Code Migration in Distributed Systems

Distributed Software Systems

Motivation for Code Migration

- Load Sharing in Distributed Systems
  - Long-running processes can be migrated to idle processors
- Client-server systems
  - Code for data entry shipped to client system
  - If large quantities of data need to be processed, it is better to ship the data processing component to the client
  - Dynamically configurable client software
    - More flexibility, Easier maintenance and upgrades of client software
- Enterprise and "Desktop Grids", e.g. SETI@home
  - Computationally-intensive tasks shipped to idle PCs around the network
Dynamically Configurable Client Software

The principle of dynamically configuring a client to communicate to a server. The client first fetches the necessary software, and then invokes the server.

Models for Code Migration

- A process has three segments
  - Code segment
  - Execution segment - private data, stack, PC, registers
  - Resource segment - references to external resources such as files, printers, devices, etc
- Weak vs strong mobility
  - weak mobility: only code segment + initialization data migrated, e.g. Java applets
  - strong mobility: code segment + execution segment
- Sender-initiated vs receiver-initiated migration
Alternatives for code migration.

Migration and Local Resources

- Process-to-resource bindings make code migration difficult
- Three types of process to resource bindings
  - Binding by identifier - when a process refers to a resource by its identifier, e.g. URL, IP address, local communication endpoint (socket)
  - Binding by value - weaker form of binding when only the value of a resource is needed, e.g. when a program relies on standard language libraries
  - Binding by type - weakest form of binding when a process indicates the type of a resource, e.g., a printer
Migration and Local Resources (cont’d)

- When migrating code, we may need to change the references to resources but cannot change the kind of process-to-resource binding.
- How a resource reference is changed depends on the resource-to-machine bindings
  - Unattached resources can be easily moved, e.g. data files associated only with the program being moved
  - Fastened resources can be moved at a high cost, e.g. a database
  - Fixed resources cannot be moved, e.g., local devices, local communication endpoint

---

Migration and Local Resources cont’d

<table>
<thead>
<tr>
<th>Process-to-resource binding</th>
<th>Unattached</th>
<th>Fastened</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>By identifier</td>
<td>MV (or GR)</td>
<td>GR (or MV)</td>
<td>GR</td>
</tr>
<tr>
<td>By value</td>
<td>CP (or MV, GR)</td>
<td>GR (or CP)</td>
<td>GR</td>
</tr>
<tr>
<td>By type</td>
<td>RB (or GR, CP)</td>
<td>RB (or GR, CP)</td>
<td>RB (or GR)</td>
</tr>
</tbody>
</table>

GR: Establish a global system-wide reference
MV: move the resource
CP: copy the value of the resource
RB: Rebind process to locally available resource

Actions to be taken with respect to the references to local resources when migrating code to another machine.
Migration in heterogeneous systems

- Weak mobility: no runtime information needs to be transferred, so it suffices to generate separate code segments for different target platforms
- Strong mobility: how to transfer the execution segment
  - One approach: runtime systems maintains a language-independent copy of the program stack
  - More common approach: Use a virtual machine

Migration in Heterogeneous Systems

The principle of maintaining a migration stack to support migration of an execution segment in a heterogeneous environment
Overview of Code Migration in D’Agents (1)

```tcl
proc factorial n {
    if ($n <= 1) { return 1; } # fac(1) = 1
    expr $n * [ factorial [expr $n - 1] ] # fac(n) = n * fac(n - 1)
}
set number … # tells which factorial to compute
set machine … # identify the target machine
agent_submit $machine –procs factorial –vars number –script {factorial $number }
agent_receive … # receive the results (left unspecified for simplicity)
```

A simple example of a Tel agent in D’Agents submitting a script to a remote machine

Example: D’Agents

- D’Agents: research middleware platform that supports various forms of code migration
- Agent is a program that can migrate between heterogeneous platforms
  - written in Tcl, Scheme, or Java
- Agent mobility
  - Sender-initiated weak mobility: agent_submit command
  - Strong mobility by process migration: agent_jump command
  - Strong mobility by process cloning: agent_clone
    - agent_clone similar to agent_jump except that invoking process continues execution at the source machine
Overview of Code Migration in D'Agents (2)

all_users $machines

proc all_users machines {
    set list "" # Create an initially empty list
    foreach m $machines { # Consider all hosts in the set of given machines
        agent_jump $m # Jump to each host
        set users [exec who] # Execute the who command
        append list $users # Append the results to the list
    }
    return $list # Return the complete list when done
}

set machines … # Initialize the set of machines to jump to
set this_machine # Set to the host that starts the agent

# Create a migrating agent by submitting the script to this machine, from where
# it will jump to all the others in $machines.
agent_submit $this_machine –procs all_users
    -vars machines
    -script { all_users $machines }
agent_receive … #receive the results (left unspecified for simplicity)

An example of a Tel agent in D'Agents migrating to different machines where it executes the UNIX who command.

