of an instance is preferably as follows:

- The instance's class
- (Integer hashCode)
- (Mutex)
- (Global id)
- The instance's fields

[0095] Most objects are not hashed, locked, imported or exported. Therefore, to reduce the size of these common objects these fields could be merged, at the cost of a slight increase in the cost of accessing them.

[0096] All information common to the instances of a particular class, including the data layout information needed by the garbage collector and marshalling code, is stored in the class.

(b) Arrays

[0097] For regularity, arrays are preferably represented as instances of a special array class. Each instance has fields containing the array's type, size, and elements. Array instances need to be handled specially by the garbage collector because, unlike other instances, the size of an array is not determined by the array's class.

(c) Class

[0098] As shown in FIG. 17, a class object 210 may be organized into five areas:

(1) Class-specific information for the garbage collector (GC);
(2) Instance-specific information for the garbage collector (GC);
(3) Data common to all classes;
(4) Instance and static method table; and
(5) Constants and static fields.

[0099] The following data is found in every class:

- Data layout information for this class;
- Data layout information for instances of this class, including whether this is an array class;
- This class's superclass;
- The instance of class "Class" for this class;
- The class loader used to load this class;
- Initialization status; and
- Interface method table index.

[0100] The method tables are sequences of pointers to code, one for each instance and static method in the class. An instance is invoked by jumping to the code found at the appropriate offset. Because instance method code offsets must be the same for a class and any subclasses, the instance method table begins at the same offset in every class.

[0101] The name of the class and the interfaces it implements are found in the Class instance. To speed up casts and run-time type checking, each class could also contain a succinct representation of its location in the class hierarchy.

[0102] Although not shown in FIG. 17, the code for a class's methods can contain pointers back to the class. Preferably class objects and their code are not in the heap. They are instead part of the class file, and are created when the class file is loaded.

(d) Thread

[0103] Threads express execution states of programs in runtime and may be instances of a thread class. In addition to the standard fields for that class, each thread contains a stack. This stack is a linked list of stack segments, each of which contain a sequence of stack frames. An implementation of a frame contains a pointer to size and type information for local variables and arguments. This information is used to properly handle routine type checking, and is also used by the garbage collector. It is possible to evaluate this information dynamically if garbage collection occurs infrequently.

(e) Subagents

[0104] A subagent is the portion of an agent that resides on a particular base. Instances within a subagent are "local" to that subagent; all other instances are "remote." Subagents are represented as instances of a subagent class. Their fields and methods are all related to the communication protocol and are detailed in that section.

(f) Remote References

[0105] References to instances that exist in other subagents have much the same representation as local instances. The class pointer does not point to the regular class, but instead to a copy of the class whose method pointers point to RPC stubs for the methods. Calling a method for a remote instance is identical to calling a method in a local instance. This avoids the need for testing the location of an object when doing a method dispatch. Such a test is required when doing a field reference for an instances other than *self.* Remote references have no fields; they have a non-null global id.

(g) Global Id

[0106] A global identifier or "global id" records the identity and current location of an instance that has been seen by more than one base. The global identity of the instance is determined by the base on which it was created along with an integer identifier assigned by that base.

[0107] Global identifiers are the mechanism by which