Itinerant Agents for Mobile Computing

David Chess, Benjamin Grosof, Colin Harrison,
David Levine, Colin Parris, and Gene Tsudik*
IBM T. J. Watson Research Center
Yorktown Heights, New York 10598

email: chess, grosof, cgh, dwl, cjparris@watson.ibm.com
and gts@zurlvm1.vnet.ibm.com

Keywords: Intelligent Agents, Mobile Computing, Script Languages, Network Services, Electronic Commerce, Information Retrieval, Knowledge Representation Languages.

1 Introduction

This paper describes an abstract framework for itinerant agents that can be used to implement secure, remote applications in large, public networks such as the Internet or the IBM Global Network. Itinerant agents are programs, dispatched from a source computer, that roam among a set of networked servers until they accomplish their task. This is an extension to the client / server model in which the client sends a portion of itself to the server for execution. An additional feature of itinerant agents is their ability to migrate from server to server, perhaps seeking one that can help with the user's task or perhaps collecting information from all of them. A major focus of the paper is the Agent Meeting Point, an abstraction that supports the interaction of agents with each other and server based resources.

Why is this extended form of client-server computing desirable or valuable? There are many detailed motivations for using itinerant agents [6]. They fall broadly into two categories: 1) support for mobile computers or lightweight devices and 2) the emerging need in rapidly evolving networks for an asynchronous method of searching for information or transaction services. For example:

1. The reduction of overall communication traffic over the low-bandwidth, high-latency, high-cost access networks typically employed by mobile computers.
2. The ability of the agent to engage in high-bandwidth communication (with a server, for example) to search through large, free text databases.
3. The ability of lightweight mobile computers to interact with heavyweight applications without prior, detailed knowledge of the remote server's capabilities.
4. The ability of the agent to integrate knowledge from the client and server and perform inferencing at the server.
5. The ability of the user to create "personalized services" by customizing agents that take up residence at a server.

These claimed advantages are, at present, hypothetical, and we are performing experiments to determine if they can be realized. We believe they merit initial exploration.

Public networks already contain an enormous number of computers capable of providing specific services. This number will continue to grow. These servers employ a wide variety of processors, operating systems, databases, application frameworks, and applications. An itinerant agent framework enables these numerous, heterogeneous servers to offer many advantages. These include a host-independent execution environment for itinerant agent programs, standard communication

*IBM Zurich Research Laboratory, Rueschlikon, Switzerland.

languages with which agents and servers can engage in dialogues, a method of employing public security services to enable authenticated access to server resources, and secure auditing and error recovery mechanisms[1].

This paper is structured as follows: We begin with an overview of the operation of an itinerant agent framework (section 1) and a review of previous work (section 2). In section 3 we consider likely applications of itinerant agents and discuss one specific example in detail. Section 4 is an architectural description of the structure of itinerant agents, the languages employed to create them, and the execution environments required at the servers. In section 5 we give a detailed description of how an itinerant agent is processed at a server. Section 6 discusses security issues, always important in network services and especially so in this case. Finally, in section 7, we describe the technical advantages of the itinerant agent framework and the services it enables.

2 Background

In this section we give an overview of the operation of an itinerant agent framework, some of the surrounding issues, and a history of work in this area.

In many cases, the itinerant agent is launched from a client device such as a laptop or desktop PC by an otherwise conventional application. We anticipate that most end-users will not write their own agents (though that is certainly possible), but that various classes of agents will be distributed by services for use by their subscribers or will be packaged with the client applications. The agent is initialized with the user's task (see below for examples) and transmitted by a message channel [26, 2]. The sending client may specify a destination service directly. But the client will more likely send the agent initially to, for example, a Yellow Pages server, which can propose servers to be visited to fulfill the user's task.

When the agent reaches a server, it is delivered to an agent execution environment, which we call an Agent Meeting Point (AMP)[2]. Upon arrival at an AMP, the agent's external wrapper is inspected for authentication credentials. After validation, the AMP examines the agent's description of itself. Ontologically named service requests are resolved to determine if the desired services are available at the AMP. If sufficient resources are available and permitted the constituent parts of the agent are passed to the services.

The executable portions of the agent are then started. In some cases the itinerant agent will be interacting directly with server resources via proxy objects, which enable access control to be enforced. In other cases, it will interact with a *static agent* resident at the AMP. The static agent may have been dispatched to the AMP by the sender of the itinerant agent, or it may have been installed by the server operator. Static agents enable the server's function to be personalized by the server's owner or by users. When the agent has successfully completed its task at this server, it may collect its state, including information acquired at the server, and request to be transported to a new host. Or, it may launch a smaller agent to deliver the acquired information to the sending client or to another server while it terminates. This ability to acquire knowledge and transport it from place to place is a key attribute of an itinerant agent. The new knowledge may be simply a new destination or it may be a security token or transaction. This ability means that the agent is not merely a program executed at a remote host and then returned to its origin, but a moving process that progressively accomplishes a task by moving from place to place.

If the agent proves to be unauthorized, or if the meeting point is unable to provide the agent with the resource it has requested, the AMP will take action based on the agent's header. It may discard the agent. It may send the indicated party a description of the failure. Or if it is capable it may propose one or more AMPs to satisfy the request.

The range of large-scale public networks, and the absence of formal administration arrangements among the many domains that comprise them, pose particular problems for navigation and authentication of itinerant agents. Unlike private networks where itinerant agents might be viewed as being "at home", itinerant agents navigating the Internet are very much on their own. Although they may be capable of sending messages back to their origins, it is counter-productive to impose a strong dependency on the (mobile) sending clients constant accessibility. So we seek a model in which the agent 1) carries with it the minimal, necessary, and sufficient information to accomplish its task, 2) can prove its authority to any AMP it may visit, and 3) can accumulate knowledge during its itinerary and make decisions based on that information. In addition the model should permit the detection of AMPs that tamper with agents and their data, and it should provide sufficient auditing for AMPs to prove they have behaved properly.

[1] In many cases, itinerant agents will in effect be performing electronic commerce. This requires us to consider many commercial issues, including payment methods. However, we believe the electronic payment systems being developed for other forms of electronic commerce will be applicable to commerce conducted by itinerant agents; hence we do not discuss such payment issues in this paper.

[2] The sending client may also have an AMP, which it can use for launching agents, but in general it is not necessary for a client to have the full functionality of an AMP.

The idea of dispatching a program for execution on a remote computer is quite old. Often the motivation has been that the local computer does not have the capacity to execute the program. Or perhaps the remote computer has direct access to some resource, such as an attached peripheral, that cannot be efficiently exported via the network. In the 1960s such schemes were employed to enable mini-computers to submit batch jobs on mainframes and to receive results, typically print files, back for local processing [3]. In the 1970s executable scripts were dispatched among networks of mini-computers to permit distributed, real-time processing [10].

Recently scripts have become a focus of attention. The concept of active mail has been used to enable widely available electronic mail services [4] to deliver executable scripts [23]. The Telescript mobile scripting language, for example, was deployed in 1994 for an initial set of services on AT&T's public PersonaLink network. Telescript provides security features intended to support electronic commerce by means of mobile agents [29].

The HotJava web browser [21] from Sun Microsystems will retrieve programs from World-Wide Web servers for immediate execution on the local machine. The programs are typically written in the Java language and compiled into a machine-independent intermediate code. The intent is to allow for great flexibility in the function and presentation of information retrieved from the server, while preserving the security of the local system by carefully controlling what the received program can do. These programs are only itinerant in a weak sense, since they are retrieved from the server by the browser, but go no further.

Other recent developments include the emergence of knowledge representation languages and protocols such as those in the ARPA Knowledge Sharing Effort. These include the Knowledge Query and Manipulation Language (KQML) [11], and the Knowledge Interchange Facility (KIF) [13], which provide a basis for agent-server or agent-agent communication. Recent interest in on-demand electronic commerce is leading to the creation of public security services such as TERISA, the joint effort of RSA and EIT [7]. The emergence of standard, distributed object-oriented frameworks is another important enabler for mobile scripts. In such frameworks, the need to transport methods can often be greatly reduced when common methods can be presumed to be remotely available. In such cases, transport of only a set of object references, instance data, and a process state are necessary.