Implementation Issues (1)

<table>
<thead>
<tr>
<th></th>
<th>Agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Tcl/Tk interpreter</td>
</tr>
<tr>
<td>4</td>
<td>Scheme interpreter</td>
</tr>
<tr>
<td>3</td>
<td>Java interpreter</td>
</tr>
<tr>
<td>2</td>
<td>Common agent RTS</td>
</tr>
<tr>
<td>1</td>
<td>Server</td>
</tr>
<tr>
<td></td>
<td>TCP/IP</td>
</tr>
</tbody>
</table>

The architecture of the D'Agents system.

- The Server is responsible for agent management, authentication, and management of communication between agents.
- The RTS layer supports the core functionality of the system, i.e., creation of agent, migration, interagent communication, etc.
## Implementation Issues (2)

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global interpreter variables</td>
<td>Variables needed by the interpreter of an agent</td>
</tr>
<tr>
<td>Global system variables</td>
<td>Return codes, error codes, error strings, etc.</td>
</tr>
<tr>
<td>Global program variables</td>
<td>User-defined global variables in a program</td>
</tr>
<tr>
<td>Procedure definitions</td>
<td>Definitions of scripts to be executed by an agent</td>
</tr>
<tr>
<td>Stack of commands</td>
<td>Stack of commands currently being executed</td>
</tr>
<tr>
<td>Stack of call frames</td>
<td>Stack of activation records, one for each running command</td>
</tr>
</tbody>
</table>

The parts comprising the state of an agent in D’Agents.

---

## Software Agents

- A software agent is an autonomous process capable of reacting to, and initiating changes in its environment, possibly in collaboration with users and other agents
  - **Collaborative agents**
    - part of a multi-agent system in which agents try to achieve some common goal through collaboration
  - **Mobile agents**
    - capable of moving between systems
  - **Interface agents**
    - agents that assist an end user in the use of one or more applications
    - have learning capabilities
  - **Information agents**
    - manage information from many different sources
**Software Agents in Distributed Systems**

<table>
<thead>
<tr>
<th>Property</th>
<th>Common to all agents?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomous</td>
<td>Yes</td>
<td>Can act on its own</td>
</tr>
<tr>
<td>Reactive</td>
<td>Yes</td>
<td>Responds timely to changes in its environment</td>
</tr>
<tr>
<td>Proactive</td>
<td>Yes</td>
<td>Initiates actions that affects its environment</td>
</tr>
<tr>
<td>Communicative</td>
<td>Yes</td>
<td>Can exchange information with users and other agents</td>
</tr>
<tr>
<td>Continuous</td>
<td>No</td>
<td>Has a relatively long lifespan</td>
</tr>
<tr>
<td>Mobile</td>
<td>No</td>
<td>Can migrate from one site to another</td>
</tr>
<tr>
<td>Adaptive</td>
<td>No</td>
<td>Capable of learning</td>
</tr>
</tbody>
</table>

Some important properties by which different types of agents can be distinguished.

**Agent Technology**

The general model of an agent platform
Agent Communication Languages (1)

<table>
<thead>
<tr>
<th>Message purpose</th>
<th>Description</th>
<th>Message Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORM</td>
<td>Inform that a given proposition is true</td>
<td>Proposition</td>
</tr>
<tr>
<td>QUERY-IF</td>
<td>Query whether a given proposition is true</td>
<td>Proposition</td>
</tr>
<tr>
<td>QUERY-REF</td>
<td>Query for a given object</td>
<td>Expression</td>
</tr>
<tr>
<td>CFP</td>
<td>Ask for a proposal</td>
<td>Proposal specifics</td>
</tr>
<tr>
<td>PROPOSE</td>
<td>Provide a proposal</td>
<td>Proposal</td>
</tr>
<tr>
<td>ACCEPT-PROPOSAL</td>
<td>Tell that a given proposal is accepted</td>
<td>Proposal ID</td>
</tr>
<tr>
<td>REJECT-PROPOSAL</td>
<td>Tell that a given proposal is rejected</td>
<td>Proposal ID</td>
</tr>
<tr>
<td>REQUEST</td>
<td>Request that an action be performed</td>
<td>Action specification</td>
</tr>
<tr>
<td>SUBSCRIBE</td>
<td>Subscribe to an information source</td>
<td>Reference to source</td>
</tr>
</tbody>
</table>

Examples of different message types in the FIPA ACL giving the purpose of a message, along with the description of the actual message content.

