Introduction to Mobile Agents: Performance, Security, and Programming Examples

Bob Gray
Dartmouth College
Hanover, NH, USA
robert.s.gray@dartmouth.edu

ASA/MA 2000
Zurich, Switzerland, September 13-15
Acknowledgements

Researchers

• George Cybenko  
  Thayer School of Engineering
• David Kotz  
  Department of Computer Science
• Daniela Rus  
  Department of Computer Science

Staff Programmers

• Ron Peterson
• Arne Grimstrup

Funders

DARPA contract  
F30602-98-2-0107

ONR contract  
N00014-95-1-1204

DoD MURI  
(AFOSR contract  
F49620-97-1-03821)

AFRL/Rome  
contract  
F30602-98-C-0006
Read More About It

D’Agents

http://agent.cs.dartmouth.edu/

This Tutorial (with note pages)

http://agent.cs.dartmouth.edu/tutorials/

Other Mobile-Agent Systems

http://www.cetus-links.org/oo_mobile_agents.html
Roadmap

• Definitions and Myths

• Why Mobility?
  – Example Applications
  – Six Reasons for Mobility
  – Competing Approaches
  – The “Big Picture”

• Mobile-Agent Systems
  – Java-Based Systems
  – Other Single-Language Systems
  – Multiple-Language Systems
Roadmap

• D’Agents
  – Overview
  – Example: Writing a Tcl agent
  – Example: Writing a Java agent

• Security
  – D’Agents Security
  – Protecting a Group of Machines
  – Protecting an Agent

• The Future of Mobile Agents
What is a Mobile Agent?

Most general form of mobile code

- Process that migrates under its own control in a heterogeneous network

Often, but not always, has “agent” characteristics

- Autonomy, adaption, learning, …
Two Kinds of Mobile Agents

**Strong Mobility**
(data, code and control state)

```
foreach machine machineList {
    ...
    move (machine)
    ...
}
```

**Weak Mobility**
(data and code only)

```
proc doTask (arg) {
    ...
}
...
move (machine, doTask, arg)
```
The Mobility Space

Mobile Code

Mobile Control State

Client/Server

Mobile agents (weak), applets, servlets

Mobile agents (strong), some load-balancing systems

Some load-balancing systems
Weak or Strong Mobility?

- **Strong Mobility**
  - More convenient for the *agent programmer*
  - Subsumes weak mobility

- **Weak Mobility**
  - Sufficient for all but load-balancing applications
  - Well suited to the event-driven style of many agents
  - Much less work for the *system developer*
  - Supported by standard Java virtual machines
Myth #1: Every Agent is Mobile

Myth

Every agent should be mobile.

Reality

Large applications should use small mobile agents to accomplish narrowly focused tasks.
Myth #2: Every Mobile Agent Moves

Myth

A mobile agent should always move to access a needed resource.

Reality

A mobile agent should move if the resource can be accessed more efficiently locally.

This depends on

- Resource “Granularity”
- Application Task
- Current Network and Machine Conditions
Myth #3: No High-Level Operations

Myth

Mobile agents are better than adding a high-level operation to a service.

Reality

Mobile agents are better than invoking low-level operations from across the network.

– Do not use mobile agents if you have sufficient time, money and access to add the “right” high-level operation to the service.

– Do use mobile agents if you can not keep up with client needs.
Myth #4: Dynamic Code Deployment

Myth

Mobile agents are different than dynamic code deployment.

Reality

Mobile agents are a (very general) form of dynamic code deployment.

Two separate questions:

– When is dynamic code deployment a good idea?
– When are mobile agents the right programming model for dynamic code deployment?
Roadmap

• Definitions and Myths

• Why Mobility?
  – Example Applications
  – Six Reasons for Mobility
  – Competing Approaches
  – The Big Picture

• Representative Systems

• D’Agents

• Security

• The future
Application: Soldiers in the Field

HQ

Wired Network

BGW

New Spec
de

Wireless Network

13 Sep 2000

ASA/MA'00

15
Phone Call Monitoring

1. Agent jumps to BGW machine with query.
2. Agent gets names.
3. Agent jumps to phone machine.
4. Agent gets calls with right time/region and uses name list to “score” them.
5. Agent sends high-scoring calls back to analyst.

First name: Bob
Last name: Gray
Weight: 140
Height: 5’8”

Phone Call Database

BGW Database

New phone calls

13 Sep 2000
ASA/MA'00
Why Mobile Agents?

Agent gets name list and starts phone-call query without interaction with the HQ machine.

Only calls that pass the application-specific filter are sent to the HQ machine.
News Monitoring

1. Agent jumps to server machine with query.

2. Persistent query server retrieves and **clusters** existing documents.

3. Agent sends clusters back to analyst.
4. Analyst marks “relevant” clusters and sends them back to the agent and PQS.

5. PQS adds each new document to the clusters. If document ends up inside a “relevant” cluster, the PQS gives it to the agent.

