Scripting Mobile Agents to Support Cooperative Work in the 21st Century

Seng Wai Loke
CRC for Enterprise Distributed Systems Technology
Monash University, Caulfield VIC 3145, Australia
swloke@dstc.monash.edu.au

Arkady Zaslavsky, Brian Yap, Joseph Fonseka
School of Computer Science and Software Engineering
Monash University, Caulfield VIC 3145, Australia
Arkady.Zaslavsky@monash.edu.au, brian11@hotmail.com, ruki@mbox.com.au

ABSTRACT

This paper discusses a scripting language approach to developing mobile agent applications that support enterprise work. We view an agent's itinerary describing which tasks to be performed when and at which location as a script glueing the tasks of the agents together in a (possibly) complex way. We present the ITAG (ITinerary AGent) scripting language which is based on the notion of the itinerary. We also discuss the enterprise-wide infrastructure needed for executing ITAG scripts, describe our current Web-based implementation and illustrate our approach with scripts for simple workflows, parallel processing, and information gathering, We also consider mobile agents for building applications over dynamically formed ad hoc virtual communities, and peer to peer applications.

1. INTRODUCTION

For building distributed applications, the evolution of distributed computing paradigms have taken us to the concept of mobile agents [5]. Mobile agents can be regarded as software components which on their own volition, instructed, or invited can move from one host to another to perform computations. Not just code but computational state (to an extent) can be transported from host to host. As yet, most of the applications mobile agents have been used for can be done in a traditional way (e.g., using RPCs). However, mobile agents can not only be used for what RPCs have been used for but also offer a number of advantages [3] such as reduction of bandwidth utilization by moving computation to where the data is, reduction in total completion time of tasks by encapsulating multiple queries (e.g. to databases) in an agent, exploitation of computational resources by moving computation to idle resources, and flexible dynamic deployment of components by moving code (in the agent) to where they are needed as needed. For mobile environments, agents can be launched to perform tasks without maintaining an expensive unreliable wireless network connection. Recent work has begun to use mobile agents for families of applications in the area of Computer Supported Collaborative Work (CSCW), and e-Business including workflow, efficient database querying, virtual enterprises, supply chain management, parallel computing over networked desktops, and electronic auctions (B2C or B2B) (e.g., see [6] and [10]).

As developments of distributed systems technology continue, we note three related technological trends impacting work in the present and for the future:

- **The networked enterprise**: The computational infrastructure of enterprises and businesses are becomingly increasingly pervasive and geographically distributed. Nodes can be of a variety of devices with widely varying memory, CPU speed and connectivity. Connectivity can span the enterprise and even cross enterprise boundaries.
- **Dynamic ad hoc virtual communities over wired and wireless environments**: An enterprise can be loosely viewed as a community which has clear boundaries, a long life time, and a clear purpose. In recent years, we are witnessing a proliferation of digital communities formed ad hoc and virtually regardless of geographical locality or induced by geographical locality. Such communities might involve a small or huge number of people, can last from several seconds to years, and might have boundaries that change frequently. With ubiquitous computing and networking, we are seeing wireless LANs and WANs of varying granularities connecting not only computers but computer-controlled networked appliances and embedded systems. Such wireless technologies enable wireless virtual communities [7] to be formed anytime and anywhere. We will see the car's wireless LAN, the home's wireless LAN, the office's wireless LAN, and beyond Internet cafes, restaurants could be offering network services via their wireless LANs. The shopping center in which the restaurant resides could offer their wireless information services. Wireless con-

nectivity can be offered at conferences or stadiums just for the period of the event. In addition, Bluetooth networking[1] enables ad hoc wireless networks to be formed just when several mobile devices are close enough to each other. Such LANs become a means to not just access wired neighbourhoods, global virtual communities and electronic marketplaces, but also to join (or form) a temporary ad hoc community to exchange or utilise resources. Many of such digital communities are characterised by their dynamicity (e.g., membership and resources within them can change rapidly). Communities themselves can be formed or dissolved rapidly, and so can connectivity across communities. With such work communities, computer supported cooperative work could happen anywhere and anytime, not constrained within the walls of offices, or working hours.

- **Peer to peer computing**: Peer to peer computing[2] has recently been given tremendous attention by technologists, businesses, and trend watchers. Peer networks have enormous potential for providing efficient CSCW via direct interaction by bypassing the single point of control and centralization of the server.

  In the peer to peer model, each peer has the same capability and responsibility. Unlike the client server model, communication is symmetric and a peer acts as both a client and a server. Peers share or exchange computer resources and services by direct exchange, i.e. there is a decentralization away from heavy-weight servers to equal-weight peers.

In this paper, we introduce a scripting language approach to developing mobile agent applications in the enterprise, in ad hoc virtual communities, and peer to peer computing. In the scripting approach [9], a scripting language is used to glue components together to assemble an application rather than programming an application from scratch. In contrast with imperative system programming languages (e.g., C, Java, etc) which were designed for building data structures and algorithms from scratch, scripting languages are designed for connecting components together. These components are assumed to already exist, and might be programmed in one or more system programming languages. A scripting language provides rapid application development and are typically typeless to simplify the connecting of components.

Our scripting language is based on the concept of the agent itinerary. An agent's itinerary describes which actions (or tasks) are to be performed when and at which location (e.g. which host), i.e. an agent's itinerary glues the actions of the agent in a (possibly) complex way while each action at a location might involve complex algorithms and data structures. Since problems suitable for the scripting language approach are those of a glueing nature where the complexity is in the connections, and problems suitable for the system programming language approach are those where the complexity is in the algorithms and data structures, we adopt the approach of using a scripting language for the itinerary and a system programming language for actions. This also encourages separation of concerns to simplify mobile agent programming since the mobility aspect of agents are abstracted away from code details implementing the computations the agents are to perform on hosts.