Technologies are thus becoming available to successfully implement an itinerant agent framework and to propagate it throughout the public networks. Such a framework or, more likely, set of frameworks, may have a significant impact on existing businesses. Many industries have multiple intermediaries between the creator of a good or service and its ultimate consumer. These intermediaries add value by applying expert knowledge, for example, to perform targeted distribution or to add ancillary services. Such industries include travel agents, financial and insurance brokers, and online information services. Itinerant agent frameworks are likely to impact such industries by replacing or eliminating some or all intermediaries. This process, which we call *disintermediation*, is facilitated in itinerant agent frameworks through the communication of knowledge as data among consumers and providers.

A corollary is that agent frameworks provide a mechanism for the creation of small, knowledge-based businesses offering services to large companies that formerly had to provide such services themselves. Examples are specialists in international trade, government regulations, language translation, and so forth. The agent framework enables people with expertise to make their skills available through the global network with a relatively low investment in computer technology, and with simple payment methods.

Agent frameworks thus play a role in the ongoing re-construction of large-scale corporations in which more services are outsourced and each enterprise becomes increasingly focused on its area of expertise. Large companies can shed supporting staffs and small companies can offer niche services.

3 Itinerant Agent Applications

In this section we suggest some typical applications of itinerant agents and give one scenario in some detail. An itinerant agent provides an extension to conventional client-server computing. Within this extension it supports several styles of application, which differ in the nature of the interaction between the itinerant agent and the server or servers it visits. The styles of interaction can be represented by *models*. Examples of these models range from simple to complex interactions:

- An *Information Dispersal/Retrieval Model*

  A client sends its agent to various host servers to update the latest version of an application, or into the network to retrieve the latest version of a technical paper on "Agent Technologies". This model represents a simple interaction based on an *ask/receive* paradigm between an itinerant agent and a static agent. In this case, the itinerant agent serves as the courier and installer for data or program content.

- A *Collaborative Model*

  The five authors of this paper send our agents to an AMP, which has a service that allows you to reserve conference rooms and schedule a meeting. This model represents a more complicated interaction in which there is a single clearly defined goal and the agents are required not only to ask for and receive information, but to evaluate and compromise based on a range of preferences. In this case, the itinerant agents convey not only the specified task, but relevant knowledge from the rule bases of the requesters, which are combined at the AMP and processed to yield a solution. The ability to transport and combine knowledge is supported by one or more *Agent Communication Languages (ACLs)*. In some cases the ACL may be a knowledge representation language from the artificial intelligence community. In other cases it may be a language developed specifically for a certain purpose, such as *Electronic Data Interchange* [17].

- A *Procurement Model*

  An example of this interaction is that of an open-bidding auction in which multiple itinerant agents attempt to bid for goods or services offered by an auctioneer (usually a static agent). This model represents a very complex interaction governed by an auction protocol in which the agents' goals and monetary resources are hidden from other agents [28]. The procurement model also includes the activities of strategy and electronic commerce. This model has many "sub models" such as *electronic malls* (many static agents and itinerant agents), *flea markets* (many itinerant agents), and *sealed bidding auctions*.

To understand this method of client/server computing more clearly, let's examine a scenario that serves to illustrate the major features of an itinerant agent framework. The scenario involves the purchase of airline seats by an employee of a company (Figure 1). The scenario is incomplete in that it does not address several important issues, including: error recovery and auditing, privacy, and various aspects of trust among the participants. However, some of these are considered in subsequent sections.

3.1 Travel Reservation Scenario

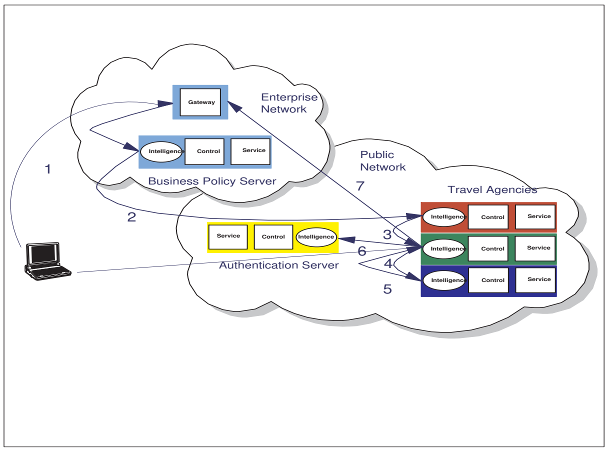

Figure 1: Travel Reservation Scenario

A mobile employee has a need to fly from New York City to Austin, Texas on a Thursday evening. His business will be completed by Friday evening, but if there is a significant fare saving, he is willing to stay in Austin until Sunday. He has a portable computer, which is able to access his company's LAN via a public network and a secure gateway. He uses a form or a dialogue to state his need. The application translates this need into a task expressed in an Agent Communication Language, using a vocabulary standardized for travel reservations. This task specification is used to create an instance of a Transaction Agent in the portable computer. The Transaction Agent is a program, expressed in a script language, that is able to interact via an AMP with a transaction server, assess the results of the transaction, make a decision, and commit a purchase. The Transaction Agent is also given the user's preferences for travel reservations (expressed as rules), and the agent is digitally signed with the user's authority.

The user's employer has a travel policy and a number of preferred providers of travel services, so the agent is sent initially from the laptop to the employer's Business Policy Server. To achieve this, the agent is partially encrypted with the Business Policy Server's public key, and then passed to a messaging system for transport via the LAN gateway to the server (step 1 in Figure 1). The mobile user then disconnects his computer from the network. At the Business Policy Server's AMP, the incoming agent is decrypted and authenticated using the AMP's private key. The agent and the AMP engage in a dialogue in which the agent communicates its task, which is to perform a transaction to obtain airline seats. The Business Policy Server's AMP verifies that the task does not violate current policy by employing an inferencing system to see whether the expressed facts conflict with the company's policy. If all is well, a subset of the policy rules are attached to the agent, it is given a list of approved travel agencies and their public keys, and the agent is encrypted with the key of the first agency's AMP. The agent is now passed to a messaging system for transport to the first server to be visited (step 2). At the first server, the AMP uses its private key to decrypt the agent, ensuring that the agent can only be executed on the intended servers. The AMP then verifies the authority of the agent by examining the credentials it carries from one of the public security services to which the travel agency subscribes. The agent and the AMP engage in a dialogue in which the agent communicates its assignment, to obtain airline seats. The travel agency AMP accepts the specification of the transaction and passes it to the on-line reservation system, parsing the Agent Communication Language expression of the task into the semantics of the reservation system, and performing the transaction dialogue. The reservation system returns (we hope) a number of candidate seats and prices, for return flights on Friday and Sunday. The agent employs a local inference engine to process each of these candidates against the user's travel preferences and the company's travel policy. After ordering the candidates according to preference, the agent selects the best candidate and requests the AMP to hold the seat for a certain time, say 10 minutes. The agent is then re-encrypted and transferred to the next server on the approved list (step 3).

The agent repeats this process at each of the servers visited (step 4)[3]. Whenever it finds a better candidate, it sends a message back to the server where it found the previous selection, releasing the hold it had requested. When it has examined a minimum number of candidates or visited a minimum number of servers, as specified by the company policy, it returns to the server of the best candidate and completes the transaction (step 5). This server's AMP issues a Ticket Agent, encrypted with the user's public key, obtained from the public security service, and sends it to the user (step 6). It then sends a Billing Agent (also encrypted), to the security service (step 7). Since the user has disconnected from the network, the Ticket Agent is held at the gateway until the user reconnects.

We do not claim that this is an accurate method for purchasing airline tickets, but it serves to illustrate many of the features and operations of an itinerant agent framework:

- The ability of the mobile user to dispatch an asynchronous task and receive a response at a later time.
- The ability of the itinerant agent to collect various kinds of knowledge to be applied to the execution of the task at the AMPs. The ability of the agent to migrate from place to place, accumulating information until it is able to complete its task.
- The ability of the agent framework to employ various public security services for its own protection, the protection of the servers, and the completion of a payment method.
- The ability of the agent to employ the visited AMPs for its own execution purposes (selecting the best seat) in addition to simply interacting with the server's resources or static agents.

In the rest of this paper we discuss how to structure itinerant agents and AMPs in order to realize these kinds of services.