Agent Communication Languages (2)

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>INFORM</td>
</tr>
<tr>
<td>Sender</td>
<td>max@<a href="http://fanclub-beatrix.royalty-spotters.nl:7239">http://fanclub-beatrix.royalty-spotters.nl:7239</a></td>
</tr>
<tr>
<td>Receiver</td>
<td>elke@iiop://royalty-watcher.uk:5623</td>
</tr>
<tr>
<td>Language</td>
<td>Prolog</td>
</tr>
<tr>
<td>Ontology</td>
<td>genealogy</td>
</tr>
<tr>
<td>Content</td>
<td>female(beatrix),parent(beatrix, juliana, bernhard)</td>
</tr>
</tbody>
</table>

A simple example of a FIPA ACL message sent between two agents using Prolog to express genealogy information.
Secure Mobile Code

- Mobile code introduces security threats
- Mobile agents need to be protected from malicious hosts
  - hosts may try to steal or modify information carried by the agent
- Hosts need to be protected against malicious agents
  - Viruses and worms are instances of (stealthy) malicious agents!!

Protecting an agent

- Malicious hosts may
  - steal information carried by an agent
  - modify an agent to change its behavior
  - destroy an agent
- Fully protecting an agent against all kinds of attacks is impossible
- Alternative: organize agents in such a way that modifications can be detected
  - Example: Ajanta system
Ajanta

Three mechanisms that allow an agent’s owner to detect that the agent has been tampered with

Read-only state
- Collection of data items signed by owner
  - message digest encrypted with private key
  - host can verify the received read-only state using the public key of owner

Append-only logs
- data collected by an agent can only be appended to the log
- Initially checksum associated with empty log, $C_{\text{init}} = K(N)$, where $N$ is a nonce and $K$ is public key of owner

Ajanta cont’d

Append-only logs (cont’d)
- When a server $S$ appends $X$ to the log, it calculates a new checksum $C_{\text{new}} = K(C_{\text{old}}, \text{sig}(S, X), S)$, where $C_{\text{old}}$ is the previously used checksum
- When the agent comes back to the owner, the owner can start reading the log at the end successively decrypting the checksum, until the initial checksum is reached

Selective revealing of state
- an array of data items, each intended for a designated server
- each entry is encrypted with the designated server’s public key
- the entire array is signed by the agent’s owner
Protecting the target

- More critical problem than protecting an agent
- Approaches
  - create a sandbox, e.g. Java
    - a technique by which a downloaded program is executed in such a way that each of its instructions can be fully controlled
  - create a playground
    - a separate designated machine exclusively reserved for downloaded code
    - resources local to other machines are physically disconnected from the playground
    - users on other machines can access the playground using traditional means, e.g. RPC

Sandbox vs Playground

(a) A sandbox
   - Trusted code
   - Untrusted code
   - Local network
   - Sandbox

(b) A playground
   - Only trusted code
   - Untrusted code
   - Local network
   - Playground
Java sandbox organization

- Only trusted class loaders are used
- Byte code verifier checks whether downloaded class contains illegal instructions or instructions that could corrupt the stack or memory
- A security manager performs various checks at run-time to ensure that the downloaded object does not make any unauthorized access to client resources
  - e.g. checks I/O operations for validity, disallows access to local files, etc.
Adding flexibility

- Playgrounds are more flexible than sandboxes
- Next step: downloaded programs are authenticated, and subsequently a specific security policy is enforced based on the where the program came from
  - authentication achieved through digital signatures
  - enforcing a security policy more challenging

Enforcing security policies

- Wallach et al proposed three mechanisms for enforcing a security policy for Java programs
  - Use object references as capabilities
  - Stack introspection
  - Name space management
    - to access local resources, programs need to include the appropriate files that contain the classes implementing those resources
    - Interpreter enforces different policies for different downloaded programs by resolving the same name to different classes
- Language-independent solutions are more difficult to implement and require support from the OS
Enforcing security policies

The principle of using Java object references as capabilities.

Enforcing security policies

The principle of stack introspection.