6. Agent does any desired filtering.

7. Agent sends the documents that pass its filter back to the analyst.
Why Mobile Agents?

• Documents that do not pass the application-specific filter are never sent to the analyst’s machine.
• The agent can situate itself on any machine in the network.
Suspect Identification

- Agent moves to soldier’s machine
  - Pictures
  - Code to “browse” the pictures
- No interaction with HQ while browsing

```c
proc showPictures {window} {
    ...
}
```
Application: Technical Reports

GUI on home machine

1. Send agent

2. Send child agents and collect partial results

3. Return merged and filtered results

Dynamically selected proxy site
Reasons for Mobile Agents

- Reduce bandwidth usage
- Reduce total completion time
- Reduce latency
- Continue when disconnected
- Balance load
- Dynamically deploy components
Reason 1: Reduce Bandwidth Usage

Dataset

Dynamically selected proxy site

Merged and filtered data stream
Reason 1: Reduce Bandwidth Usage
TR Application: Bandwidth Usage

Number-of-documents scenario: Bandwidth usage

- Downloading
- Agents (20% relevant)
- Agents (0% relevant)

Tcl agent in the D’Agents system
TR Application: Bandwidth Usage

Number-of-queries scenario: Bandwidth usage

Number of bytes vs. Number of queries

- RPC
- Agents

Tcl agent in the D’Agents system
Reason 2: Reduce Total Time

Fact

• Sending an agent avoids remote interaction.

Goal

• Avoiding remote interaction leads to faster completion times.

Current Systems

• Do not meet the goal in all network environments
• Tradeoff: Local interaction vs. interpretive overhead
TR Application: Total Time

10 Mb/s Network

Why? Java 1.0 is slow. Transmitting documents over a 10 Mb/s link is nearly as fast as inspecting them with a Java agent.
TR Application: Total Time

2 Mb/s Network

- D’Agents (Java, 100%)
- D’Agents (Java, 20%)
- D’Agents (Java, 0%)
- Client/Server
TR Application: Tcl versus Java

10 Mb/s Network

Number of Documents

Time (milliseconds)

- D’Agents (Tcl, 0%)
- D’Agents (Java, 0%)
- Client/Server

10 Mb/s Network
Reason 3: Reduce Latency

1. Observe high average latency to clients

2. Move to better location

Sumatra chat server [RASS97]

2 to 4 times smaller latency in trial runs
Reason 4: Disconnected Operation

Agent continues its task even if the link to its home machine goes down (temporarily).

Dynamically selected proxy site
Reason 5: Load Balancing

Machine A

Agent moves to balance load

Machine A

Machine B

Machine B
Reason 6: Dynamic Deployment
Traditional Techniques: RPC

RPC

Invocation

Results

Queue invocations

Queue results

- No mobile code
- Client constrained to server’s interface
- All requests and results over client’s network link
- No latency or bandwidth reduction
- Blocked if link goes down
Stored Procedures and REV

- Perfect if accessing one server
- Difficulties if accessing more than one server
  - Procedures usually cannot communicate with each other
  - Procedures usually cannot send out their own procedures (e.g., no proxy)
Applets and Servlets

Applets

1. Request Web page

2. Receive page and “embedded” applet; execute applet

Servlets

1. Send servlet

2. Get results

• Same limitations as stored procedures and REV …
• But more attention to security
**TR without agents**

1. Lot of work.

   Pre-installed application-specific proxy

   Queued RPC

   Higher-level database interface

2. Much less work, but now try to extend it.

   “Queued” REV

   Client Code

   “Queued” REV

   Client Code
The Big Picture

• **Simple, unified framework** for implementing many distributed applications
  – Application might require several traditional techniques.
  – Application might require “impossible” pre-installation.
  – Different applications require different traditional techniques.
  – Extending other techniques produces mobile-agent systems.
  – Agents move or stay still as their needs dictate.

• **Current systems**
  – Performance advantages for some applications
  – Performance penalties for others
Next-Generation Systems

- Remote communication just as fast as RPC
  No penalty for stationary agent
- Just-in-time compilation and software fault isolation
  Execution at near-native speeds
- Code caching
  Low migration overhead

Thus …

- Mobile agents will do no worse than traditional implementations, and will often do better.
- Service providers will see a load only a little bit higher than if they had provided the high-level operations themselves.
Should I Even Consider Mobile Agents?

• If pre-installing code is easy, no need for mobile agents.

• Pre-installing must be difficult …
  – Unpredictable client needs?
  – Unpredictable service needs?
  – Unpredictable network environment?