4 Itinerant Agent Framework

4.1 Overview

In the following sections we describe the principal aspects of an itinerant agent framework that can support scenarios of the kind shown above. The framework naturally draws upon existing components of networks, such as name servers, directories, routers and so forth, and adds several new facilities:

[3] The manner in which the agent is authenticated at these servers is discussed in Section 6.

- Itinerant Agents

  These may be expressed in various programming languages and may transport knowledge expressed in various forms. They must be structured to engage in a progressive dialogue with the AMPs until they are able to execute or are rejected. An agent that executes at an AMP will usually complete its activity before moving to another AMP. However, an agent may also elect to suspend its activity at an AMP, transport itself to another AMP, and resume activity. In this case both the content of the agent (its physical state) and the representation of the environment in which the agent is executing (its execution state) must be moved to the new AMP. We will describe the abstract structure of itinerant agents and how they are processed by the AMP in section 4.2.

- Agent Languages

  Two kinds of languages are involved. One is the language in which the programmatic content of the agent is written; this is usually (though not necessarily) a script language. The second is a language for knowledge representation, which provides the means to express goals, tasks, preferences, and vocabularies appropriate to various domains. We will review the requirements for these languages in section 4.3.

- Agent State Accumulation

  In general, as agents interact with servers and other agents they will accumulate state information representing the results of these interactions. In section 4.4, we will discuss how state may be accumulated and managed in the agent framework.

- Agent Meeting Places

  These have various subcomponents, and they are the principal means by which a server becomes part of the agent framework. We think of the AMP as a broker between the agents, which are making requests for resources and services, and the applications that implement these resources and services. We will describe the AMP and its sub-components in detail in sections 4.5 and 4.6.

- Public Security Services

  These security services, which are trusted by the servers taking part in the agent framework, include certificates of authenticity and other services to the mobile agents. The framework places certain requirements on these services, and we describe them in section 6.

There are several closely related topics that we will not cover in this paper:

- User Interface Agents

  These are used by client applications to launch agents and to receive and present their results. Itinerant agents may well be launched from AMPs, but they can also be launched and received directly by client applications, which may vary widely.

- Static Agents

  Itinerant agents are a subset of the larger question of intelligent agents acting on behalf of users. The same basic framework applies to these agents and their access to services, but with a different set of communications, protocol, and security issues.

- Non Agent Invocation of AMP Services

  Many of the services provided by the AMP to local agents are of interest to other AMPs, static agents and normal applications within the larger computing infrastructure. RPC or CORBA/OMG [14] style bindings can be made available to these potential clients. Again, a set of issues that are not germane to itinerant agents apply to these services.

- Agent Status Management

  One of the inhibitors to building and deploying itinerant agent-based solutions is the complexity of managing the activity of an agent traversing multiple loosely coupled nodes. A complete itinerant agent framework will include services to register work as it is initiated, track its success or failure, and recover from system failures. These services need to address such issues as recovering in the face of single and multiple AMP failures, partitions of the underlying network, and poorly behaved services, AMPs, and agents. Therefore, we will generally mention that the problems exist, that there are services planned within the framework to address them, and defer detailed discussion to papers with these issues as their primary focus.

- Virus Control Services.

  It is well known that accepting unknown programs into a computer invites corruption and subversion of that computer. There are efforts in progress to define such virus services and it is not the purpose of this paper to describe any of these.

We begin the discussion of the framework by first describing the basic structure of an itinerant agent.

4.2 Itinerant Agent Structure

The basic structure of an itinerant agent (as shown in Figure 2) can be divided into three distinct portions: the *Agent Passport*, the *Table Of Contents (TOC)*, and the *Components*, which are discussed below.

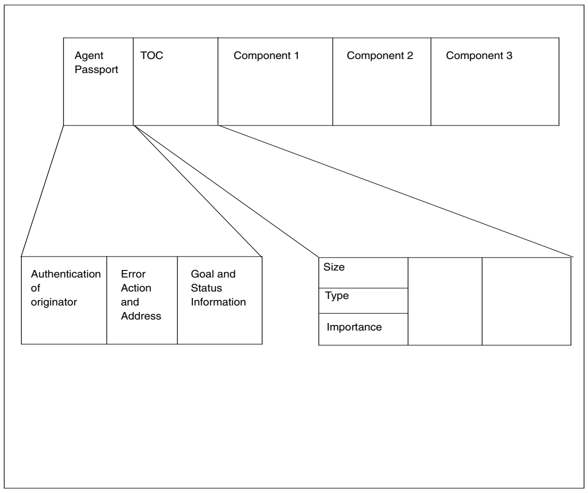

Figure 2: Encapsulated Agent Format

The *Agent Passport* consists of the basic information required to permit the agent to move from AMP to AMP. This includes:

- Authentication of Originator

  This certificate includes the name or the authority of the request's owner and the name or names of other authority sanctioning entities. In the example of Figure 1, this certificate may carry the identity of the originator of the agent and the "network name" of the business server. This may allow the travel services to provide the agent with the business rate for the tickets.

- Error Actions and Addresses

  This is the action the AMP should take when an error occurs while processing the agent. Some possible actions include: discarding the agent without comment; delivering an error notification to a specified address; routing the agent to another AMP.

- Goals and Status Information

  This information includes a representation of the agent's goal, and can include its relationship to other agents and their goals. Note that this agent may be the child of an agent present at the AMP, returning after an assignment from that parent. In some cases, the agent may require the AMP to notify an external entity (another agent or a client) of its status or progress based on some specified condition. The address of the external entity and the conditions are provided in this information.

Note also that this structure is that of only the agent, not any headers or segmentation imposed by communications protocols used to move it between AMPs.

The *Table Of Contents* for the body of an agent provides a map of its structure. Each component has a size, type and importance. The size, as expected, is the size of the component. The type field contains a simple representation of what is required to process the component. The importance field describes whether the component is necessary for the agent to be instantiated at the AMP. This permits agents to carry obscure components through AMPs that do not support these components, and to avoid unpacking components that will not be used at any AMP.

Note that in order to provide for arbitrary length expressions of the service type needed to support the component, the type fields are stored in a variable length section of the TOC. Offsets to the start of each type and its length are stored in the fixed table structure of the TOC.

4.3 Agent Language

4.3.1 Agent Programming Languages

One of the basic goals of this framework is that it should support a broad range of languages. In theory, any executable environment willing to conform to the architecture of the AMP can be supported. In practice, however, some languages are more practical for writing itinerant agents than others. We will briefly discuss some of the desirable attributes for an agent programming language.

- High Level Language

  While not strictly required, it is very difficult to secure environments that permit raw machine code to be executed as agent code.

- Object Oriented

  While procedural languages can be made to fit within an agent framework, Object-Oriented (OO) languages provide several advantages for the builder of agent frameworks. The object abstraction provides a good leverage point for access control and data mobility.

- Mobility Hooks

  Mobility is a key feature of itinerant agents, so it is desirable that an agent language easily support moving the agent. However, agent language support is not enough and some standardized primitives will be needed to invoke the collection of an agent's parts and the movement of the agent from one AMP to another. The language support needed for movement is discussed in more detail in section 4.4.1.

- Fork / Spawn construct

  As with mobility, itinerant agents will frequently desire to split up their processing among multiple copies of themselves at multiple AMPs or spawn agents to traverse the network. Again it is not necessary that the language expressly include support for this as primitive, as some standardized primitives will have to be provided by the environment.

- Distributed Computing

  As itinerant agents are dealing directly with a distributed computing environment, it is useful for the agent language to provide constructs that help programmers manage these issues. The ability to create proxy objects to represent remote objects is one example of this sort of support. Other examples include expressing synchronization points between multiple threads of execution, providing facilities to flatten and unflatten objects, and constructs to facilitate coordinating the management of objects shared between environments.

4.3.2 Knowledge Representation Languages

Languages for knowledge representation (KR) provide the means to express goals, tasks, preferences, beliefs, and vocabularies appropriate to various domains. Such goals and beliefs may take the form of facts, rules, other logical formulas, defaults, probabilities and utilities, fuzzy sets and logic, neural networks, and plan and action operators, to name a few.

By a language, we mean a formalism that defines what one can express in the way of beliefs (e.g., data) and goals (e.g., requests), as well as other similar entities (e.g., preferences). A language has syntax: including structure, operators, arguments, sequence constraints, block delimiters, and connectives. A language may also have semantics: a mapping of the language's syntactic entities to a set of meanings that are described in terms of another formal system. For example, the semantics of knowledge representations based on classical logic is often described in terms of model theory.

