Chapter 12

Summary

Mobile agents were first conceived in the 1990s. Since then, various mobile agent systems have been designed, each focused on some relevant part in the design space of mobile agents: secure execution of mobile, portable (interpreted) code, interfaces for interacting with the environment or with other agents, directory services for locating agents, or on applications that require disconnected use or that use multiple agents to complete a task.

Most existing systems impose few constraints on the environment, for example on how agents may join the system or where they can migrate to. Mansion changes this: Mansion focuses on a programming paradigm for mobile agents which is controllable. Agents are mobile, but cannot migrate anywhere: they must adhere to certain rules conceived by an application designer. Within the rules of the application, agents are autonomous: they can be written in any programming language and contain any algorithm to search for information or to locate and interact with other agents, but they cannot escape the confines of the application.

The environment that Mansion provides to agents is called a world. A world is implemented as a distributed system consisting of various rooms that can be reached by following hyperlinks. The hyperlink structure originates from a world entrance room. This environment has similarities to the Web in that it uses hyperlinks, but it is also very different: it is impossible to enter a room without entering a world entrance room and following hyperlinks to migrate to the room first. Further, to be able to migrate to a room, an agent first has to physically migrate to a machine that hosts the room and that is trusted by the room’s owner. Because agents run on a machine controlled (or at least trusted) by the room’s owner, it becomes possible to enforce security policies on agents and on the way in which agents navigate a world and on how information flows from a room to the outside world.

Worlds are application-specific: their content and structure depends on the application, and the hyperlinked structure of a world ensures that some structure can be imposed on it, to help agents find their way and to locate content efficiently. Various types of world can be constructed, from very large-scale, loosely structured worlds to highly constrained, relatively
small worlds for specific applications.

A prototype world has been constructed consisting of about 20 rooms, with each 2 rooms representing a hospital, which are hyperlinked as a linked list of rooms. An agent can enter the world and search for sensitive content by entering the first room of the world and searching it, then moving onward to the next room. To search sensitive content, a room is provided by each hospital that confines the agent. In the prototype world, these confined rooms contain (personally identifiable) MRI scans.

Confined agents can search content, but when they leave the room they retain no memory of what they found or did in the room. An agent must write any interesting content that matches its search parameters in a list and pass it to a guardian agent that is provided by the confined room. After inspection of the list, the guardian agent may pass information on the items in the list to the room’s owner. This room owner can decide to contact the agent’s owner. This example is, for example, useful when searching for patients with a rare disease in preparation of a clinical trial—the important factors being that the agent cannot leak any sensitive data to the outside world and that there is sufficient incentive to find patients by searching datasets of sensitive data spread all over the world.

Other, less sensitive applications of Mansion can be more automated. An example is an agent searching for movies or music files, where the guardian agent may simply take the export list to prepare a shopping cart using which the agent’s owner can later buy the content found by its agent. Agents can also meet and interact to speed up their search or to negotiate a deal. The main function of rooms—with or without confinement—is to provide content that is difficult to index or find using a search engine, and to allow this content to be accessed and searched directly, on-site, by mobile agents.

Content in a room can be stored in an object: objects are passive and have methods that can be invoked by agents. Objects can only be invoked by an agent in the same room. Agents can help other agents to find information: agents can communicate with other agents of which they know the agent identifier, using migration transparent communication channels. Agents and objects are registered in a special object in the room, the Room Monitor Object (RMO), which also contains hyperlinks to other rooms. Each entity (agent, object, hyperlink) is annotated using an attribute set that allows agents to locate content effectively. The world designer determines what attributes are allowed in a world, based on application requirements.

The system is designed for scalability and security. Rooms and objects can be distributed over multiple physical machines, to allow the system to scale when many agents enter a room or access an object. Each room, object and agent is known by a location-independent, self-certifying identifier that not only functions to locate the entity, but also allows for authenticating it—the entity must be able to authenticate itself using a public key that corresponds to the self-certifying identifier. Each object (including the room monitor object) is accessible only from a middleware process that is a member of a zone: a zone is a set of processes indicated by a single self-certifying identifier that indicates the certificate authority of this zone. The middleware processes hold the same self-certifying zone identifier as the objects which are accessible from these middleware processes. The identifier of every zone in a world is
registered in a world-wide service. This per-world service is managed by the world owner and ensures that only trusted zones are part of a world. This bootstraps a semihierarchical trust model of a world.

In addition to the world’s zone structure, a (nonhierarchical) structure of hyperlinks between rooms exist. Each room owner can, independently from other room owners, add hyperlinks to other rooms to its room. In some cases—such as the medical world described above—a world owner may impose hyperlink constraints on a world that limit how rooms may hyperlink to other rooms. Other worlds may be loosely structured, where rooms may link to other rooms in the world as their owners see fit.

Other important components of Mansion are the world entrance rooms, through which agents can enter a world. In a world entrance room, agents can find information about the world and follow hyperlinks to enter the (first) rooms of a world. Thus, an agent can be helped to find its way. A world design document describes the main constraints of a world, so that agent programmers can ensure their agents can locate content or other agents effectively.

For security, the system presented in this thesis provides several mechanisms. A jailing system allows for protecting hosts against malicious agents, and for confining agents effectively, irrespective of the language in which the agent is written: the jailer confines binary Linux processes (which may contain an interpreter) such that these processes can only access a scratch jailing directory and can only connect to specific, predefined network endpoints. Starting from this, the Mansion middleware can ensure that an agent can only invoke (allowed) methods of the Mansion API. The Mansion API contains the calls to follow hyperlinks, invoke objects and connect to agents, and enforces both logical and security constraints on what an agent is allowed to do. The middleware is RPC-structured: it consists of several processes that invoke each other using RPC calls; agents are also executed as independent processes that invoke the Mansion API using RPC.

Besides the jailer, the middleware system makes use of an object server (containing the Mansion objects, currently written in C++), and it uses the Agent Operating System (AOS) which provides primitives for storing agents in so-called agent containers (ACs), and for shipping an agent in such a container when it migrates. An AC contains segments with the agent’s code, initial data and instructions, as well as data collected on its way. When an agent migrates, hashes and properties of its segments are contained in a data structure that is iteratively signed by the sending Middleware processes, thus constructing an efficient audit trail that ensures that tampering with an AC can be detected by the agent’s owner. By coupling audit trail verification to a handoff protocol that updates the agent’s address in one of the world’s agent location services, a tampered-with agent can be contained at its current location, protecting (distributed) resources at the world level against rogue agents at the same time.

All combined, Mansion provides a structured, closed environment in which the most important properties can be controlled by the world owner, yet where agents can implement their own algorithms for finding content and for searching (raw) data in the world autonomously for their owners.
A prototype world shows that it is possible to design a coherent world structure for which programming agents is straightforward. The prototype agents are small, yet they can navigate the world and search its content autonomously in an efficient way. The underlying trust model and security primitives ensure the system’s security and scalability. An exploration and implementation of applications is described as validation of the concepts, the design, and the implementation presented in this dissertation.