As proposed in [11], a scripting language should closely match the nature of the problem in order to minimize the linguistic distance between the specification of the problem and the implementation of the solution, thereby resulting in cost reductions and greater programmer productivity. Our itinerary scripting language provides a higher level of abstraction, and economy of expression for mobility behaviour: the programmer expresses behaviour such as "move agent A to place p and perform action a" in a simple direct succinct manner without the clutter of the syntax of a full programming language. Our approach not only encourages reuse of components (implementing actions), for example, in different itineraries, but also facilitates identification and reuse of patterns in mobility behaviour. Also, since the itinerary provides a bird's eye view of the mobile agent application, the itinerary scripting language enables high-level design of the application to be represented.

In the following section, we first present a conceptual architecture of an enterprise-wide infrastructure over which mobile agents can roam. Then in §3, we present our itinerary scripting language, and in §4, present examples of enterprise applications scripted in our language, and consider mobile agents for CSCW in ad hoc virtual communities and peer to peer computing. We conclude in §5.

2. A CONCEPTUAL ARCHITECTURE OF MOBILE AGENTS IN THE ENTERPRISE

Because ITAG scripts involve mobile agents, an enterprise-wide infrastructure is needed to execute ITAG scripts effectively and usefully. Mobile agents run within agent server programs which we call *places* that receive agents and execute them. Two places might be on the same or different machine. When at a place, an agent can utilise the resources at the place. An enterprise needs to be equipped with a network of places interfaced to resources or people so that agents can utilize these resources (e.g., a database) and interact with people (e.g., to ask for information) in accomplishing actions. A place is also where a user can interact with agents and issue commands to launch agents (or applications).

Figure 1 depicts an agent infrastructure consisting of a network of places interfaced to resources such as a database management system (DBMS), an ERP system and people. Such interfacing might involve wrapping resources with the ability to interact with agents. Note that such an infrastructure is mainly useful for large organizations consisting of distributed resources. Without such an infrastructure and interfacing to resources, the actions agents can do will be limited.

It is possible for the infrastructure to be organized into security domains as proposed in [2], each domain consisting of a set of places and agents have to obtain authorization to enter a domain. Also, the infrastructure of two organizations might be connected to allow agents from one organization to move into another organization as in the case of inter-

[1]See http://www.bluetooth.com
[2]See http://www.oreillynet.com/p2p

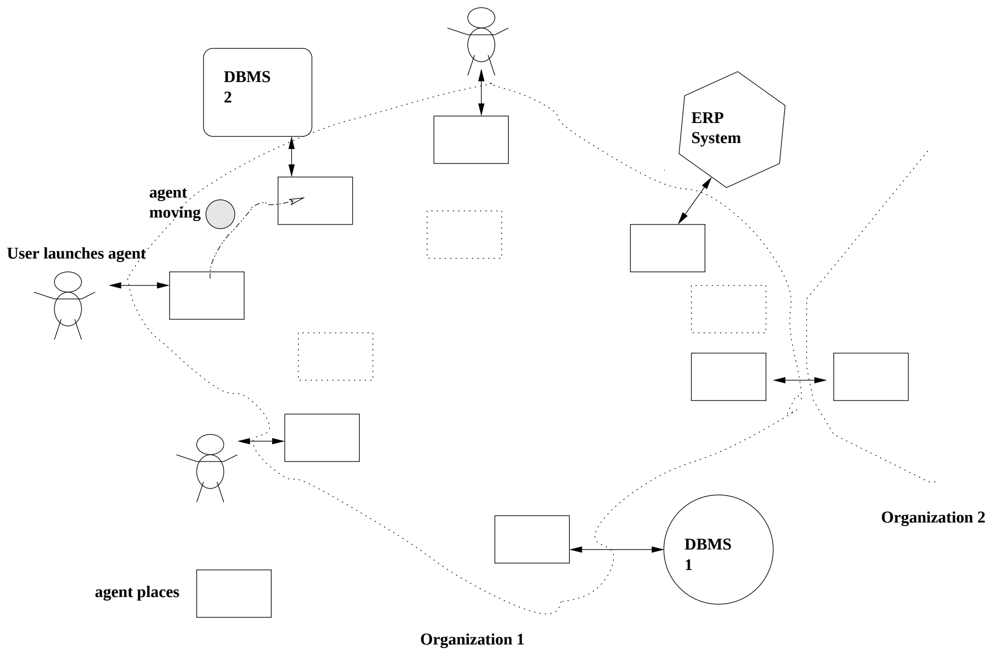


Figure 1: A sketch of an agent infrastructure.

organizational workflows. The architectural details of such an infrastructure is outside the scope of this paper.

From a place, a user can configure and script applications, and launch these agents into the infrastructure.

3. ITAG: THE ITINERARY SCRIPTING LANGUAGE

Our itinerary scripting language which we call ITAG is based on the itinerary algebra developed in [8]. For completeness, we outline the algebra below. In [8] are more details of their operational semantics and algebraic properties.

3.1 An Algebra of Itineraries

We assume an object-oriented model of agents (e.g., with Java in mind), where an agent is an instance of a class given roughly by:
mobile agent = state + action + mobility

State refers to an agent's state (values of instance variables) possibly including a reflection of the agent's context. Action refers to operations the agent performs to change its state or that of its context. Mobility comprises all operations modifying an agent's location, including moving its state and code to other than the current location. While mobility assumes that an agent moves at the agent's own volition, the itineraries may be viewed as a specification or plan of agent movements.

