shared data found on the same subagent may be accessed with local references.

[0045] On the other hand, communication between agents is performed by using a remote reference if the object implements a Remote interface, or otherwise by copying. Arrow C refers to an object 52c having a Remote interface in another agent, so this reference is valid, while arrow D refers to an object 54b without a Remote interface in another agent, so this reference would be invalid. In order to avoid creating this invalidity, whenever an object without a Remote interface appears as an argument or a return value of a procedure call, a copy of the object should be passed. Communication between agents is described in more detail below.

Object Space

[0046] Conceptually, tasks access data within an agent through a global object space that defines a mapping between a remote reference to an object (e.g. the name of the object) and the object's physical location on a machine in the system.

[0047] FIG. 3 shows a conceptual, diagrammatic view of the operation of an object space. A single agent, Agent, spans two bases, Base1 and Base2, on two machines, Machine1 and Machine2. When an object 55 is created in the Agent on Base1 and Machine1, its identity and location are recorded in Agent's object space 70, together with a symbolic (or remote) reference 55' to object 55. Subsequent references to object 55 may be made using symbolic reference 55', which are handled by the system by querying the object space 70 and resolving the symbolic reference 55' to the appropriate physical location for object 55. Thus, object space 70 enables transparent access to elements in Agent which spans across multiple physical machines. In particular, references in Base2 on Machine2 to object 55 on Machine1 may be made using the symbolic reference 55'. Thus, references to data (regardless of whether they are local to a base or found on another base) do not require knowledge of the physical location where the data resides.

[0048] In practice, the use of a globally-accessible physical object space (such as shared memory) to mediate all references to data within an agent is possible but may be prohibitively expensive. A more efficient representation of the object space used by the present invention uses global identifiers as the primary addressing technique for object spaces and will be discussed below (see subsection entitled "Global Id").

Agent and Object Migration

[0049] A particularly useful feature of the present invention is program mobility. The distributed agent system of the present invention incorporates several user-level migration methods for agents and objects, and one system-level migration method for threads. Migration is important to the invention since it is an important means by which mobility is realized. Unlike other agent systems, the present invention, as a consequence of its distributed agent metaphor, allows any object, and not just agents, to move freely about a network ensemble during runtime. Thus, mobility is realized at both the agent and the object level. Tasks and data may freely and dynamically migrate among the machines in the network associated with creating their agent. By allowing objects and agents to migrate, the invention provides a degree of adaptability and flexibility heretofore unachieved by the prior art.

[0050] Agent migration causes an entire agent of the present invention to be moved in a single atomic step. When an agent consists of multiple threads executing on different bases and an agent migration method is called, all of these threads are preferably gathered at one of the bases before migrating to the target base. Object migration permits internal data and associated threads to migrate. (Further details are provided below in the section entitled "Runtime System.")

[0051] It should be noted that in certain situations, some types of migration may be undesirable. To accommodate these scenarios, the invention provides an "Anchored" property for use when instances of the class implementing it are statically dependent on process-specific objects, such as I/O ports or interface objects to existing software. These objects should not migrate even if the agent which encapsulates them does. The invention also provides a "Pinned" property for objects, which is similar to the Anchored property but expresses a dynamic constraint. For example, when an object temporarily requires significant communication to a specific location, it can first migrate to that location, set a Pinned property, and then communicate efficiently. During this period, the object cannot be moved. If the object must migrate again, its Pinned property must first be reset.

[0052] As noted above, agent migration results in all of an agent's elements, such as objects and threads, being moved except for anchored or pinned objects. The invention provides for two types of agent migration: weak migration and complete migration. Agent migration is illustrated in FIGS. 4A-4C. FIG. 4A shows an agent 40j residing on a base 30j prior to migrating to a target base 30k. Agent 40j contains several objects 50, including several migratable objects 57 and an anchored object 58.

[0053] FIG. 4B shows the results of agent 40j migrating under weak migration. In weak mode, anchored or pinned objects remain at the original location and are accessed with remote references in order to maintain consistent values. Thus, as shown in FIG. 4B, after migration agent 40j spans both base 30j and base 30k (the portion of agent 40j on each base is a subagent). Migratable objects 57 are moved to destination base 30k, but anchored object 58 remains at base 30j, and a remote reference 59 is created within agent 40j on base 30k for accessing the anchored object on base 30j.