Description

FIELD OF THE INVENTION

[0001] The present invention relates to the field of distributed and parallel computer software programming, and in particular to a software system including distributed agents which exhibits enhanced process mobility and communication and facilitates the construction of network-centric applications suited for both homogeneous and heterogeneous network environments.

BACKGROUND

[0002] In distributed computer systems, emphasis has traditionally been placed on issues concerning the partitioning and transmission of data among a collection of distinct computers or "machines." Typically, these systems allow code to be distributed and accessed in one of two ways. For example, in client-server systems, each machine holds code controlling the resources found on that machine. In others, the same code image is found on all machines. In either case, some form of message-passing is used to invoke operations on remote sites.

[0003] Traditionally, process mobility (i.e. moving executing processes from one machine to another) has not been an issue of significant importance. In client-server based systems, process mobility is essentially irrelevant; tasks are heavyweight (i.e. contain a large amount of state) and control resources resident on a particular machine. In systems where all machines share the same code image, process mobility may be used to help performance by improving locality and load-balancing. However, tasks typically execute heavyweight procedures, making task migration infeasible. Moreover, an efficient task migration policy that has simple, well-understood semantics has not been achieved to date.

[0004] Recently, process mobility is becoming increasingly important to the implementation of distributed computer systems. Enhanced process mobility allows computations to dynamically reconfigure themselves, taking advantage of improved data locality, and reducing the number of non-local communication events initiated. Several distributed system models have been developed to provide a certain measure of process mobility.

Imperative Glue Systems

[0005] Imperative "glue" systems have been developed which generally operate as seamless extensions to an existing imperative programming language and add distribution and communication support to the existing language. Unfortunately, computation in imperative languages involves frequent modifications to shared global data, which is exactly what a distributed program needs to avoid. Two basic approaches have been developed to deal with this problem: distributed shared memory (or "DSM") and remote procedure call (or "RPC").

[0006] With DSM, while the distributed nature of the computation is largely invisible to the programmer, implementation complexity is greater than in a system which uses message-passing explicitly. All data is conceptually associated with a global address. Thus, the machine where a thread executes no longer influences the behavior of the program: dereferencing a global address may involve a remote communication to the machine "owning" the contents of that address. While DSM provides a mechanism to implement parallel dialects of imperative languages in a distributed environment, programmers have little control in specifying how coherence and consistency are realized. In particular, issues of process mobility become largely irrelevant since the distribution of data and tasks is implicitly handled by the implementation, and not explicitly managed by the program. While DSM simplifies programming, it is likely to be more effective when combined with mechanisms to explicitly control distribution and communication.

[0007] RPC provides a way of breaking a program into discrete parts, each of which runs in its own address space. Unlike DSM, RPC communication is explicit in the program, so programmers have complete control over costs. However, the semantics of RPC are substantially different from that of an ordinary procedure call. In particular, when a procedure P makes an RPC call to a procedure Q, the arguments to Q are marshaled and shipped to the machine where the computation should be performed. Stub generators on procedures linked to the application program are responsible for handling representation conversion and messaging. Arguments passed to a remote procedure are passed by copying. Thus, side effects to shared structures can no longer be used for communication between caller and callee. As a result, imperative programs must be substantially modified to run in a distributed environment using RPC. Consequently, programming a distributed agent system using RPC semantics is significantly more complex and subtle than sequential programming on a serial machine.

[0008] Process mobility, the ability to migrate a thread of control (or task) along with its associated state, is especially difficult. The imperative nature of these languages means that a large percentage of data found in programs must be global. Without using RPC, communication among processes must be via side effect, and not via allocation and copy. Thus, the advantages of having mobile processes is greatly mitigated. Conceptually, processes are highly mobile in these languages because they carry no state, but because they must frequently reference global (shared) data, process migration becomes useful only if the data they access moves along with the process requiring them. Given that global data is likely to be shared among several processes, the implicit coupling of data and code in imperative lan-