Languages (e.g., first-order logic) are often parametrized by vocabularies. By a vocabulary, we mean a collection of building blocks for representation, in the following sense. In classical logic (the underpinning of most practical knowledge representation systems today), this includes a description of the predicates and functions (symbols) out of which belief and goal expressions are syntactically built. More informally, these predicates and functions correspond to what are otherwise known as attributes, relationships, and entities, and need to be described in terms of their symbols and what kinds of arguments they take (arity and types). Vocabulary also includes other kinds of definitional information such as taxonomic (or subsumption) hierarchy and mutual exclusion (or distinctness) among various classes.

There are, of course, many different kinds of KR languages. A leading approach to interoperation and high-level languages for interchange and interaction is via the ARPA Knowledge Sharing Effort [22, 25].

The Knowledge Sharing Effort today has several parts; we focus here on three.

- KIF

  Knowledge Interchange Format (KIF) [13] is a rich standard language for interchange between other KR languages. It can express beliefs, rules, facts, partial descriptions of procedures, and more. Currently under construction are translators for a number of specific KR languages (services, systems) already in practical use. These translate between various KR languages and KIF, and therefore transitively to each other. Translating between n KR languages through one central common language, such as KIF, reduces the number of translators from $O(n^2)$ to $O(n)$.

- Ontolingua

  Ontolingua [15] [16] is a facility (system and approach) for creating and maintaining vocabularies. In Ontolingua a number of domain-specific vocabularies already exist for mechanical and electrical engineering, where the vocabularies contain terminologies usually found in standard introductory textbooks. One of the major advantages of agents is that they can bring domain-specific knowledge to bear on problems. This entails the opportunity, and need, for domain-specific vocabularies. To develop and maintain a single comprehensive global vocabulary is practically impossible. Interestingly, an AMP itself as a domain requires a vocabulary. The Knowledge Sharing Effort approach supports named vocabularies, which qualify vocabulary terms. This helps to reduce confusion when agents and AMPs are communicating in multiple vocabularies.

- KQML

  Knowledge Query and Manipulation Language (KQML) [12, 11, 5] is a high-level protocol and language for agent-service and agent-agent interactions and communications. KQML can be viewed as providing a "package" layer to wrap around the transport of "content" such as might be described in KIF. KQML is based on the concept of a performative from speech act theory in linguistics. Currently defined performatives include: ask, tell, forward, and broker. KQML passes as parameters: the language and vocabulary in which the content is represented. Also, KQML can nest performatives, such that, more specialized performatives can be nested within more general-purpose performatives.

EDI (Electronic Data Interchange) [17] deserves special mention as a KR language because of its importance as a standard practical framework for electronic commerce. While not an AI-style language, it serves much the same purpose.

Knowledge representation is important for routing in the broad sense. In order for itinerant agents to encapsulate the goals of their user, and to find services that have answers related to these goals, information must be expressed in terms that can be used to match goals and the services that can fulfill these goals.

The need to support KR languages and the opportunity to employ them thus motivate a number of components within our proposed AMP. In particular, the Deep Request Handler, the Linguistic Registry, and the Shallow Request Handler are oriented toward knowledge representation issues.

4.4 Moving Programs and Accumulated State

4.4.1 Moving Programs

Concerning the movement of itinerant agents, two possibilities exist. A simple case where the agent is a program that runs to completion at the agent's destination, and a more complex case in which an agent program begins executing at one AMP and then decides to move, complete with its current state, to another AMP.

In the simple case, the program can be encoded, loaded, and run until it signals completion. This can be done with only a few changes to an existing execution engine[4]. In the more complex case in which a program wishes to move from one server to another in mid execution, an important question arises how to extract the program's variables and state from the execution environment and insert them into another execution engine.

Briefly, there are several approaches to the problem of moving executing programs. One of the easiest is to allocate all information associated with the program's execution on the program stack and transport the stack, along with the engine, from one execution environment to another. Another approach involves keeping a registry of variables as they are created and moving the registry and the variables. In these two cases and other similar schemes, more extensive changes are required to the execution environment.

4.4.2 Accumulated State

An agent traversing several servers within a network of servers may need to accumulate information derived at each of the servers it visits. Agents can accumulate this information through two primary mechanisms: by adding new objects containing the new information and state to its existing collection of objects, or capturing state through changes within its existing collection of objects.

Consider an agent that is searching for information matching some specific request. As the agent traverses the network of servers, it adds these matching documents to its collection of items. Note that many of these items could be marked as cargo in the TOC and not be instantiated as the agent visits various AMPs.

State can also be captured within an agent's existing collection of objects. The degenerate case of this is an agent that consists of a single object, that is, an executable program and its execution context. An agent searching for the lowest price offer for a specific item need only carry the current lowest bid (and the identity of the bidder) in a local variable within its execution context. As the agent moves from AMP to AMP, the program and its context are saved, moved, and restored.

More complicated combinations of the above mechanisms are possible. An agent may build a separate object for holding the state of its work and update this object as it traverses a collection of servers. It is also possible that an agent will send state updates back to its origin point as it traverses the network.

4.5 Agent Meeting Point Structure

It is our belief that an agent infrastructure will be most naturally expressed in terms of a set of object services. Objects provide a number of attributes that are of particular interest in the distributed agent environment. Objects form a coherent, concrete element of mobility. We can move an entire object and we can refer to remote objects through proxy objects. We can apply authentication and access controls to objects more readily than to procedural data structures. By using distributed computing frameworks such as those being built by the OMG/CORBA community, we can delegate a large number of distributed computing issues to the computing framework.

The basic structure of the AMP is a set of OO classes designed to provide a very lightweight, minimal framework for building specific meeting points. The focus of this framework is providing the set of base services that can be used to register the services that a specific AMP will use.

The base services will include a Shallow Request Handler, that will permit components that snap into the framework to express what function they provide in terms of a simple knowledge representation scheme.

Although even very basic services will be accessed through the Shallow Request Handler, we will define the AMP as fundamentally including these functions. This permits us to provide a sufficiently rich and robust description of the AMP while allowing for incremental development and extension of the AMP structure.

In this initial work we focus on the design of a minimal structure that can support basic agent interaction. Yet the structure can be easily expanded to support a broad range of service types and interaction models. Our approach is to articulate an

[4] This execution engine is usually an interpreter.

underlying set of component frameworks that focus on building a minimum set of services for registering and routing work within the AMP.

On top of the Shallow Request Handler and the base components we define additional parts that provide specific services. These parts center on the Deep Request Handler and the Linguistic Registry, the focus for managing more complex knowledge representation and semantic issues.

For the purposes of this paper, we will ignore the admittedly complex issues surrounding the proper construction of an object-oriented, distributed computing environment and focus on the portions of an agent meeting place that are agentry specific.

4.6 Functional Decomposition of the *AMP*

The functional decomposition is shown in Figure 3, with the components comprising the architecture discussed in subsequent sections:

- Base Object Services
- Communication Portals
- Authentication Services
- Concierge
- Shallow Request Handler
- Linguistic Registry
- Deep Request Handler
- Resource Manager
- Agent Execution Environments (Including Scripting)
- Event Delivery Services
- Agent Status Services

4.6.1 Base Object Services

The Base Objects Services are CORBA/OMG based object services or their equivalent. Included are language-independent, object interface bindings, remote object invocation, marshaling of objects, object based access control services, persistence, and replication.

4.6.2 Communication Portals

The Communication Portals are responsible for managing the arrival and departure of itinerant agents, as well as routing messages to the residing agents or services in the AMP. The Communication Portals support protocol handlers, each of which manages a specific protocol, providing both inbound and outbound services to encapsulated agents. They also provide a consistent interface to the bulk of the AMP through *location* objects, which provide a set of methods to send agents and messages to abstract locations. For inbound services, the Communication Portals extract the arriving mobile agent and pass it to the itinerant agent concierge, along with a location object for the arriving agent's location.

Communication Portals may support either session-oriented connections to other parts of the infrastructure or may support messaging based protocols. The Communication Portals mask the underlying transport service and are responsible for the transport-specific details associated with the transports and their media, such as header, trailers, and data representation.