We assume that agents have the capability of *cloning*, that is, creating copies of themselves with the same state and code. Also, agents can communicate to synchronize their movements, and the agent's code is runnable in each place it visits.

Let $\mathbf{A}$, $\mathbf{O}$ and $\mathbf{P}$ be finite sets of agent, action and place symbols, respectively. Itineraries (denoted by $\mathcal{I}$) are now formed as follows representing the null activity, atomic activity, parallel, sequential, nondeterministic, conditional nondeterministic behaviour, and have the following syntax:

$$\mathcal{I} ::= \mathbf{0} \mid A_p^a \mid (\mathcal{I} \parallel_{\oplus} \mathcal{I}) \mid (\mathcal{I} \cdot \mathcal{I}) \mid (\mathcal{I} \mid \mathcal{I}) \mid (\mathcal{I} :_{\Pi} \mathcal{I})$$

where $A \in \mathbf{A}$, $a \in \mathbf{O}$, $p \in \mathbf{P}$, $\oplus$ is an operator which, after a parallel operation causing cloning, recombines an agent with its clone to form one agent, and Π is an operator which returns a boolean value to model conditional behaviour. We specify how $\oplus$ and Π are used but we assume that their definitions are application-specific.

We assume that all agents in an itinerary have a starting place (which we call the agent's *home*) denoted by $h \in \mathbf{P}$.

Given an itinerary I, we shall use $agents(I)$ to refer to the agents mentioned in I.

- *Agent Movement* (A_p^a). A_p^a means "move agent A to place p and perform action a". This expression is the smallest granularity mobility abstraction. It involves one agent, one move and one action at the destination. The underlying mobility mechanisms are hidden. So are the details of the action which may change the agent state or the context in which it is operating at the destination place:

  $$a : states(A) \times states(p) \rightarrow states(A) \times states(p)$$

  In our agent model, each action is a method call of the class implementing A. The implementation must check that a is indeed implemented in A.

  $\mathbf{0}$ represents, for any agent A, the empty itinerary A_{here}^{id}, where the agent performs the identity operation $id \in \mathbf{O}$ on the state at its current place *here*.

- *Parallel Composition ("$\|$").* Two expressions composed by "$\|$" are executed in parallel. For instance, $(A_p^a \parallel B_q^b)$ means that agents A and B are executed concurrently. Parallelism may imply cloning of agents. For instance, to execute the expression $(A_p^a \parallel A_q^b)$, where $p \neq q$, cloning is needed since agent A has to perform actions at both p and q in parallel. In the case where $p = q$, the agents are cloned as if $p \neq q$. In general, given an itinerary $(I \parallel J)$ the agents in $agents(I) \cap agents(J)$ are cloned and although having the same name are different agents.

  When cloning has occurred, decloning is needed, i.e. clones are combined using an associated application-specific operator (denoted by $\oplus$ as mentioned earlier). For example, given the expression $(A_s^d \parallel A_t^e) \cdot A_u^f$ and suppose that after the parallel operation, the configuration has clones. Then, decloning is carried out before continuing with A_u^f. The resulting agent from decloning resides in the original place (in this case s). We have associated decloning with this operator instead of the sequential operator as in [8] having found this to be more intuitive and natural.

- *Sequential Composition ("$\cdot$").* Two expressions composed by the operator "$\cdot$" are executed sequentially. For example, $(A_p^a \cdot A_q^b)$ means move agent A to place p to perform action a and then to place q to perform action b. Sequential composition is used when order of execution matters. In the example, state changes to the agent from performing a at p must take place before the agent goes to q.

  Sequential composition imposes synchronization among agents. For example, in the expression $(A_p^a \parallel B_q^b) \cdot C_r^c$ the composite action $(A_p^a \parallel B_q^b)$ must complete before C_r^c starts. Implementation of such synchronization requires message-passing between agents at different places or shared memory.

- *Independent Nondeterminism ("$|$").* An itinerary of the form $(I \mid J)$ is used to express nondeterministic choice: "I don't care which but perform one of I or J". If $agents(I) \cap agents(J) \neq \emptyset$, no clones are assumed, i.e. I and J are treated independently. It is an implementation decision whether to perform both actions concurrently terminating when either one succeeds (which might involve cloning but clones are destroyed once a result is obtained), or trying one at a time (in which case order may matter).

- *Conditional Nondeterminism ("$:$").* Independent nondeterminism does not specify any dependencies between its alternatives. We introduce conditional nondeterminism which is similar to short-circuit evaluation of boolean expressions in programming languages such as C.

  We first introduce *status flag* and *global state function*. A status flag is part of the agent's (say, A's) state, written as $A.status$. Being part of the state, $A.status$ is affected by an agent's actions. $A.status$ might be changed by the agent as it performs actions at different places. A global state function Π need not be defined in terms of status flags but it is useful to do so. For example, we can define Π as the conjunction of the status flags of agents in a set Σ: $\Pi(\Sigma) = \bigwedge_{A \in \Sigma} A.status$. We can view Π as producing a global status flag. From the implementation viewpoint, agents in Σ must communicate to compute Π.

  An itinerary of the form $I :_\Pi J$ means first perform I, and then evaluate Π on the state of the agents. If Π evaluates to true, then the itinerary is completed. If Π evaluates to false, the itinerary J is performed (i.e., in effect, we perform $I \cdot J$).