• … and efficiency must be important.
  – Bandwidth or latency
  – Total time
  – Load
  – Disconnections
If So, Apply a Rule of Thumb

- Relative to mobile agents, client/server is network bound
  
  Use mobile agents if your application does a little processing on a large dataset

- Relative to client/server, agents are CPU bound
  
  Do not use mobile agents if your application does a lot of processing on a small dataset
## Guidelines (Current Systems)

<table>
<thead>
<tr>
<th>Primary Goal</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load balancing</td>
<td>Do not use mobile agents</td>
</tr>
<tr>
<td>Latency reduction</td>
<td>Network time dominates</td>
</tr>
<tr>
<td></td>
<td>Use mobile agents</td>
</tr>
<tr>
<td></td>
<td>CPU time dominates</td>
</tr>
<tr>
<td></td>
<td>Do not use mobile agents</td>
</tr>
<tr>
<td>Bandwidth conservation</td>
<td>Use mobile agents (but have agents decide if and when to move)</td>
</tr>
<tr>
<td>Disconnection handling</td>
<td>Use mobile agents</td>
</tr>
</tbody>
</table>
## Guidelines (Current Systems)

<table>
<thead>
<tr>
<th>Primary Goal</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total-time reduction</strong></td>
<td>More application specific, but if ...</td>
</tr>
<tr>
<td></td>
<td>Network time dominates</td>
</tr>
<tr>
<td></td>
<td><strong>Use mobile agents</strong></td>
</tr>
<tr>
<td></td>
<td>(but have agents decide if and when to move)</td>
</tr>
<tr>
<td></td>
<td><strong>CPU time dominates</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Do not use mobile agents</strong></td>
</tr>
</tbody>
</table>
Roadmap

• Definitions and Myths
• Why Mobility?

• **Representative Systems**
  – Java-Based Systems
  – Other Single-Language Systems
  – Multiple-Language Systems

• D’Agents
• Security
• The future
Representative Systems

Java-based systems
- Aglets, Voyager, Jumping Beans

Other single-language systems
- Messengers, Obliq, Telescript

Multiple-language systems
- Ara, Tacoma, D’Agents
Same General Architecture

1. Agent decides to move

2. A contacts B and sends serialized code and state

3. B authenticates and restarts agent

• Two general kinds of server
  – Single process
    Every agent is a thread.
  – Multiple process
    Every agent is a process.

• Hybrid servers as well
  – D’Agents

13 Sep 2000

ASA/MA'00
## Differences and Similarities

<table>
<thead>
<tr>
<th>Java</th>
<th>Other languages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single language</strong></td>
<td><strong>Multiple languages</strong></td>
</tr>
<tr>
<td><strong>Weak mobility</strong></td>
<td><strong>Strong mobility</strong></td>
</tr>
<tr>
<td><strong>Imperative</strong></td>
<td><strong>Functional or declarative</strong></td>
</tr>
<tr>
<td><strong>Interpreted</strong></td>
<td><strong>Just-in-time (JIT) compilation</strong></td>
</tr>
<tr>
<td><strong>Stand-alone</strong></td>
<td><strong>Tight Web integration</strong></td>
</tr>
<tr>
<td><strong>Unique protocols</strong></td>
<td><strong>FIPA/OMG standards</strong></td>
</tr>
</tbody>
</table>
Why Java?

Portability

Compiled into bytecodes for a stack-based virtual machine

Efficiency

Just-in-time (JIT) compilation

Security
Aglets

IBM http://www.trl.ibm.co.jp/aglets/

- Java
- Weak mobility
- Event-driven programming model
  (dispatch, onDispatching, onArrival, …)
- Persistent store
- “Proxies” for location transparency
- Machine protection
Jumping Beans

- Java
- Weak mobility
- **Central server** for tracking, managing and authenticating agents (but also failure point and bottleneck)
- Persistent store
- Machine protection

Voyager

ObjectSpace
http://www.objectspace.com/products/

• Java

• **Built on top of CORBA**

• **Weak mobility**

• Persistent store

• **Federated directory service and group communication (multicast)**

• **Machine protection**
Why not Java?

- Limited per-thread resource accounting (or enforcement)
- No strong mobility
- Support for multiple languages?
  - Inefficient on top of Java VM
Messengers

University of Geneva
http://cuiwww.unige.ch/tios/msgm/home.html

• MØ (similar to Postscript)

• Weak mobility

• Intended for low-level services, e.g., in the Messenger Operating System (MOS)

• Bulletin boards for data exchange and service listings

• Limited security, but work on fine-grained resource control in progress
Obliq

DEC Research (Compaq)
http://www.luca.demon.co.uk/Obliq/Obliq.html

- Obliq (interpreted, lexically scoped, object-oriented)
- Weak mobility
- Full proxy references
- Visual Obliq
- Limited security (access checks in Visual Obliq)
Telescript

General Magic
http://www.genmagic.com/

• Telescript (OO language similar to Java and C++)

• Strong mobility

• Telescript is compiled into bytecodes for a RISC virtual machine

• Persistent store

• Machine protection

• No longer available
Why not a single language?