In addition to managing the arrival and departure of agents, Communication Portals are responsible for the handling of messages directed to components and agents residing within the AMP. Agents can communicate remotely with other agents or AMPs through the passing of messages. Communication Portals utilize the Shallow Request Handler to direct such messages into the AMP.

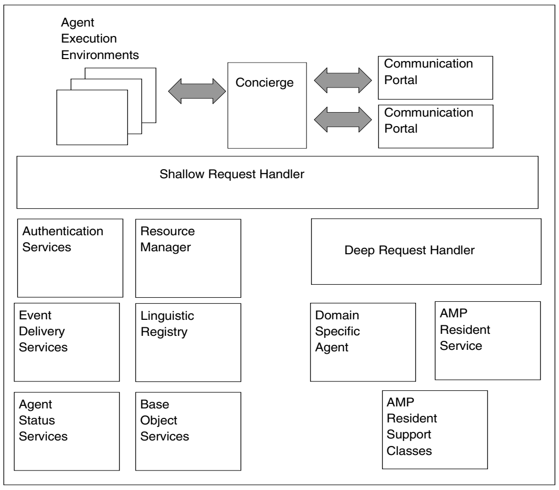

Figure 3: Agent Meeting Points - Functional Level Decomposition

4.6.3 Authentication Services

The Authentication Services are an OO encapsulation of the underlying distributed authentication scheme used in the agent network. While the Authentication Services component will primarily be used by the Concierge, it is also generally available to other components and agents that need to perform authentication.

4.6.4 Concierge

The Concierge acts much as a concierge does in a full service hotel. It examines an agent's credentials, makes sure that the facilities that the agent has requested are available, and helps the agent arrive at the AMP. The Concierge is also available to help agents who wish to travel to other AMPs. The Concierge provides a set of functions to gather an agent and its collected parts, package them and their credentials into a standard form, and pass them to the desired Communications Portal.

In order for the Concierge to perform its task it requires an intimate understanding of the basic headers of the encapsulated agent (as shown in Figure 2). Upon arrival of an encapsulated agent, the Concierge strips the *Agent Passport* from the encapsulated agent. This passport contains the information needed to authenticate the agent (as discussed in Section 4.2). Either the passport is validated, or the information within the passport is used to manage the graceful rejection of the agent. The passport is also used to assign the agent an *Access Control Factor,* which determines the permissions that apply to the agent and its components.

If the agent is admitted to the AMP, the Concierge then uses the TOC to examine the goal of the agent and the basic services it requires. The Concierge strips this information from the agent header and passes it to the Shallow Request Handler. The Shallow Request Handler (possibly with use of the Linguistic Registry and Deep Request Handler) passes back both the list of mapped resources and a list of those resources that could not be mapped. The Concierge then decides to either setup the agent or generate an error report.

The approach of having the Concierge manage goals through the Shallow Request Handler (and by implication the Deep Request Handler as needed), permits goals to be expressed at multiple levels of language ranging from low-level requests in the Shallow Request Handler's vocabulary to high-level requests expressed in KQML performatives. As the Deep Request Handler is an extensible component of the AMP and capable of translating representations, we can extend this range as new representations are developed.

4.6.5 Shallow Request Handler

The Shallow Request Handler forms the glue that binds the AMP, acting as the interface between agents and the components of the AMP. These components register themselves using a description of the service they perform. The Shallow Request Handler then uses this description to route requests from agents to the components that support them.

The Shallow Request Handler uses a limited vocabulary to match user requests with available services. The Shallow Request Handler is also capable of recognizing requests expressed in forms of notation beyond its vocabulary. When such a request is passed to the Shallow Request Handler, it checks its cache for a translated version of the request. If it does not find one it sees if the Linguistic Registry has the vocabulary registered, and if so, passes the request to the Deep Request Handler for translation.

The Shallow Request Handler uses the access control factors assigned agents by the concierge to limit their access to those objects and services they are permitted to manipulate.

4.6.6 Linguistic Registry

The Linguistic Registry is a database used to support medium-level and high-level understanding of agent communications. It registers 1) languages and 2) vocabularies (that is, ontologies). It does not itself keep all of the information about the languages and vocabularies; rather, it keeps track of which languages are known and which AMP services know these languages. This is important because we expect that various itinerant agents may represent their goals, facts, and rules in different high-level languages. The Linguistic Registry's purpose can be viewed in terms of the paradigm of the DARPA Knowledge Sharing Effort approach (recall section 4.3.2, especially Ontolingua). There, a prime focus is support for multiple agent communication languages and vocabularies. Other components of the itinerant agent framework, such as the Concierge, employ the Linguistic Registry to help them find the information needed to understand and converse with particular agents. One scenario: an incoming agent describes in KQML its topmost-level goal ("performative" in KQML), and names its language and vocabulary via KQML parameters. The Concierge consults the Linguistic Registry and discovers that this AMP does not know the vocabulary the agent needs. The Concierge finally tenders its regrets to the agent: in effect, "sorry, I can't meet your basic needs".

4.6.7 Deep Request Handler

The Deep Request Handler helps the Shallow Request Handler deal with more special or difficult requests. It maps a request by an agent into one or more service destinations, which may be another (non-agent) service or another agent, either in this same AMP or at a remote server. The Deep Request Handler provides, in effect, an extended directory service. It can be viewed as a kind of facilitator, and hence as itself a kind of agent. An interesting case occurs when the service destination is another agent that performs a similar task, that is, a facilitator agent. Generally, an alternative aspect of the Deep Request Handler is the provision of a "social directory" of other agents, which may include their identities, interests, languages (vocabularies), and addresses.

The Deep Request Handler is oriented towards requests provided in the form of a description. This description might be partial, such as, "I'd like to talk to a travel agent who handles international airplane flights". The Deep Request Handler matches service request descriptions to its knowledge of available services. This process might involve, for example, inferencing over a specialized knowledge base of facts and rules describing available services.

The Deep Request Handler translates and reformulates the request, perhaps using inferencing, into one that can be executed via a (list of) destination(s). To do so, it employs the language and vocabulary information obtained with the assistance of the Linguistic Registry. The Deep Request Handler may itself call upon other services or agents to help. These agents are special-purpose intelligent agents that are domain-specific.

Information access is an important category of request. Descriptions of information resources, at the level of database schemas and associated conceptual hierarchies, are thus an interesting kind of knowledge base the Deep Request Handler might employ.

More examples of requests fielded by the Deep Request Handler include:

1. "I want a printer that can handle Viceroy 4000 format and has more than 300dpi."

2. "Tell me the other participants in the ongoing automobile auction being held at this AMP."

3. "Tell the other auction participants about my identity and interests."

4. “Give me the list of any other agents who have the goal of buying or selling DOS-compatible double-speed CD-ROM drives."

5. “I want to subscribe to the Refrigerator Today news service.”

6. “I want to append to the Refrigerator Today news service.”

7. “I want to publish Refrigerator Today as a news service on a major public network.”

4.6.8 Resource Manager

The Resource Manager serves two primary purposes. It acts as a registry for all of the active agents within the AMP and the resources associated with these agents. It also serves as the focus for managing the use of resources within the AMP. It is where an agent’s current resource allocation is stored. As components within the AMP perform services for the agent, the cost of these services is deducted from the resources available to the agent.

By its nature, access to the Resource Manager must be strictly limited to those trusted components of the AMP that deal with allocating and managing agent resources. The current allocation any agent has may be determined by that agent, but the allocation can only be increased through trusted interfaces.

The Resource Manager can also act as a firebreak against the excessive use of resources by agents. Either the total permitted resources can be limited, or the rate at which the resources are consumed can be controlled.

4.6.9 Agent Execution Environments

These are execution environments that have registered to the AMP, offering to interpret scripts or agents whose encoding they support. In addition to registering their named environment with the meeting point, the Agent Execution Environments must provide access to the base facilities of the agent meeting point. This includes access to all of the exported object encapsulations.

An Agent Execution Environment can couple into the AMP at several levels. A full-function coupling would include registering not only the ability to accept agents to execute, but individual objects and scripts that have suspended themselves during execution. Based on which services the Execution Environment volunteers, more or less function is available to agents using a specific environment.

4.6.10 Event Delivery Services

Event Delivery Services are concerned with conveying information from services within the agent AMP to agents that have expressed an interest in this information.