  The semantics of conditional nondeterminism depends on some given Π, expressed by writing "$:_\Pi$".

Symbol	ASCII	Controlled English
A_p^a	`[A,p,a]`	`move` A `to` p `do` a
$\cdot$	`.`	`then`
$:_\Pi$	`:{op}`	`otherwise using op`
$\|$	`|`	`or`
$\|_\oplus$	`#{op}`	`in parallel with using op`

Table 1: Translations.

3.2 An Example: Voting

An agent V, starting from home, carries a list of candidates from host to host visiting each voting party. Once each party has voted, the agent goes home to tabulate results (assuming that home provides the resources and details about how to tabulate), and then announces the results to all voters in parallel (and cloning itself as it does so). Assuming four voters (at places p, q, r, and s), *vote* is an action accepting a vote (e.g., by displaying a graphical user interface), *tabulate* is the action of tabulating results, and *announce* is the action of displaying results, the mobility behaviour is as follows:
$V_p^{vote} \cdot V_q^{vote} \cdot V_r^{vote} \cdot V_s^{vote} \cdot V_h^{tabulate} \cdot (V_p^{announce} \parallel V_q^{announce} \parallel V_r^{announce} \parallel V_s^{announce})$

Note that we leave out brackets due to the associativity of the binary operators. This example can be generalized to n voters.

3.3 Pragmatic Considerations

This subsection outlines pragmatic considerations in our current implementation of the above itinerary scripting language involving syntax, semantics and execution environment.

3.3.1 Implementation Syntax: ASCII and Controlled English Representations

To allow the programmer to type the itinerary expressions into the computer, we provide an ASCII syntax and a Controlled English version. The translations are given in Table 1.

When the operators are used without `op`, we assume a pre-specified system default one, i.e. `using op` is an optional clause.

Hence, $A_p^a \cdot A_q^b \cdot A_r^c$ can be described as follows: "(move A to a do p) then (move A to b do q) then (move A to c do r)."

Apart from the above basic elements of the language, we define the following five phrases that map down to more complex expressions:

1. A^a_h is translated as `return` A `do` a.
2. $A^a_p \cdot A^a_q \cdot A^a_r \cdot A^a_s$ is translated as `tour` A `to` p, q, r, s `in series do` a.
3. $A^a_p || A^a_q || A^a_r || A^a_s$ is translated as `tour` A `to` p, q, r, s `in parallel do` a.
4. $A^a_p | A^a_q | A^a_r | A^a_s$ is translated as `tour` A `to one of` p, q, r, s `do` a.
5. $A^a_p : A^a_q : A^a_r : A^a_s$ is translated as `tour` A `if needed to` p, q, r, s `do` a. Similarly, we also have $A^a_p :_\Pi A^a_q :_\Pi A^a_r :_\Pi A^a_s$ translated as `tour` A `if needed to` p, q, r, s `do` a `using` Π.

So, for example, the voting itinerary can be described as follows:

```
((move V to p do vote)
then (move V to q do vote)
then (move V to r do vote)
then (move V to s do vote)
)
then (return V do tabulate)
then
((move V to p do announce)
in parallel with (move V to q do announce)
in parallel with (move V to r do announce)
in parallel with (move V to s do announce)
)
```

Using the phrases, the voting itinerary can also be described more succinctly as follows:

```
(tour V to p,q,r,s in series do vote)
then (return V do tabulate)
then (tour V to p,q,r,s in parallel do announce)
```

In our current implementation, the actions of agents are names of methods and places have the form:

```
<hostname>:<portnumber>/placename
```

or

```
<hostname>/placename
```

with a default port number. For example, for the host

```
nemesis.csse.monash.edu.au
```

and name `a`, we have:

```
nemesis.csse.monash.edu.au/a
```

Such long place names might make scripts unwieldy. Predeclared abbreviations might then be used.

3.3.2 Implementation Semantics

Our current implementation is in the Java programming language and is built on top of the Grasshopper mobile agent toolkit.[3] We follow the operational semantics outlined above as far as possible. We use Java multi-threading to implement the parallel operator.

For non-determinism, we are implementing it by concurrency as well but retaining the result from the fastest branch only and discarding the results from other branches. In doing this, we assume that side-effects from other branches are not significant. An alternative implementation is to first choose at random (e.g., using a random number generator) a branch to follow, and then execute only that branch. This avoids side-effects but is only pseudo-random. In general, it is difficult to achieve true non-determinism.

3.3.3 Web-Based Execution Environment

We intend our execution environment for the ITAG language to be Web-based. Figure 2 shows the Web-based system interfaced to the Grasshopper system. In our current implementation, the user first types in itinerary scripts into an applet (running in a Web browser). Then, the itinerary script is parsed into a binary tree representation and executed by an interpreter. Execution is as follows: the interpreter translates the actions specified in the script into commands which are then forwarded (via the communication handler) from the applet to a master Grasshopper agent which, in turn, sends commands (via a proxy) to the slave Grasshopper agents which are initially at a place (the home). These agents on receiving the commands are then launched into the network of places to do their work. Note that in Grasshopper, each place is called an *agency* and each agency registers itself with a directory facility called the *region*. Only three agencies are shown in the figure - there could be many more.

The code for the actions of agents at a place and operations associated with parallelism (i.e., combining agents) and conditional non-determinism are pre-written, stored on a Web server (which we call the *code server*), and pulled in on-demand. For example, if the agent is to move to a place p to do action a, the agent first moves to p and pulls the code (actually, a class with method a) for action a from the Web server to p, and then executes it storing the results in its state. A user, or a programmer on a user's behalf, can write his or her own actions and store it with the code server for use in scripts.