- Many applications for mobile agents
- No one language is good for all of them.
Ara

http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/

- C/C++, Tcl and Java
- Strong mobility
- C/C++ compiled into bytecodes for RISC virtual machine
- Server plus all agents inside one Unix process
- Machine protection
Tacoma

University of Tromsø / Cornell University
http://www.tacoma.cs.uit.no/

• C, Tcl/Tk, Scheme, Python, Perl (public release), several more internally

• Weak mobility

• **Single, simple abstraction: meet**
  – Easy to add a new language
  – Less opportunity for optimization

• Machine protection
D’Agents

http://agent.cs.dartmouth.edu/

- Tcl and Java (public), Scheme (internal)
- Strong mobility
- Multi-threaded server, but agents in separate processes
- Machine protection
Recommendations

• Java only
  – Full support
    Voyager or Jumping Beans
  – Overall Performance
    Voyager
  – Ease of agent programming
    Aglets
  – Ease of administration
    Jumping Beans

• Scripting or multiple languages
  – Ara, Tacoma or D’Agents
  – All-around
    D’Agents
  – Ease of adding a new language
    Tacoma
  – Agent Execution Speed
    Ara (C/C++)
Roadmap

• Definitions and Myths
• Why mobility?
• Representative systems
• D’Agents
  – Overview
  – Example: Tcl agent
  – Example: Java agent
• Security
• The future
Multiple languages

- Tcl
- Java
- Scheme

Performance

- Halfway there
- Communication and migration overhead not as low as possible

Support services

- Directory service
- Tracker
- Debugger (Tcl)
- ...
D’Agents: Mobility Model

Strong mobility / No proxy references

1. Capture agent state
2. Sign/encrypt state
3. Send state to B

Machine A

Agent

Server

... searched = searched + 1;
jump B;
meet with search-engine;
...

Machine B

6. Resume agent
5. Restore state
4. Authenticate

... searched = searched + 1;
jump B;
meet with search-engine;
...
D’Agents: Architecture

Machine A

 agents

 Java VM

 Tcl interp.

 Scheme interp.

 Server

 Transport (TCP/IP)

 VM / Interpreter

 VM

 Security

 State Capture

 Server stubs
D’Agents: Hot Interpreters

- Pool of Tcl interpreters
- Pool of Scheme interpreters

Pipes from VM / interpreters to server

Server
How Fast Are Hot Interpreters?

Migration across DummyNet

Time (milliseconds)

Payload size (bytes)
Tcl/Tk

- Interpreted scripting language
- “Everything is a string.”
- Ideal for “glue” applications
  - Call high-level operation 1, modest processing to decide next operation, call high-level operation 2, …
- Tk provides GUI capabilities.

```tcl
proc processFile {filename} {
    # open the file and read in contents
    set fd [open $filename r]
    set contents [read $fd]
    close $fd

    # split the file contents into a list and process each line
    set splitContents [split $contents \n]
    foreach line $splitContents {
        ...
    }

    # MAIN SCRIPT - call our only procedure
    set code [catch {
        processFile agent.data
    } errorMessage]
    if {$code} {
        puts $errorMessage
    }
}
```

http://www.scriptics.com/
Aside: Capturing Control State

Flag Tcl command

Procedure WHILE_EXPRESSION
if (expr)
  flag = WHILE_BODY
  push body onto stack
else
  flag = NEXT_COMMAND

Procedure WHILE_BODY
if (error)
  flag = NEXT_COMMAND
else
  flag = WHILE_EXPRESSION

Evaluate and pop body

Procedure WHILE_BODY
if (expr)
  flag = WHILE_BODY
  push body onto stack
else
  flag = NEXT_COMMAND

Procedure WHILE_BODY
if (error)
  flag = NEXT_COMMAND
else
  flag = WHILE_EXPRESSION
Agent Tcl Primitives (I)

agent_begin
   Register with the local agent server
agent_name <name>
   Obtain a name in the namespace
agent_end
   Unregister and exit

agent_send
   Send a message to an agent
agent_receive
   Receive a message
Agent Tcl Primitives (II)

**agent_jump <machine>**
Migrate to a new machine and continue from the point of the jump

**agent_submit <machine>**
-procs <procedure-list>
-vars <variable-list>
-script <script>
Create a new (child) agent

1. agent_submit B
   -procs search
   -vars query
   -script
   \{search $query\}

2. search $query
Agent Tcl Primitives (III)

agent_fork <machine>
Clone an agent (*fork* a child agent)

Machine A

Machine B

agent_fork B

Exact copy that continues from fork point
Other Features

Meetings

Second communication mechanism (direct connection between agents)

Select

Wait for a message, meeting request, data arriving over a meeting, etc.

Status reports

Which agents are running on machine A?
Who owns agent X?