Event generators can be local services, agents residing within the AMP, or programs coupled to the AMP through the underlying network. Event generators register the type of event they deliver to Event Delivery Services through the Shallow Request Handler. The event they are delivering is described so agents may search for available events.

Agents add entries to the Event Delivery Services through the Shallow Request Handler (possibly with the assistance of the Deep Request Handler and the Linguistic Registry). Once a request for event delivery has been set up, the Event Delivery Services act as the focal point for distributing events to all interested parties.

4.6.11 Agent Status Services

As agents traverse the network of servers they require mechanisms that track their progress and possibly report back to their origin. The Agent Status Services component provides a set of services that enable agents entering or leaving the AMP to log some significant aspect of its activities at this AMP. This log may include information associated with the agent’s arrival, its success at instantiating itself, and the transactions it conducted. The AMP also uses the Agent Status Services to report failures and errors to the originating AMP.

Agents can register their controlling AMP within their passport and request that the Agent Status Service provide reports to this controller. The controlling AMP then receives reports about the progress of the agent as it traverses the network. Further, as subordinate agents are spawned by an agent, they can register to either the local AMP (which can be asked to perform status forwarding) or at the controlling AMP.

The Agent Status Services are responsible for providing persistent storage of status reports and attempting reliable delivery of status reports to the controlling AMP. By relieving the agents of the need to embed the support needed to provide these services, the writing of reliable agents is significantly simplified.

5 Itinerant Agent Arrival and Departure

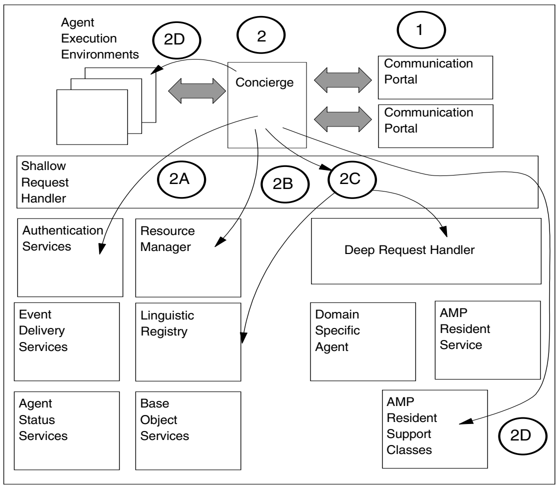

Figure 4: Agent Arrival

In this section we will walk through the steps that occur when an itinerant agent arrives at or departs from an AMP. The flow of an agent will be followed from its arrival through a Communication Portal, to the Concierge as shown in Figure 4. The actions taken by the Concierge will be described, followed by the instantiation of the agent's parts, and the initial execution of the agent.

We will then walk through the inverse scenario, of taking a running agent, building a package of its parts, and dispatching it to another AMP.

5.1 Agent Arrival

The itinerant agent's arrival proceeds in the following steps:

- (1) Encapsulated Agent Processing

  The agent arrives through a Communications Portal. At this point, the agent is wrapped in headers associated with the communications channel. The Communications Portal is responsible for fully receiving the agent and assembling it. Once assembled, the agent is then passed to the Concierge.

- (2a) Agent Authentication

  The initial step by the AMP's Concierge consists of origin authentication[5], data integrity check, and optional decryption of the agent's body.

  Origin authentication is obtained by verifying the certificate included in the agent passport. A data integrity check is performed by computing a strong (publicly- known) one-way function over the agent's body and matching against the corresponding value found in the agent passport. (This implies that the passport itself – or portions thereof – must be digitally signed by the originator.) The body of the agent can be optionally encrypted in transit. The originator needs only to encrypt for the first-hop AMP. Thereafter, each AMP can encrypt the agent under the next-hop AMP's

[5] Agent authentication, data integrity, and other security issues are discussed in greater detail in Section 6.

key as it gets ready to perform agent hand-off. Alternatively, if the originator knows the exact path beforehand, it can encrypt an individual copy of the agent for each AMP in the path[6].

In addition to its body (the executable component) the agent may carry some *appendages* in the form of information it collects from the visited AMPs. These appendages must be encrypted and/or integrity-protected individually by the AMPs that contribute them. Moreover, to protect against malicious AMPs tearing off other AMPs' information, agent hand-off between any two AMPs can be handled in such a way as to provide a reliable trace of the agent's state at every point in its itinerary. Next, the Concierge builds a local credential for the agent in the form of an *Access Control Factor,* which will be applied to all requests from the agent to use services within the AMP.

- (2b) Agent Registering

  Once the basic agent authentication is complete, the Concierge builds a status block for the agent and registers it to the Resource Manager. This block becomes the anchor for all information about the agent while it resides within the AMP.

- (2c) Resource Matching

  Once the basic authentication is complete, the Concierge examines the table of contents describing the parts of the agent. Each element in the TOC is passed to the Shallow Request Handler, which matches requests expressed in the native language of the AMP. It passes more exotic requests to the Deep Request Handler. The Concierge receives two collections, one of mapped elements, one of unavailable elements. It then determines if all the required elements are available, performing a graceful error delivery if not.

- (2d) Component Distribution

  The agent's elements are then routed to the service elements that support them and control is passed to one or more of the agent's parts. Each part is passed to the support element with a collection of handles to the other parts of the agent. Each part is also passed a number of handles to other useful objects. These include the agent's *goal and status* object, the local AMP's name, and the agent's passport object.

- (3) Agent Execution

  After instantiating all of the parts of the agent, one part, identified in the agent's TOC, begins executing. This will most likely be the procedural portion of the agent, which choreographs the execution of the agent's overall mission. This central component of the agent may have suspended its execution at a previous AMP and will need to have its execution state restored to continue its execution. In this case, the execution state, which would have been stored as one of the agent's components and identified in the TOC, will be restored.

5.2 Agent Departure

The inverse of an agent's arrival at an AMP is its departure. In this case, we start with a set of components running within the AMP. Each component must be stopped, stored in a transportable form, and added to a package. In order to transport the package from AMP to AMP, the package may need to be encrypted. At a minimum, the body will be sealed for data integrity checking and the passport signed by the AMP.

Any event registrations performed by the agent are removed, and the status block for the agent is updated. The Access Control Factor issued for the agent is revoked and its status block deleted. Upon successful delivery to a Communications Portal, the Concierge will update the agent's status management information, and write a log record recording the agent's passage through the AMP.

6 Security Issues

It is difficult to exaggerate the value and importance of security in an itinerant agent environment. It is, without a doubt, one of the cornerstone issues. While the availability of strong security features would not make itinerant agents immediately appealing, the absence of security would certainly make itinerant agents very unattractive. This section presents a brief overview of the basic security issues and concerns. More detailed treatments, concentrating on some of the new issues that arise in agent-based and highly-distributed systems, may be found in [9] and [1].

[6] Obviously, this is not very efficient.

The most fundamental security building block in an itinerant agent environment is the digital signature. Digital signatures are typically associated with public key cryptography[7]. The best-known and most popular method of obtaining digital signatures is the RSA public key cryptosystem[27].

Very germane to public key-based digital signatures is the concept of universal certification of all potential **signers** – the users that originate agents as well as the AMPs that execute them. Universal certification is a lengthy and cumbersome process, mostly because it involves the establishment of a global certification hierarchy similar to that in the electronic mail milieu. (Privacy-Enhanced Mail, PEM[20], is a case in point.) The good news is that certification hierarchies have been springing up in the last few years and there is no need to create a new hierarchy for the purpose of itinerant agents. A single certification hierarchy and one general certificate per entity[8] should be enough for a number of electronic activities.