Figure 3 shows a screen dump of the ITAG system running as an applet in Internet Explorer. The applet displays the set of available agents, places, actions and operations to aid the writing of scripts. The "command" is the controlled English syntax which gets parsed and converted into the ASCII syntax. The right hand side of the screen dump shows the

[3]See http://www.grasshopper.de

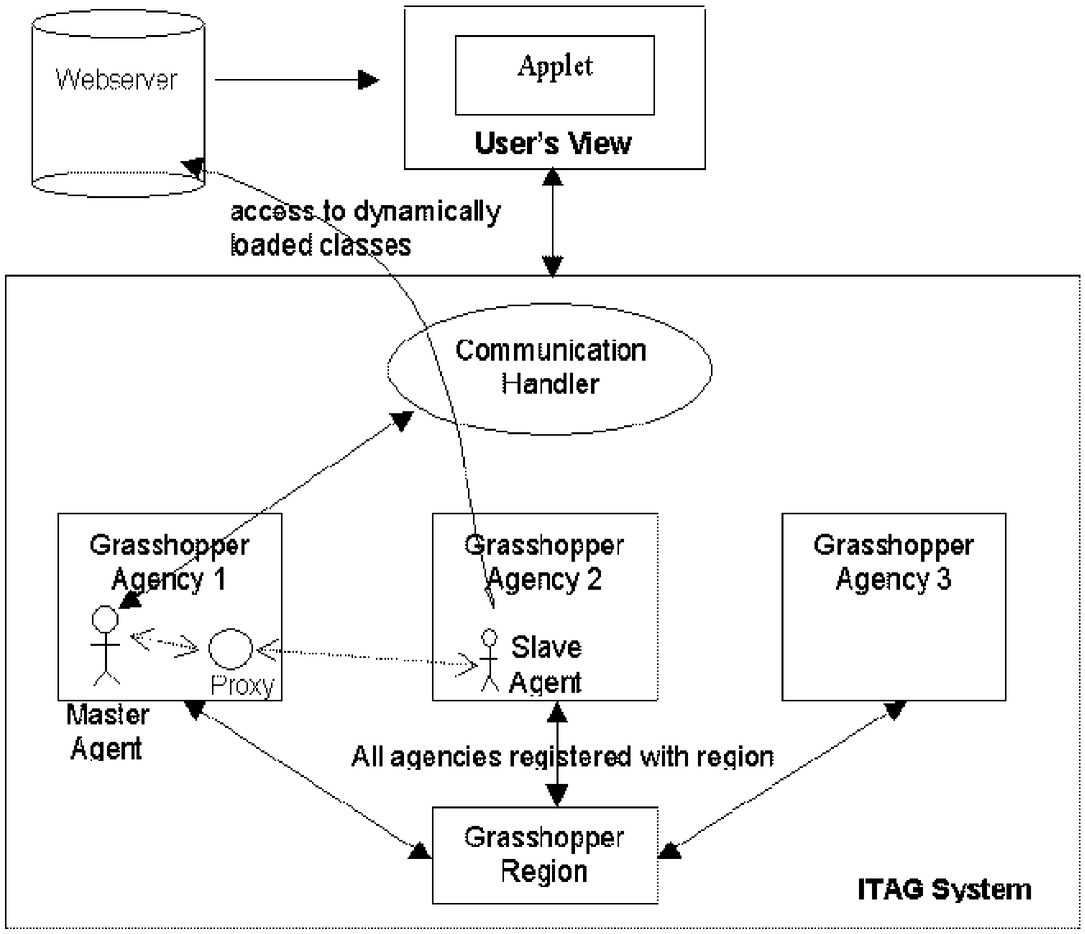


Figure 2: Web based ITAG system.

execution trace of an agent collecting information from three places in parallel. Note the interleaving of trace messages which indicates the parallelism.

Web browsers are a common means of accessing an enterprise's Intranet, and our use of a Web applet provides a view that these mobile agent applications are just another service of the enterprise's Intranet. Launching and controlling mobile agents from Web browsers have been supported in the Fiji extension of the Aglets toolkit [4], and W-MAP [1], though they do not have a scripting language such as ours.

4. APPLICATIONS

This section discusses applications of ITAG scripting. We first give several examples of applications in the enterprise, and then, propose the use of mobile agents in ad hoc virtual communities and peer to peer computing.

4.1 Mobile Agents for the Enterprise

We aim mainly for lightweight applications (e.g., ad hoc workflows), lightweight in the sense that they can be quickly scripted as long as the required actions code are available from the code server. We give four examples of such applications in this section.

4.1.1 Meeting Scheduling

Consider a meeting scheduling application whose itinerary was first described in [8]. This is a two phase process: (1) Starting from home, the meeting initiator sends an agent which goes from one participant to another with a list of nominated times. As each participant marks the times they are not available, the list of nominated times held by the agent shortens as the agent travels from place to place. After visiting all places in its itinerary, the agent returns home. (2) At home, the meeting initiator selects a meeting time from the remaining unmarked times and informs the rest. We assume that the code for the following actions have been written and stored on the code server: `ask` is an action which displays unmarked nominated times to a participant and allows a participant to mark times he/she is unavailable, `finalize` allows the meeting initiator to select a meeting time from the remaining unmarked times, and `inform` presents the selected meeting time to a participant. Then, if the agent is named `M` with four participants (excluding the initiator) reachable from the places
`www.mycompany.com/paul` (abbreviated to `p`),
`www.mycompany.com/john` (abbreviated to `j`),
`www.mycompany.com/rachel` (abbreviated to `r`),
and `www.mycompany.com/steiner` (abbreviated to `s`),
the ITAG script is:

```
(tour M to p, j, r, s in series do ask)
then (return M do finalize)
then (tour M to p, j, r, s in parallel do inform)
```

Note that the expression of mobility is separated from the coding of the three actions. One of the advantages of this approach to meeting scheduling is that the participants are prompted for action and are informed of the final meeting time automatically. The system is pro-active as oppose to a passive system such as a Web site where participants have to visit several times in order to organize a meeting.

