EMAA: An Extendable Mobile Agent Architecture

Russell P. Lentini, Goutham P. Rao, Jon N. Thies, and Jennifer Kay

Lockheed Martin Advanced Technology Laboratories
1 Federal Street, A&E, 3W
Camden, NJ 08102
{rlentini, grao, jthies, jkay}@atl.lmco.com

Abstract

The Extendable Mobile Agent Architecture (EMAA) is a new agent architecture specification that aids in the development of an agent system. The architecture's component design has layers of abstraction, providing a generic system. EMAA provides a framework for autonomous asynchronous mobile software agents to migrate among computing nodes in a network and exploit the resources at those nodes. A single implementation of EMAA can provide a foundation for several different agent applications.

Introduction

The Extendable Mobile Agent Architecture (EMAA) is a new agent architecture specification that aids in the development of an agent system. EMAA uses an object-oriented design, but deviates from the traditional message-passing paradigm among objects that are distributed throughout a network. It provides a simple way for a mobile agent to migrate from one computing node to another and to use the resources at that node. EMAA does not impose any restriction on the behavior of the agents, thus allowing autonomous behavior. This paper explains this new mobile agent architecture, and the extendable nature of its components.

One of the major benefits of a distributed computing environment is the ability to break problems up into smaller sub-problems and solve those sub-problems in parallel. EMAA exploits this characteristic by breaking an agent's high-level goal into tasks that can be executed in parallel. Another characteristic of distributed computing environments is that resources may be distributed among different nodes in the network. The concept of breaking a program up into sub-programs fits nicely with this structure because a task that needs to exploit a resource can be packaged into a separate, independent component.

The resources available at a particular node are determined by the application requirements. In some situations it is convenient to package the logic needed to access these resources into a distinct component. In the EMAA methodology, these service-providing components are called servers.

EMAA is an architecture specification; its implementation is totally reusable and extendable. A single implementation of EMAA may serve as a foundation for several different agent systems.

In this paper we introduce the concept of taskable agents and explain its robustness. This warrants an explanation of the components needed to support such an agent. We first give an overview of the architecture. We then concentrate on the various components that make up the system and their interactions.

The Extendable Mobile Agent Architecture

The architecture has three major components: the agents, servers and the dock. Agents perform the specialized, user-defined work. Agents travel through the system via the docks. The primary purpose of a dock is to serve as a daemon process used for sending and receiving agents between docks at other nodes. Furthermore, nodes that offer specialized services to agents must do so via a server.

Agents

The key functionality of a mobile-agent computing paradigm is the ability to have a program execute at one node, and then migrate to another node with its state preserved. This program is called the agent.

EMAA views an agent as a vehicle to carry and execute a number of programs, know as tasks, at different nodes. The goal of the agent is to complete all of its tasks, defining the concept of taskable agents. Since the agent is a mobile program, one must consider both the task as well as where that task is to be performed. EMAA supports the mapping of a task to a node, allocating resources for the task to be executed, through a table known as an itinerary. An itinerary contains a mapping of which computing nodes these tasks need to be performed on, and the program to be executed in an event that this task failed to be performed for whatever reason (the event handler).

There are a number of advantages from splitting an agent's goal into smaller sub-goals:

Parallelism: Multiple tasks may be executed in parallel, perhaps at different nodes; a single agent can start a task component on one node, and while that task is in progress, migrate to another node and start another task.

Configurable agents: Agents can be configured at run-time, without having to re-program a new agent for every new goal.

Reusable components: If two agents have slightly different goals, it may be possible to identify some common subtasks. The programs needed to accomplish these tasks may be re-used if they are properly created.

Servers

In many agent applications, one of the compelling reasons that an agent will visit a computing node is to utilize the resources at this node. There are three important points to be noted here. First, to conserve bandwidth we want to migrate as little code with an agent as possible. Second, the code or logic needed to exploit the resources at the node will usually be the same for all agents. Finally, it is desirable to separate the implementation of these resources from the implementation of the agent application.

It is beneficial to package the code needed to access these resources into separate components known as servers. Servers remain at nodes where the service is offered. It is beneficial to have a common interface between the agents and the servers that offer similar services at different nodes, even though the servers may differ in their implementation. This keeps the agent machine-independent.

Dock

The most important features of the dock are to serve as a daemon process to receive agents and as a placeholder for other components. The dock consists of the following components:

1. Communications façade
2. Agent manager
3. Server manager
4. Event Manager

Communications Façade: The communications façade is the daemon process itself and is meant to handle all connections to any other computing node. It manages the transmission of agents to and from the local node.

A well-known daemon process or component at a node is needed to receive agents. It is a good idea to add functionality to this same component to also send agents, instead of the agents interacting directly with the communications façade at the node the agent wants to migrate to. There are two compelling reasons for this. First, the communication software is packaged into one component, separating it from the rest of the agent system. Second, there is a more controlled mechanism for sending and receiving agents. Such a mechanism allows for optimization of network use in the presence of autonomous components.

Agent Manager: The agent manager is responsible for registering the agent and initializing it for execution. An agent manager is used in agent collaboration and agent validation for security reasons.

Server Manager: The server manager manages the server components at a node. The server manager provides a controlled mechanism to start and stop a service. The server manager is a good place to incorporate an economy model.

Event Manager: An event is defined as an announcement that some component in the system has reached an important state that may be of interest to other components. Many times it is unknown if and when a component will generate an event. For this reason, agents and servers dependent on events must have some way of "listening" for an event that may or may not happen. To support this, an event management scheme where a component can generate an event or wait for one is needed.

A mobile agent system by definition is dynamic in the number and types of components resident at the dock at any given time. This characteristic makes it difficult to pre-program a component that will raise an event to communicate directly with any other components which may need to be notified when that event occurs. Even if this could be done, it is more efficient to handle events through a centralized event manager.

Dock Interaction

Component Loading: When an agent migrates to a node, there may be some components of the agent already present at the destination node. It is more efficient for an agent to migrate with only the references to its components and use those present at the destination. If any component is not found at the node, then the agent manager can request the *component loader* to contact the last node the agent came from that is known to have that missing component.

Resource Servers: It is useful to have some mechanism to advertise a node's resources to all other nodes. This allows every agent to be aware of the resources available on the network. An agent can determine what resources it can use to achieve a goal at runtime. Therefore there is the concept of inter-node resource sharing. EMAA specifies a *resource server* for this purpose.

An Application to Information Discovery

Using the EMAA design, Advanced Technology Laboratories has developed a mobile-agent-based information retrieval and dissemination application. The Domain Adaptive Integration System (DAIS) is a DARPA funded project with the general purpose of pushing and pulling military intelligence information across low-bandwidth, unreliable networks. These networks consist of heterogeneous sets of nodes containing numerous distributed heterogeneous databases. The system has proven to be both easy to develop and extremely robust, running on even 4.8Kbs half-duplex wireless networks which lose connection frequently.