Some observations on itinerant agent security:

- It is impossible to hide anything within an agent without the use of cryptography. If part (or all) of an agent is to be private, it has to be protected cryptographically.
- It is impossible to communicate secretly with a large, anonymous group of potential AMPs. This is, essentially, an issue of scale. One of the most appealing features of itinerant agents is their potential for seamless roaming of the global Internet (or a comparable internetwork), which contains a large and dynamic population of AMPs. An itinerant agent of the future is likely to visit a large number of AMPs.
- It is impossible to **prevent** agent tampering unless trusted (and tamper-resistant) hardware is available in AMPs. Without such hardware, a malicious AMP can always modify/manipulate the agent. If every AMP is equipped with, say, a general-purpose trusted processor, only agents that bear valid and recognizable credentials (certificates) and signatures would be allowed to execute. Currently tamper-resistant hardware and the enabling software are expensive and not readily available. Nonetheless, this is an area of active research (see, for example, [24]) and if it proves useful in agent-based computing, economies of scale will drive the prices down.
- It is impossible to verify with complete certainty that an arbitrary program (such as an incoming agent) is not a virus [8]. In practice, the problem of writing a program that can verify the correct (or even simply non-malicious) behavior of another program is unsolved. In general, the more an agent travels from environment to environment, the more opportunities there will be for tampering; trusting an agent means trusting every program that ever had write access to it, and (therefore) every program that ever had write access to any of them, and so on transitively. Well-designed access controls and hierarchical authentication services will reduce the danger, but these issues of trust will have to be addressed in designing AMP controls and agent abilities.
- Even agent strategies that seem logical and benign if considered by themselves can lead to undesirable results when embedded in a complex and highly interactive world [18, 19]. Security and integrity systems will have to deal with both innocent and malicious instances of various sorts of emergent behavior. Mail loops, the simplest and oldest example, can appear in an array of new forms as transmitted objects become active themselves. This is an area of ongoing research, but it is already clear that programs designed to move, and sometimes reproduce, will offer a number of new challenges.

Even though the above is somewhat disheartening, certain security goals are, nonetheless, attainable:

- Origin Authentication

  The origin of the agent can be unambiguously established including the public key certificate of the originator as part of the agent. (In practice this is not enough since data integrity goes hand in hand with origin authentication. In other words, the entire agent must be signed and integrity-protected.)
- Data Integrity

  The body (executable code) of the agent can be integrity-protected, thus allowing for after-the-fact **detection** of tampering. For example, a malicious AMP receives a *shopping* agent that specifies a ceiling of $400 for an airline ticket. The AMP cheats and ups the ceiling to $500. However, since it cannot produce a corresponding agent integrity check (computed with the aid of digital signatures), the originator can subsequently refuse to pay.

[7] Albeit, not all public key methods are amenable to digital signatures.
[8] This entity may be an amalgam of the user, the AMP, and an organization.

- Access/Itinerary Control

  The number and the identity of the AMPs can be restricted. The originator can explicitly specify (or otherwise restrict) the AMPs allowed to execute its agent. This is, of course, meaningful only if the agent is protected from tampering (if the agent includes a strong integrity-checking mechanism.) Also, access control by explicit naming is not very desirable, since – as mentioned above – this severely handicaps the notion of a *free-roaming* agent. Consequently, it is more likely that access control (or policy) will be coarser-grained, such as by attributes such as computing power, occupancy level, organizational affiliation, and pricing, to name a few.

- Agent's Privacy

  As alluded to above, it is impossible to keep an agent private unless its itinerary is known in advance. (In the latter case, the agent code can be wrapped up and delivered as multi-destination secure email, as in PEM [20].) Since one of the aesthetically appealing features of itinerant agents is precisely the freedom of movement, it is unrealistic to expect all agents to be launched with a set itinerary. The bottomline is the fundamental trade-off between the security advantages of fixed itineraries and the flexibility of free roaming.

- Privacy and Integrity of Gathered Information

  An agent's foremost task is to gather information, and the privacy of this information is perhaps of greater concern than the privacy of the agent's code. We identify two modes of information gathering: *stateless* and *stateful*. The former means that an agent intermittently sends parcels of acquired information *home* to its originator. In the extreme case, the agent sends the information home at every hop. In the stateful mode, the gathered information is attached in some way to the agent (the amount of carried information grows), but it is only delivered to the originator upon the agent's eventual return.

  Protecting information in the stateless mode is not difficult; the AMP currently executing the agent can take care of encrypting/signing and delivering the information to the agent's originator (or someone designated by the originator). Multiple AMPs executing the same agent cannot interfere with one another unless a malicious AMP unduly "terminates" an agent, thus preventing it from migrating to other AMPs.

  Stateful mode is similar with respect to information privacy; each AMP can simply attach its own encrypted information to the agent. An AMP may also attach a signed record with handoff parameters, including the identities of the AMP that the agent was received from, and of the next AMP it was sent to. This allows for subsequent verification of the agent's path, and is commensurate with the added data, the length of which is roughly proportional to the number of AMP hops.

  There remains a danger of a group of (at least two) AMPs collaborating to *tear off* information added by a rival AMP. However, this is only a problem if the agent's itinerary is treated as a *loose*, rather than a *strict*, route. (A loose route means that some hops may be omitted; a strict route implies traversal of all hops.) If an itinerary is treated as a strict route, traversal of all hops in the intended order can be securely verified.

  In general, an AMP can make use of the itinerary and other signed data attached to an incoming agent to decide on the privileges the agent should have. Some of this information can be made available to other agents, enabling them to determine how much the agent can be trusted. The general AMP architecture can be made flexible enough to support various models of trust.

  When an AMP receives a stateless-mode agent, it can verify that the agent program carries a valid signature of the agent's originator. It can thereafter be reasonably confident that the agent expresses the wishes of the originator. Stateful agents, on the other hand, carry not only the agent's body signed by the originator, but also state information. The latter can be signed only by the previously-visited AMP, where the agent reached its current state. In deciding how much to trust a stateful agent, an AMP must consider how much it trusts not only the originator, but also all preceding AMPs.

7 Discussion and Summary

In previous sections we motivated our work by describing several application models (and some applications) that can, collectively, be best supported by an itinerant agent framework. In this section we discuss some of the technical advantages of our itinerant agent framework, presenting them in the context of a few of the motivations and examples mentioned previously. Much of our initial motivation was derived from the desire to support mobile computers or lightweight devices with their inherent persistence and resource capacity limitations. These limitations and the subsequent benefits of the itinerant agent frameworks are described below. We also describe the framework support for agent and client interaction and present topics for future work.

7.1 Framework Support for Mobile Devices

Mobile devices are intermittently connected to a network (or server). Our itinerant agent framework provides the mobile client with the ability to formulate an agent request (possibly while disconnected), launch the agent during a brief connection session, and then immediately disconnect. When launched, the agent would proceed directly to a specified AMP or to a "known" AMP from which it would obtain the address of an AMP suitable to its needs. The *Concierge*, using the contents of the agent passport, would determine if the initially chosen AMP was suitable. If so, it would execute the agent after creating the environment requested in the agent's table of contents. In the event the AMP was not suitable, the Concierge would take the appropriate action based on the information in the passport. Such action might be to forward the agent to a suitable AMP, determined through its communication portals. The agent's response to the client, if any, is collected during a subsequent connection session.

Because the client may not be connected when the agent's response is ready, the agent may elect to wait at a predetermined AMP until the client is connected. The agent can make use of the *Agent Status Services* to indicate its status and await an indication from its client. Or it can alert the user to reconnect by using the services of the predetermined AMP to send a "page" to the client. It should be noted that the agent launched by the client can be written in any scripting language, and can communicate in any agent communication language, supported by the AMP. Also, the client may use any of a number of transport services, managed by the AMP, to launch the agent. The framework permits the support of multiple scripting and communication languages. The common abstractions required for interoperation among these communication languages are service descriptions (and other types) handled by the Shallow Request Handler and the Deep Request Handler, together with the Linguistic Registry.

When connected, mobile devices have low-bandwidth, high-latency, high-cost connections. Modems now provide 28.8 kbps links on dial-up lines, but wireless links above 12 kbps per client are unlikely to be widely available in this century and have significant usage costs. In this case the technical advantage provided by the framework lies in the ability of an agent to perform both information retrieval and filtering at a server, and to return to the client only the relevant (and reduced) information. In many cases a single agent may also be able to replace several tens of remote procedure call messages. The availability of the framework would permit an agent to pose a complex query that can be processed by the *Deep Request Handler* and the *Linguistic Registry*, and possibly answered by the *AMP Resident Services*. This would reduce the volume of data normally sent between the client on the mobile device and the information server.