4.1.2 Parallel Processing

We can also write a script for parallel processing over a network of places within the Intranet. Assuming that the above four available places (assuming they are located on different hosts for parallelism to be exploited), the agent is `D`, and the action each agent i has to perform (`job`i), and the operation `combine_results` to combine results have been coded. Then, the following ITAG script does the job:

```
((move D to p do job1)
in parallel with (move D to j do job2)
in parallel with (move D to r do job3)
in parallel with (move D to s do job4)
)
then (return D do combine_results)
```

Note that if some place (say `r`) becomes unavailable and another place is available, the above script can be easily changed. Similarly, the actions can be changed if different jobs are required reusing a similar itinerary.

4.1.3 Information Gathering

This final example involves an information gathering application. The aim is to search three data resources sequentially stopping as soon as the required information is found. Assuming that the data resources are accessible from places `db1`, `db2`, and `db3`, the agent is `I`, `search` is the action of searching a data resource, and `found` determines if the required information is found, we can use the following script:

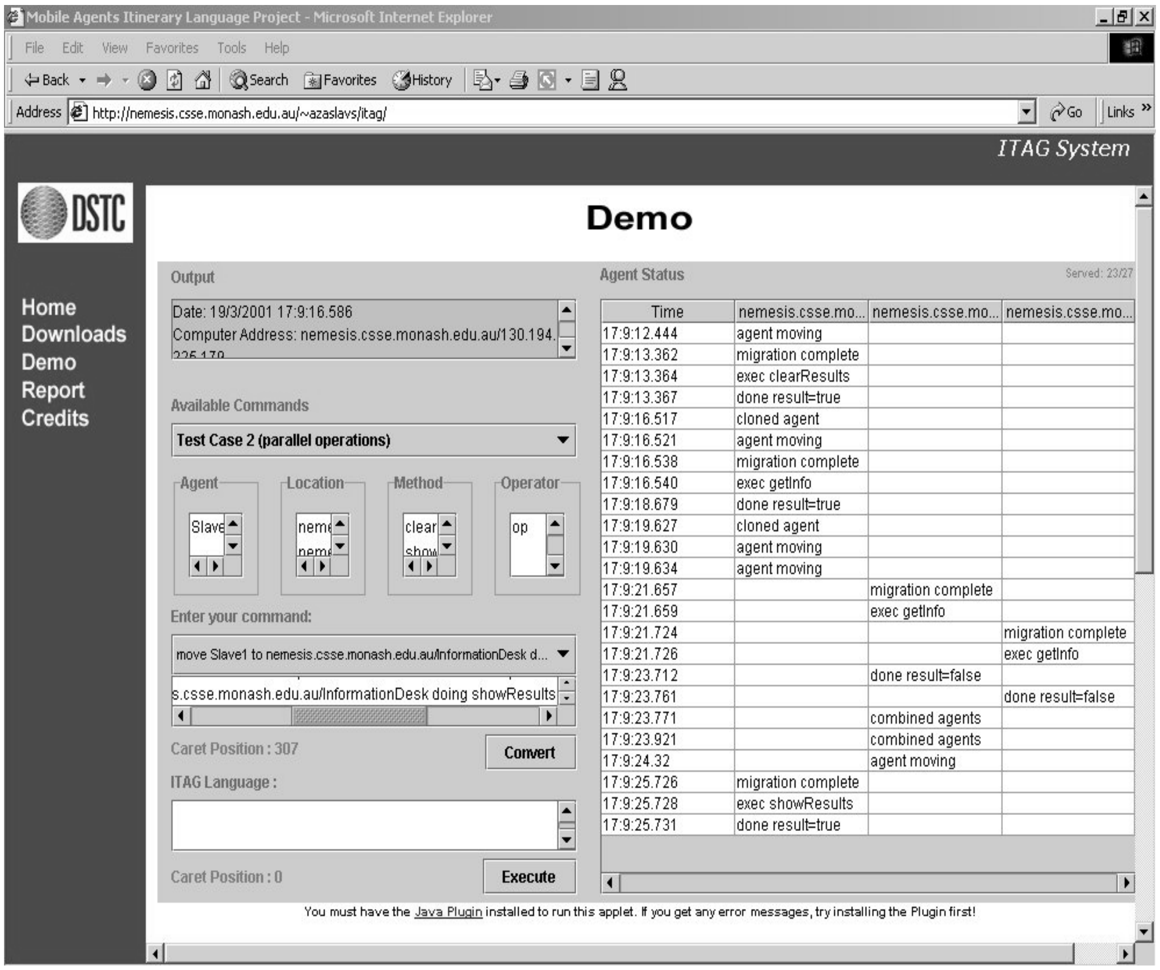
```
tour I if needed to db1, db2, db3 do search
using found
```

4.1.4 Discussion

As shown in the above examples, ITAG scripting has the potential to simplify the programming of complex applications greatly but relies on an existing agent infrastructure over which agents can roam and assumes that action code are available from the code server. Such an infrastructure has been proposed in [6] and in [10], an infrastructure has been built which supports distributed database access and parallel computing with mobile agents. Among the actions agents can do are access Web databases, provide personalized views of databases, and support data warehousing. However, they do not use a scripting approach for their applications.