Notifications

Message from server on agent birth

Event-driven programming

Event handlers for all types of incoming communication
Addressing an Agent

**Symbolic name** (chosen by agent)

**Numeric id** (chosen by system, unique per machine)

location-independent addressing through the yellow pages
A Skeleton Agent

Child Agent

```tcl
proc child {machines} {
    global agent
    # migrate through machines
    set results {}
    foreach machine $machines {
        agent_jump $machine
        # do task, update results
    }
    # send back results and end
    agent_send \
        $agent(root) 0 $results
    agent_end
}
```

Parent Agent

```tcl
agent_begin

    # submit child
    set machines {A B ...}
    agent_submit \
        $agent(local-server) \ 
        -procs child \ 
        -vars machines \ 
        -script {child $machines}

    # get results
    agent_receive \
        code results -blocking
    puts $results
    agent_end

```

... set results {}

    # migrate through machines

foreach machine $machines {
    agent_jump $machine
    append results \
    [exec who << {}]
}

    # send back results

agent_send \n    $agent(root) 0 $results

...
TR Application

1. Register
2. Ask for query
3. Submit proxy agent
4. Submit search agents
5. Query database
6. Return results
7. Collect results
8. Send merged results
9. Display results
10. End

“Hardcoded” proxy and document locations

bald

tioga

muir

13 Sep 2000

ASA/MA'00 78
TR: Talking to the User

```
agent_begin # register with the agent system

puts -nonewline "Enter a query:"
gets stdin query

set proxy bald; set collections {muir tioga}
set me $agent(local)

agent_submit $proxy \ # submit the proxy agent
    -proclist {proxyAgent searchAgent} \
    -varlist {me query collections} \
    -script {proxyAgent $me $query $collections}

# receive the query results
agent_receive code results -blocking

puts "Query results:"
puts $results

agent_end       # done
```

“Front-end” Agent

Note: Position of some comments is invalid Tcl syntax.
TR: Sitting in the Middle

proc proxyAgent {parent query collections} {
    global agent
    set numCollections [llength $collections]
    set me $agent(local)

    foreach machine $collections {  # submit search
        agent_submit $machine \      # agents
            -proclist searchAgent -varlist {me query} \  
            -script {searchAgent $me $query}
    }

    set results {}
    for {set i 0} {i < $numCollections} {incr i} {
        # collect results
        agent_receive code partialResults -blocking
        # merge into results so far
    }

    agent_send $parent 0 $results # send back results
    agent_end # done
}

Proxy Agent
(submitted by the front-end agent)

Note: Position of some comments is invalid Tcl syntax.

13 Sep 2000
TR: Searching the Collections

proc searchAgent {parent query} {

    global agent

    # send the query to the database interface agent
    # and receive the results

    agent_send "$agent(local-server) TR" \
        0 "keyword-query $query"

    agent_receive code results -blocking

    # in a more complex version of the TR, use the
    # abstract of the most relevant document as a
    # new query, compare the two result lists, etc.

    # send back the results and stop

    agent_send $parent 0 $results
    agent_end
}

Search Agent
(submitted by the proxy agent)
TR: Step by Step (Redux)

1. agent_begin
2. gets stdin query
3. agent_submit proxyAgent
4. agent_submit searchAgent
5. agent_send/agent_receive
6. agent_send
7. agent_receive
8. agent_send
9. puts $results
10. agent_end
http://www.javasoft.com/

- Similar to C++
- Object-oriented
- Pointer-free
- Garbage-collected
- Multi-threaded
- Static
- Compiled into bytecodes
  - Stack-based VM
  - Much faster than Tcl

```java
public class Rectangle {
    public int width = 0;
    public int height = 0;

    // constructor
    public Rectangle(int w, int h) {
        this(w, h);
    }

    // compute the area of the rectangle
    public int area (void) {
        return width * height;
    }
}

class UseRectangles {
    public static void main(String[] args) {
        Rectangle r = new Rectangle (10, 20);
        int a = r.area();
        ...
    }
}
```
Aside: VM that Captures Thread State

Thread stack

Arguments

Local variables

int

obj

5

float

3.14

obj

Java objects on heap

“Mark-and-sweep” to find and serialize all reachable objects.