Mobile devices have limited storage and processing capacity. While there are laptop computers today with 500 MB disks and Intel $Pentium^{TM}$ CPUs, there will always be a class of devices that tries to make do with minimal resources; one example is Hewlett Packard's successful $HP95/100/200^{TM}$ series, which natively offers only 2 MB of storage. The technical advantage afforded here lies in the ability of an agent to perform retrieval, processing, and filtering at an AMP supporting the information server, and to return to the client only the relevant information. The AMP would provide the agent with all of the necessary tools that could not be supported at the client. These may include an inference engine, a front-end natural language interface to a database, and other scripts for sorting and presenting the resulting information.

Because of the limited resource capacities in mobile devices, they may not be able to carry detailed information on the various AMPs in the network. Rather, they would launch their agents with high-level descriptions of their goals. They would rely on the *Linguistic Registry*, the *Deep Request Handler*, and *Communication Portals* to determine (possibly through an inference engine) the needed services. They would then access those services, or if necessary, direct the agent to another AMP. When additional services are required by an agent *Base Object Services* can be used to permit access to them, if available, at other AMPs. The resulting information transmitted to the device is minimized and the device itself does not need to perform any significant filtering or processing.

7.2 Framework Support for Agent and Client Interactions

Our itinerant agent framework also permits the AMP to support multiple agent and client interaction models. A few of the many possible interaction models were described in Section 3; These included, the *Information Dispersal/Retrieval*, the *Collaborative*, and the *Procurement* model. These models are instantiated by the use of a facilitator agent[9] and various components o f the AMP. Agents resident in the AMP can advertise their services by using the *Shallow Request Handler*.

[9] Previously mentioned in Section 4.6.7.

Agents who wish to subscribe to a service will add an entry to the *Event Delivery Services* for that service. A match can then be made between a provider of the service and a subscriber. This match permits us to gather parties interested in a specific model of interaction.

The facilitator agent, which presides over the agent interactions (ensuring that the model is adhered to), executes via the *Agent Execution Environment*. It uses the *Shallow Request Handler* and the *Event Delivery Services* to advertise its services and to access agents interested in its services. Recall (see section 7.1) that the facilitator and agent can be written in any supported scripting language and multiple scripting languages may be present in a single interaction.

Client interaction models can be supported by the creation of "personalized services". In these models the AMP supports servers that offer basic APIs and exports them via the *Base Object Services* for use by itinerant agents. Clients (or other agent authors) then have the freedom to use the server as they see fit. Clients can create agents that act as intermediaries between themselves and a server. As a result, they can present their requests and receive responses in a manner most suitable to them, thereby creating a "personalized" service.

Thus in a procurement model of agent interaction, a user can browse the catalogues of several vendors rather than simply using the client application provided by the vendor. He or she has the freedom to do so by dispatching an itinerant agent to forage the vendor's servers for information relevant to a purchase. The user can then retrieve this information in the desired manner. The client can also create proxy agents at this AMP and send them messages via the *Communication Portals*. The agents will receive these messages or user requests can then return the appropriate information to the user in the most suitable form.

7.3 Aggregate Technical Advantages

In the previous subsection we presented several of the technical advantages provided by our framework. While some subset of these technical advantages may be present in other solutions, collectively the framework provides an overwhelming technical advantage when compared to any currently posed solution. Our itinerant agent framework permits the secure interaction of agents written in multiple scripting languages, communicating in various agent communication languages, and traveling via multiple transport services in a heterogeneous network.

The framework also supports the use of standardized and non-standard distributed object-oriented frameworks, enabling the support of a variety of execution environments. This support allows client and service providers to write agents in a scripting language of their choice and to interface their legacy systems with proxy agents suited to any interfacing application. The introduction of the *Authentication Services* provides clients and service providers with access to a variety of authentication and encryption schemes and permits them to use the schemes in any manner suitable for their use.

7.4 Summary and Future Work

In this paper we presented an overview of a prototype framework for the support of itinerant agents. This framework provides a secure facility that can support interaction between agents requiring diverse execution environments, communicating via multiple languages, and traveling via multiple transport services. We motivated this presentation by discussing, in general, several examples of the use of agents and, in detail, an example of a travel reservation scenario.

This initial work addressed several major issues in the area of itinerant agents. While there are some issues that have not been addressed, they are being actively pursued and will be presented in subsequent publications. Several of these future topics have been presented in Section 4.1, including graphical user interfaces for launching and receiving agents, static agent support, agent status management, and virus control services. Another topic of interest to us is the support of electronic cash services. We believe such services would be of key importance in making network services truly viable.

Acknowledgements

The authors wish to acknowledge the many people who stimulated and contributed to the discussions that resulted in this work. In particular we would like to thank Stephen Brady, Jeff Kephart, Steve White, Robin Williamson, and the OREXX team at Endicott.

References

[1] Emergent Phenomena in Distributed Systems. *http://www.research.ibm.com/massdist*.

[2] Middleware to go mobile. *Computerworld*, 29(4), January 1995.

[3] J. K. Boggs. IBM Remote Job Entry Facility: Generalized Subsystem Remote Job Entry Facility. *IBM Technical Disclosure Bulletin,* 752, August 1973.

[4] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions). *Internet RFC 1521*, 1993.

[5] H. Chalupsky, T. Finin, R. Fritzson, D. McKay, S. Shapiro, and G. Wiederhold. An overview of KQML: A knowledge query and manipulation language. Technical report, April 1992.

[6] D. M. Chess, C. G. Harrison, and A. Kershenbaum. Mobile agents: Are they a good idea? *IBM Research Report, RC 19887*, October 1994.

[7] S. Chokhani. Toward a national public key infrastructure. *IEEE Communications Magazine*, 32(9):70–74, September 1994.

[8] F. Cohen. Computer Viruses: Theory and Experiment. *Computers and Security* 6:22-35, 1987.

[9] L. Hoffman (ed). *Rogue Programs: Viruses Worms and Trojan Horses*. Van Nostrand Reinhold, New York, 1990.

[10] M. Crowley-Milling et al. The Nodal System for the SPS. CERN, 78-87, 1978.

[11] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM '94)*, ACM Press, November 1994.

[12] T. Finin and others: External Interfaces Working Group of the DARPA Knowledge Sharing Effort. Specification of the KQML Agent-Communication Language plus example agent policies and architectures. Technical report, Working paper, June 1993.

[13] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format Version 3.0 Reference Manual. Technical report, Stanford University, Technical Report Logic-92-1, January 1992.

[14] Object Management Group. Common Object Request Broker Architecture and Specifications. *Document number 91.12.1*, 1(1).

[15] Tom R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[16] Tom R. Gruber. Ontolingua overview (World Wide Web home page). Technical report, Stanford University, Knowledge Systems Laboratory, 1995. Hypertext document on World-Wide-Web: URL http://www-ksl.stanford.edu/knowledge-sharing/ontolingua/index.html.

[17] L. J. Haisting. *EDI: A New Way of Doing Business*. St. Paul Software, St. Paul, MN, 1993.

[18] J. Kephart, T. Hogg, and B. Huberman. Collective behavior of predictive agents. *Physica D*, 42, 1990.

[19] J. Kephart, T. Hogg, and B. A. Huberman. Can predictive agents prevent chaos? In *Economics and Cognitive Science*, Pergamon Press, Oxford, 1991.

[20] J. Linn, S. Kent, D. Balenson, and B. Kaliski. Privacy enhancement for internet electronic mail: Parts i-iv. *Internet RFC 1421-1424*, 1993.

[21] Sun Microsystems. The HotJava Browser: a white paper. *White Paper*, 1995.

[22] R. Neches, R. Fikes, P. Patel-Schneider, R. Patil, T. Senator, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3), September 1991.

[23] J. K. Ousterhout. *TcL and the Tk toolkit*. Addison-Wesley Publication Company, 1994.

[24] E. Palmer. An Introduction to Citadel – a secure crypto coprocessor for workstations. In *IFIP SEC'94 Conference*, Curacao, Dutch Antilles, May 1994.

[25] R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA Knowledge Sharing Effort: Progress Report. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, San Francisco, CA., November 1992.

[26] J. B. Postel. SMTP (Simple Mail Transport Protocol). *Internet RFC 821*, 1982.

[27] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key crypto-systems. *CACM*, 21(2), 1978.

[28] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. The MIT Press, 1994.

[29] J. E. White. Telescript Technology: The Foundation for the Electronic Marketplace. General Magic Inc., Mountain View, CA, 1994.