Other applications can be considered such as extending services or upgrading software throughout the enterprise by agents moving components to where needed when needed. Moreover, such components might encapsulate protocols to enable applications on distributed nodes to communicate via a specialized protocol.

In our scripting approach, extensibility and reuse are accommodated. Action code can be added to the code server to enable scripts of greater functionality. The same action code can be reused in different scripts (provided the action code is appropriately written with reuse in mind), and conversely, scripts themselves can be reused while the action code changes. For example, if the database system is replaced by a new one, the same script can be used but



Address: http://nemesis.csse.monash.edu.au/~azaslavs/itag/

Date: 19/3/2001 17:9:16.586
Computer Address: nemesis.csse.monash.edu.au/130.194.226.170

Available Commands: Test Case 2 (parallel operations)

Agent: Slave | Location: neme | Method: clear, show | Operator: op

Enter your command: move Slave1 to nemesis.csse.monash.edu.au/InformationDesk d...

s.csse.monash.edu.au/InformationDesk doing showResults

Caret Position : 307

Caret Position : 0

Time	nemesis.csse.mo...	nemesis.csse.mo...	nemesis.csse.mo...
17:9:12.444	agent moving		
17:9:13.362	migration complete		
17:9:13.364	exec clearResults		
17:9:13.367	done result=true		
17:9:16.517	cloned agent		
17:9:16.521	agent moving		
17:9:16.538	migration complete		
17:9:16.540	exec getInfo		
17:9:18.679	done result=true		
17:9:19.627	cloned agent		
17:9:19.630	agent moving		
17:9:19.634	agent moving		
17:9:21.657		migration complete	
17:9:21.659		exec getInfo	
17:9:21.724			migration complete
17:9:21.726			exec getInfo
17:9:23.712		done result=false	
17:9:23.761			done result=false
17:9:23.771		combined agents	
17:9:23.921		combined agents	
17:9:24.32		agent moving	
17:9:25.726	migration complete		
17:9:25.728	exec showResults		
17:9:25.731	done result=true		

You must have the Java Plugin installed to run this applet. If you get any error messages, try installing the Plugin first!

Figure 3: Screen dump of ITAG system.

the action code implementing how the agent queries the database system can be changed. Moreover, as far as the scripts are concerned, the action code need not be written in Java but any other full programming language (e.g. C or a rule-based language such as Prolog) can be employed. Moreover, although we have used Grasshopper in our current implementation, note that the scripts are independent of the underlying agent system.

We have not been concerned with the execution performance of ITAG scripts since this relies very much on the underlying agent toolkit and overall execution time is dominated by action execution.

We have not dealt with the issue of security in depth since we assume that ITAG scripts are executed within a secure Intranet of the enterprise and that action code are programmed by parties friendly to the enterprise. We also rely on the security facilities provided by the underlying mobile agent toolkit.

4.2 Mobile Agents for Ad Hoc Virtual Communities

Mobile agents can be a useful paradigm for building and deploying CSCW applications over ad hoc virtual communities. For example, applications usually possible within an enterprise such as ad hoc workflows, information gathering and monitoring, or parallel processing can be made possible within such ad hoc communities. Redeployment of components by moving the agent (with such components) to where needed when needed is particularly useful for dynamically formed ad hoc communities where long set up times are inconvenient, lifetime is relatively short, and connectivity and resources might vary. Such components can also encapsulate protocols to enable applications on distributed nodes to communicate via a specialized protocol. We do not claim that mobile agents is the only paradigm to use but contend that it is one of the paradigms which can be usefully employed.

We believe that a scripting approach is suitable for developing mobile agent applications in such communities by allowing applications to be assembled quickly and easily, changes to be made more easily, and diversity to be accommodated (e.g., actions within an itinerary can in principle be programmed in several different programming languages). Our language is also interpreted, and thus, immediately executable without long compile times.

Moreover, we believe that a Web (or even WAP[4]) based approach is convenient not only for accessing such mobile agent applications but also for building communities of agent places. A Web site could provide a facility for registering new places which are added to the existing set of places. When new places appear (or disappear), the applets for the human participants are updated to show the list of available places.

4.3 Mobile Agents for Peer to Peer Computing

We think that mobile agents have much to contribute to peer to peer computing and to CSCW software. One model in which mobile agents can be applied in peer to peer computing is to treat an agent running at a place as a peer. Then, such mobile peers can be relocated from one machine to another to exploit resources. New peer to peer systems can be spawned and existing systems dynamically extended by sending agents (the peers or peer extensions) to the appropriate sites.

A network of peers can form the infrastructure over which mobile agents can roam. Mobile agents can be sent to interact with peers in a peer network. Instead of the default means in which query messages are routed through peer networks as in the Gnutella peer to peer file sharing system,[5] the agents can autonomously move through the peer network with their behaviour specified in itineraries. Each peer can determine how to forward its arriving agents (e.g., either according to an existing itinerary or overriding the itinerary with its own). Then, the itinerary script does not contain addresses or place names but uses peer identifiers. For example, a model of the enterprise can be used as a map of places to which agents can roam. Such a model need not necessarily be a low-level model of available hosts and places but might be a map of people and roles they play. Instead of place names (which identify agent servers), one could use the names of persons or the names of roles (e.g., CEO, Human Resource manager, administrator, etc) in itineraries. Person or role names dynamically resolve to the name of a place from which a person or the person playing the specified role can be reached. For instance, at one time, manager might resolve to a place running on a desktop but at another time (say, when the manager is out of the office) to a place running on a PDA.