Added type field to stack.
Agent Java and Agent Tcl

1. Same infrastructure

2. One-to-one “command” mapping
Agent Java: Major Classes

class Agent {

    int handle;

    // constructor creates an internal C++ instance
    public native int createNativeAgent ();
    public Agent() { handle = createNativeAgent (); }

    // register and unregister (timeout parameter not shown)
    public native AgentId begin (String machine);
    public native void end ();
    // migrate to a new machine (timeout parameter not shown)
    public native void jump (String machine);
    public native AgentId submit
        (String machine, AgentEntryPoint newAgent);

    ...  
}

public abstract class AgentEntryPoint
{
    public abstract void run
        (Agent agent);
}

13 Sep 2000  ASA/MA'00  86
Agent Java: Minor Classes

public class AgentId {
    private String machine; // symbolic name for the machine
    private String name;   // symbolic name for the agent
    private int id;        // numeric id of the agent
    // access methods
    ...
}

public class Message {
    private int code;
    private String message;
    // access methods
    ...
}

public class RecMessage extends Message {
    private AgentId senderId; // sender
    private SecurityVector securityInfo; // security vector
    // access methods
    ...
}
The “who” Agent: Parent Agent

...// register with the agent system
Agent a = new Agent();
AgentId id = a.begin (10); // timeout after 10 seconds

  // submit the child agent
Vector machines = new Vector();
machines.addElement (new String (“A”));
machines.addElement (new String (“B”));
...
ChildAgent childAgent = new ChildAgent (machines);
AgentId childId = a.submit (“localhost”, childAgent, 10);

  // wait for and display the result and then end
RecMessage result = a.receive (10);
System.out.println (result.getMessage());
a.end (10);
...
class ChildAgent extends AgentEntryPoint {

    private Vector m_machines;   // machine list
    public ChildAgent (Vector machines)
        { m_machines = machines; }
    public String executeWhoCommand (...);

    public void run (Agent a) {
        String results = "";
        // migrate through all the machines in the list
        for (int i = 0; i < m_machines.size(); ++i) {
            String machine = (String) m_machines.elementAt (i);
            a.jump (machine, 10);
            results = results + executeWhoCommand();
        }
        // send back result and exit
        Message message = new Message (0, results);
        a.send (a.getRootId(), message, 10); a.end (10);
    }
}
public static String executeWhoCommand() {
    // NOTE: enclosing try-catch, check for non-zero
    // exit code, etc., are not shown

    Runtime runtime = Runtime.getRuntime();
    Process process = runtime.exec("/usr/bin/who");
    int exitCode = process.waitFor();

    InputStream whoOutputStream = process.getInputStream();
    DataInputStream whoOutput =
        new DataInputStream (whoOutputStream);

    String users = ""; boolean done = false;
    while (!done) {
        String line = whoOutput.readLine();
        if (line == null) { done = true } else {users = …}
    }
    return (users);
}
TR Application

1. Register

2. Submit proxy agent

3. Query yellow pages for locations

4. Send query and receive results

5. Send merged results

6. Display results

7. End

Machine 1

Machine n

“Hardcoded” proxy and directory locations

mayday

bald

"Hardcoded" proxy and directory locations
String query = "..."; // keyword query
String proxy = "bald"; // hard-coded proxy machine
String yp = "mayday"; // hard-coded yellow-pages machine

// register with the agent system
Agent a = new Agent();
AgentId id = a.begin (10); // timeout after 10 seconds

// submit the child agent
ProxyAgent proxyAgent = new ProxyAgent (id, query, proxy, yp);
AgentId childId = a.submit (proxy, proxyAgent, 10);

// wait for and display the results and then end
RecMessage result = a.receive (10);
System.out.println (result.getMessage());
a.end (10);
TR: Asking the Yellow Pages

```java
class ProxyAgent extends AgentEntryPoint {

    private AgentId m_parent;   // id of parent ("front-end")
    private String m_query;     // keyword query
    private String m_yp;        // yellow-pages machine

    public ProxyAgent
        (AgentId parent, String query, String yp)
    {
        // set member variables
    }

    public void run (Agent a) {
        // ask the yellow pages to locate TR collections;
        // note that method extractTrIds is not shown
        AgentId ypId = new AgentId (m_yp, "yellow pages");
        Message ypMessage = new Message (0, "locate tr");
        a.send (ypId, ypMessage, 10);
        RecMessage ypResponse = a.receive(10);
        Vector trIds = extractTrIds(ypResponse.getMessage());
    }
}
```
// send a query message to all the TR collections
for (int i = 0; i < trIds.size(); ++i) {
    AgentId = (AgentId) trIds.elementAt(i);
    Message trMessage =
        new Message (0, "keyword-query " + m_query);
    a.send (trId, trMessage, 10);
}

// collect the responses
String results = "";
for (int i = 0; i < trIds.size(); ++i) {
    RecMessage trResponse = a.receive (10);
    results = results + ...;  // merge into results
}

// send back results and exit
Message message = new Message (0, results);
a.send (m_parent, message, 10); a.end (10);
1. Agent a =
   new Agent();
   a.begin(...);

2. a.submit (proxy, proxyAgent, ...);

3. a.send (ypId, ...);
   a.receive(...);

4. a.send (trId, ...);
   a.receive(...);

5. a.send (m_parent, ...);

6. println(...);

7. a.end(...);
D’Agents: Base Performance

10 Mb/s Network

D’Agents (Migration)
D’Agents (Messages)
TCP/IP connection

Payload or Message Size (bytes)
Time (milliseconds)

