distributed reference counts is preferably used to allow the identification of instances whose remote references have been garbage collected. Each global id contains a non-zero reference count. Sending an instance to another subagent requires sending one or more reference counts along with the three ids described above. These reference counts are added to those in the global id on the receiving subagent.

[0127] If an instance in a message has a global id whose reference count is one, the sending subagent must delay sending the message. It cannot send the instance without a reference count and it must keep the one that it has. Additional reference counts are requested from the subagent that currently contains the instance. Once they arrive, the message can be sent along with some of the newly arrived reference counts. When a global id is no longer referenced by a subagent, its reference counts are sent back to the subagent containing the instance. Once that subagent has received all extant counts for the instance, the instance can be reclaimed by the agent's local garbage collector.

Exemplary Uses For The Invention

[0128] As will be readily understood by one or ordinary skill in the art, the distributed agent system of the present invention clearly has wide applicability in the field of distributed computing, and can be implemented on a wide spectrum of network communication systems from low-bandwidth, high latency communication over modems to high-bandwidth, low-latency communications such as found in clusters of high performance computers. As particular examples of such utility, the present invention offers effective support for network-centric application in which mobility is important. Such applications may include mobile software assistants capable of automatically retrieving and updating data on a corporate intranet, and adaptable query engines that execute queries and migrate database state among machines in a network to optimize availability and bandwidth. In addition, distributed applications which require high performance, such as data mining, warehousing, and search applications, will also benefit from use of the present invention. The foregoing examples are to be understood as being merely exemplary and merely serve to illustrate but a few of the many and varied possible uses for the present invention.

[0129] While there has been described and illustrated herein a distributed agent system which provides an object-based encapsulation model (an agent) which allows the processes and state of the agent to be distributed over multiple potentially heterogeneous machines, enables transparent access of data resident on another machine, and allows easy and efficient process migration, in whole or in part, among distinct machines, it will be apparent to those skilled in the art that further variations and modifications are possible without deviating from the broad teachings of the invention.

Claims

1. A distributed software system for use with a plurality of computer machines connected as a network, the system comprising:

   a plurality of bases, each base providing a local address space and computer resources on one of a plurality of computer machines;

   at least one agent comprising a protection domain, wherein the protection domain of the at least one agent resides on at least one of the plurality of bases;

   a plurality of objects contained within the protection domain of the at least one agent, a first object residing on a first base of the plurality of bases and a second object residing on a second base of the plurality of bases, wherein the first object on the first base may access the second object on the second base without