Many peer to peer systems have their own peer identifiers. For example, the Internet messaging and chat system ICQ[6] uses its own ICQ numbers as peer identifiers. Itineraries could make use of such peer identifiers in place of hostname and port number (in place names). This would allow agents to be routed to the right people whichever host they are logged on to without the script writer's intervention.

5. CONCLUSIONS AND FUTURE WORK

We contend that a scripting approach is well-suited for developing mobile agent applications and presented ITAG based on the notion of the agent itinerary. We have also discussed the infrastructure for executing ITAG scripts, a Web-based implementation, and presented examples of scripts for ad hoc workflows, parallel processing, and information gathering. In addition, the uses of mobile agents for ad hoc dynamically formed communities and peer to peer computing have been briefly considered.

There are several promising directions for future work. Firstly, although our scripts provide agents with a degree of autonomy and flexibility in performing tasks via the nondeterminism and conditional nondeterminism operators, we aim to provide greater autonomy by allowing methods to generate (or choose) places instead of explicitly specifying places as

[4]Wireless Application Protocol. See http://www.wapforum.org

[5]See http://www.gnutella.co.uk

[6]See http://www.icq.com

in the current scripts. Such methods could be governed by a well-defined decision procedure so that there is a clear rationale (to the programmer) concerning which places will be generated (or chosen). However, there is a trade-off between autonomy and control - users might desire more predictable and controllable behaviour. Secondly, we would like to support scripts which involve multiple agents as is possible in theory in the itinerary algebra. Thirdly, building exception handling into the agent infrastructure will be important - some agent toolkits have exception handling features and reflecting these features up to the scripting level without cluttering scripts will be a challenge. Other interesting areas of work include connecting infrastructures across organizations to support inter-organizational information gathering and workflows, and to involve wireless connectivity among nodes. Scripts that cater for possible changes in connectivity will be needed.

6. ACKNOWLEDGEMENTS

The work reported in this paper has been funded in part by the Co-operative Research Centre Program through the Department of Industry, Science & Tourism of the Commonwealth Government of Australia.

7. REFERENCES

[1] M. Avvenuti, A. Puliafito, and O. Tomarchio. W-MAP: A Web Accessible Mobile Agent Platform. In *Proceedings of the Austrian-Hungarian Workshop on Distributed and Parallel Systems*, Budapest, Hungary, September 1998. Available at <http://sun195.iit.unict.it/Papers/dapsys98.ps.gz>.

[2] M. Cremonini, A. Omicini, and F. Zambonelli. Modelling Network Topology and Mobile Agent Interaction: an Integrated Framework. In *Proceedings of the 1999 ACM Symposium on Applied Computing*, pages 410–412, San Antionia, USA, Feb/Mar 1999. ACM Press.

[3] R. Gray, G. Cybenko, D. Kotz, and D. Rus. Mobile Agents: Motivations and State of the Art. In J. Bradshaw, editor, *Handbook of Agent Technology.* AAAI/MIT Press, 2000. Draft available as Technical Report TR2000-365, Department of Computer Science, Dartmouth College.

[4] IBM. Fiji - Running Aglets in Web Pages. (updated) 1999. Web page at <http://www.trl.ibm.co.jp/aglets/infrastructure/fiji/fiji.html>.

[5] D. Kotz and R. Gray. Mobile Code: The Future of the Internet. In *Proceedings of the Workshop on Mobile Agents in the Context of Competition and Cooperation at Autonomous Agents '99*, Seattle, U.S.A., May 1999. Available from <http://mobility.lboro.ac.uk/MAC3/>.

[6] S. Loke. An Overview of Mobile Agent Technology for Distributed Applications: Possibilities for Future Enterprise Systems. *Accepted and being revised for Informatica Journal*, 2001.

[7] S. Loke, A. Rakotonirainy, and A. Zaslavsky. An Enterprise Viewpoint of Wireless Virtual Communities and the Associated Uses of Software Agents. In *Internet Commerce and Software Agents.* Idea Group Publishing, 2001.

[8] S. Loke, H. Schmidt, and A. Zaslavsky. Programming the Mobility Behaviour of Agents by Composing Itineraries. In P. Thiagarajan and R. Yap, editors, *Proceedings of the 5th Asian Computing Science Conference (ASIAN'99)*, volume 1742 of *Lecture Notes in Computer Science*, pages 214–226, Phuket, Thailand, December 1999. Springer-Verlag.

[9] J. Ousterhout. Scripting: Higher Level Programming for the 21st Century. *IEEE Computer*, March 1998. Available at <http://www.scriptics.com/people/john.ousterhout/scripting.html>.

[10] G. Samaras, P. Evripidou, and E. Pitoura. A Mobile Agent Based Infrastructure for eWork and eBusiness Applications. In *Proceedings of the E-Business and E-Work Conference*, Madrid, Spain, October 2000.

[11] D. Spinellis and V. Guruprasad. Lightweight Languages as Software Engineering Tools. In *Proceedings of the USENIX Conference on Domain-Specific Languages*, California, U.S.A., October 1997.

[12] A. Tripathi, T. Ahmed, V. Kakani, and S. Jaman. Distributed Collaboration Using Network Mobile Agents. February 2000. Available at <http://www.cs.umn.edu/Ajanta/papers/asa-ma.ps>.