0 5000 10000 15000 20000 25000 30000
0 20 40 60 80 100 120 140 160 180 200

13 Sep 2000
ASA/MA'00
What Lowers Performance?

1. All messages through server (plus TCP/IP)

2. Interpreter initialization (plus TCP/IP)
Roadmap

• Definitions and Myths
• Why Mobility?
• Representative Systems
• D’Agents

• Security
  – D’Agents Security
  – Protecting a Group of Machines
  – Protecting an Agent

• The Future
Three Security Concerns
D’Agents: Security

(Hard limits, but no scheduling)

(Other research groups)

(Ongoing research: market-based resource control)
D’Agents: Authentication

Public
Private
Public
Private
Home Network
Signed with
Agent becomes anonymous.
Signed with
Claims
Network B
Network A

Agent has ’s permissions.

13 Sep 2000
ASA/MA'00
101
D’Agents: Protecting the Machine

1. Authenticate
2. Accept or reject
3. Resume execution
4. open tutorial.ppt r
5. Access request (read) and security vector {owner, untrusted machines?}
6. Yes / no / quantity
7. If yes, open
Same Managers for All Languages

- Separation of
  - Enforcement
  - Policy
- All at application level (with normal Tcl and Java support)

Java agent (digitally signed)

Server

Java VM

Java security manager

Same filesystem manager (and same access policies)

Files

Filesystem Manager
Secure TR Application

1. Register
2. Ask for query
3. Submit search agents
4. Make queries
5. Collect results
6. Merge results
7. Display results
8. End
Secure TR Application

# AGENTKEY = “Robert S. Gray <BTV 11/30/71>”
# AGENTPASS = ...
# turn on digital signatures and register as an agent

security signatures on
agent_begin

# submit the search agents, receive the results, etc.
# me = ...; query = ...; collections = ...

foreach machine $collections {
    agent_submit $machine -proclist searchAgent \
    -varlist {me query} -script {searchAgent $me $query}
}

set results {}
for {set i 0} {i < $numCollections} {incr i} {
    ...
}
...
Secure TR Application

```
proc searchAgent {parent query} {

    global agent

    # send the query to the database interface agent
    # and receive the results

    agent_send "\$agent(local-server) TR" \ 0 "keyword-query \$query"
    agent_receive code results -blocking

    # in a more complex version of the TR, use the
    # abstract of the most relevant document as a
    # new query, compare the two result lists, etc.

    # send back the results and stop

    agent_send \$parent 0 \$results
    agent_end
}
```
A stationary agent that provides an interface to Cornell’s Smart system

Secure “Smart” Agent

```java
// register
Agent agent = new Agent();
AgentId id = a.begin (10);
a.name ("TR", 10);

// handle queries
while (1) {
    RecMessage request = a.receive (-1);
    SecurityVector secVec =
      request.getSecurityInfo();
    String owner = secVec.getOwnerKeyname();
    if (!secVec.isOwnerAuth() ||
      (!onAccessList(owner))) {
// reject request
    } else {
// handle request, possibly with
// another thread
    }
}
```
D’Agents: What’s Missing?

• Schedule resource access

• Maintain authorization across multiple hops

• Protect a group of machines
  – Machines that are not under single administrative control
  – Market-based resource control

• Protect an agent
  – Malicious machines
  – Side benefit: Helps maintain authorization across multiple hops
  – Many techniques
Market-Based Resource Control

Goals
- Economic incentive to accept agents
- Resource scheduling
- No “runaway” agents

Agent
- Gets resource share proportional to ticket share
- Optimizes utility function (parameters include risk tolerance, resource needs, and initial cash)
Threats to an Agent

• Modification
  – Mount an attack against a later machine
  – Perform work on behalf of the attacker
  – Reach an incorrect conclusion

• Theft
  – Electronic Cash
  – Algorithms
  – Sensitive Data
Protecting an Agent

• No complete technical solution
• Some partial technical solutions
  – Do not prevent modification or theft
  – Detect modification or theft as soon as possible, or
  – Make stolen information useless
Partitioning

“Control” agent on trusted machines

“Work” agent on untrusted machines
Partitioning II

Main agent on trusted machines

“Scout” agent(s) on untrusted machines

Trusted Machine

Untrusted Machine
Replication and Voting

$N$ agents access $N$ equivalent copies of the service

Vote to decide the result
Components

- Identification
  - Signed with owner’s key

- Code
  - Unprotected (except in transit)

- Parameters (constants)
  - Encrypted with key of home machine or trusted proxy

- State

- Data from Service 1
  - Can prevent (detect) component removal
Self Verification

- Identification
- State Verification Code
- Other code
- Parameters (constants)
- State

Signed with owner’s key

Each machine runs the agent’s own state verification code when the agent arrives.
Black-Box Messup Algorithms

- Make human analysis of the code more difficult
  - Thus, make targeted modification more difficult
- Does not prevent programmatic analysis
- However ...

set a 5
set b 10
set c [expr a * b]

set $m(17) \ [expr m(0) + m(17) + m(73)]
set $m(19) \ [expr m(10) + m(11) + m(74)]
set $m(103) \ [expr m(17) * m(19) + m(27)]
Encrypted Algorithms

```
set a 5
set b 10
set c \ [expr a * b]
```

<table>
<thead>
<tr>
<th>Encrypted but executable code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001101011100</td>
</tr>
<tr>
<td>1110011101010</td>
</tr>
<tr>
<td>1101010101010</td>
</tr>
<tr>
<td>1110101010101</td>
</tr>
</tbody>
</table>

Remote machine

Answer = 52

Encrypted results

```
1001101011100
1110011101010
1101010101010
1110101010101
```

13 Sep 2000 ASA/MA'00
Protecting an Agent

• How do we combine these techniques to protect the agent?
  – Efficient
  – Easy to program
  – Minimal restrictions on migration patterns
  – Rapid detection of malicious modifications

• If the agent is protected well, we can maintain access rights across hops.
  – How do we measure confidence?
  – How do we relax access restrictions?
Aside: Secure but Fast!

• Specially designed bytecode set (e.g., Java) or proof-carrying code (PCC)
  – Many constraints do not need to be checked at runtime.

• Just-in-time compilation (JIT) and software fault isolation (SFI)
  – Near native-code speed
  – Colusa Software achieved performance within 20-30% of native compiled code.
Security Summary

• Different applications require different levels of security.
  – Many applications only require authorization.
  – Many applications only require machine protection.

• Possible to deploy an MA system today that adequately protects individual machines in many application environments.

  – Remaining issues
    • Denial of service attacks
    • Loss of authorization
Security Summary

• Protecting machine groups
  • Trivial if single administrative control
  • Market-based resource control

• Protecting an agent
  – Many techniques
  – Key issue

  How can the techniques be efficiently combined?

• Social and legal pressures will always play a role.

• Many of the hardest problems are not specific to mobile code.
Roadmap

- Definitions and Myths
- Why Mobility?
- Representative Systems
- D’Agents
- Security
- The Future
Current Trends Lead to Mobile Agents

Information overload
- Increased need for personalization

Diversified population
- "Customization"
- Too many unique, dispersed clients to handle
- Mobile code to server or proxy
- Mobile Agents
- Avoid "star" itinerary

Bandwidth gap
- Avoid large transfers

Mobile users and devices
- Disconnected Operation
- Mobile code to client
- High latency
- Multiple sites to visit
Migrating to Migrating Code

- Applets
- Proxies provided by existing ISP’s
- Proxies that accept servlets
- Services that accept servlets
- Mobile Agents

Intranet

Internet
Current Systems

- Sufficient access restrictions for some application environments

- Sufficient performance for many applications, but not many other applications
  - Interpretive overhead
  - Migration overhead
  - Slow remote (and local) communication

- Wide variability in debugging, administrative and other tools
Next-Generation Systems

• Sufficient access restrictions and resource scheduling for most application environments
• Decent debugging and administrative tools
• Remote communication just as fast as RPC
  No communication penalty for a stationary agent
• Just-in-time compilation with code caching
  • Small migration overhead
  • Small execution overhead
  Not much more CPU load than if the service had provided the operation itself
Key Issues

• Performance and scalability
• Security
  • Protecting the agent
  • Relaxing access restrictions on a multiple-hop agent
  • Market-based resource control
• Fault tolerance
• Monitoring and administrative control
• Application suite
Conclusion: Cons?

- Security is too big a concern?
- Overhead for moving code is too high?
- Networks will be so fast that performance will not be an issue?

- Yes … in some environments, for some applications.
- No … many environments, many applications.
- Mobility is just another technique, good in some cases, bad in others.
Conclusion: Pros

• Unifying framework for many distributed applications
• Efficiency without pre-installation
• Treats data and code symmetrically
• Supports disconnected networks in a way that other technologies cannot
• Cleaner programming model
Conclusion

• Examine your application and application environment

• Ask if agent mobility can help
  – Unpredictable client needs?
  – Poor network?
  – Disconnected operation?

• If so, select your mobile-agent system
  – Many systems (including a few commercial systems)
  – Different performance and security characteristics
Read More About It

D’Agents

http://agent.cs.dartmouth.edu/

This Tutorial (with note pages)

http://agent.cs.dartmouth.edu/tutorials/

Other Mobile-Agent Systems

http://www.cetus-links.org/oo_mobile_agents.html
Source Code

D’Agents (server and Tcl module)

http://agent.cs.dartmouth.edu/

D’Agents (Java module)

Available by request (due to Sun licensing restrictions).
Contact robert.s.gray@dartmouth.edu.

D’Agents (Scheme module)

Coming soon.