8. Mobile Agents, Mobile Code

• Motivation for mobile computation

• *Pervasive* Networking and Computing
  ➢ Connectivity and computing is cheap and available

• *Ubiquitous* Networking and Computing
  ➢ Connectivity and computing power is available everywhere (independently from location)

• *Mobile* Computing
  ➢ Network nodes can be placed everywhere – wireless communication

• Easy-to-use Technologies for naive users
  ➢ World Wide Web
    ▪ Electronic commerce
    ▪ Internet phone

• Growing need for *scalability*
• Diffusion of network services and applications to large segments of the society
  ➢ *Customizability*
  ➢ *Flexibility*
  ➢ *Extensibility*
• Well-established models and technologies
  ➢ Essentially RPC-based
    ▪ CORBA etc.
• *Code mobility*
  ➢ The capability to dynamically change the bindings between code fragments and the location of execution
8.1. Code Mobility

8.1.1. Technologies

- A true distributed system is *location transparent*
- In a Mobile Computing System (MCS), applications are *location-aware*
  - Computational Environment (CE)
    - Location of the execution
  - Executions Units (EU)
    - Sequential flow of computation (e.g. a thread)
    - Code segment, data space, execution state
  - Resources
    - Sharable among EUs (e.g. files)
Code and Execution State Mobility

- **Strong mobility**
  - Code, data space and execution state of an EU can migrate
  - Migration
    - The EU is suspended, moved to the new CE and resumed
  - Remote Cloning
    - Creates a copy of the running EU at the remote CE
    - The original EU is not detached
  - Proactive migration or cloning
    - Time and destination are determined by the migrating EU
  - Reactive migration or cloning
    - Time and destination are determined by another EU
  - Strong mobility is entirely transparent to the user
  - The transmitted code just resumes on the new CE
  - Expensive

- **Weak mobility**
  - Code and data space can migrate, but not the execution state
  - The migrated code will be restarted at a given procedure
    - It starts in a similar way as an interrupt-handler
  - Fetch or ship
    - Either an EU fetches the code dynamically or ships code to execute to another CE
  - Stand-alone code or code fragment
    - Stand-alone code instantiates a new EU
    - A code fragment is linked to an already running code
  - Synchronous or asynchronous mobility
    - Synchronous: requesting EU suspends execution
    - Asynchronous: immediate or deferred
Data Space Management

- Upon migration, bindings to resources must be rearranged
- Resource = (I, V, T)
  - I: unique identifier
  - V: value
  - T: type
  - Transferable vs. Not transferable
    - Principally not transferable is e.g. a printer
    - Transferable resources maybe
      - Free – can be freely migrate to another CE
      - Fixed – associated to a CE (e.g. a huge file)
- Resource binding to an EU may be
  - By identifier
    - The EU requires to be bound to a uniquely identified resource that cannot be substituted
    - E.g. a connection to a certain database
  - By value
    - The value (content) of the resource must not change due to the migration
    - The identity of the resource is not relevant
    - E.g. the value of a variable
  - By type
    - Bound resource must be compliant with a given type
    - Identity and value of the resource are not relevant
    - E.g. a system resource such as some network connection, or a block of memory
- Different bindings to the same resource is meaningful. E.g.
  - An EU makes a binding by identifier to its display
  - It makes a binding by type to the same resource
  - After roaming it has actually two displays:
    - The first one permanently associated to the original CE
    - The second one at the actual CE, wherever it is
8.1.2. Design Paradigms (or Methodology)

- Architectural concepts
  - Components
    - Code components – represents algorithms
    - Resource components – data or devices
    - Computational components – processors
  - Interactions
    - Events involving two or more components
  - Sites
    - Hosts computational components
    - They represent the locations
  - Interactions among components on the same site are cheaper than those located on different sites
  - We assume that a computation can be carried out, if the following are all on the same site
    - The know-how describing the computation
    - The resources used during the computation
    - The corresponding computational component
  - The computational component $A$ at site $S_a$ needs a service. There is a site $S_b$ that is involved in the service
  - Louise and Christine interact and cooperate to make a cake (result of the computation). They need
    - A recipe (the know-how about the service)
    - The ingredients (movable resources)
    - An oven (a not moveable resource)
    - A person making the cake (computational component)
  - The main design paradigms are
    - Client/Server
    - Remote Evaluation
    - Code on Demand
    - Mobile Agents
Client-Server

- Louise would like to have a cake
- She does not know the recipe and she has neither ingredients nor an oven
- She calls Christine and asks her to make a cake for her
- Christine makes the cake and delivers it to Louise

- The client component $A$ at site $S_a$ sends a request to site $S_b$
- The server component $B$ at site $S_b$ performs the service using its own resources and returns the result to $A$

Remote Evaluation (REV)

- Louise wants to prepare the cake
- She knows the recipe but she has neither ingredients nor an oven
- She calls Christine, tells her the recipe and asks her to make such a cake for her
- Christine makes the cake and delivers it to Louise

- The client component $A$ at site $S_a$ sends the know-how (a procedure) to site $S_b$
- The server component $B$ at site $S_b$ performs this procedure using its own resources and returns the result to $A$. 
Code on Demand (CoD)

- Louise wants to prepare the cake
- She has both ingredients and oven, but she lacks the recipe
- She calls Christine, and asks her to tell the recipe
- Louise makes the cake

- The client component $A$ at site $S_a$ has all the necessary resources but does not know the procedure to process them
- It asks for the know-how at site $S_b$
- The server component $B$ at site $S_b$ delivers the know-how
- The client component $A$ at site $S_a$ executes this procedure

Mobile Agent (MA)

- Louise wants to prepare the cake
- She has both ingredients and the recipe, but no oven
- She prepares the cake
- She goes to Christine and completes the cake in her oven
- Either she eats it there or she comes back to home with the cake

- The client component $A$ at site $S_a$ has the necessary know-how
- Some of the required resources are at site $S_b$
- $A$ migrates to site $S_b$, carrying the know-how and maybe some intermediate results (some data) with
- $A$ completes the computation at site $S_b$, using its resources
- As opposed to REV and CoD, a whole computational component (including its state and some resources) is moved, not just code
8.1.3. Applications

• There is no best paradigm nor a best technology
• Paradigm and technology are principally orthogonal
• In practice, they must conform to each other and application
• So, we have to clearly distinguish
  ➢ The application or application domain (e.g. a system to control a remote telescope)
  ➢ The design paradigm (e.g. Remote Evaluation)
  ➢ The technology used (e.g. Java Aglets)
• Mobile code applications are still rare
• Performance
  ➢ Mobile code is usually executed by an interpreter
• Security
  ➢ If code can move easily, malicious code can move easily as well (‘ideal’ for virus propagation)
  ➢ Authentication of the sender of mobile code
    ▪ A server may want to authenticate the client
    ▪ The client may also want to authenticate the server
  ➢ Determination of the exact rights of the migrated code at the destination
  ➢ Sandbox
    ▪ Potentially dangerous calls are restricted by security control components
  ➢ Organizational approach
    ▪ Allow mobile agents only to trustworthy institutions
    ▪ Maybe to institutions with “good reputation”
  ➢ Manipulation detection
    ▪ Does not protect against read attacks
  ➢ Blackbox
    ▪ Obfuscate code and invalid it before the attacker has time to crack it (Mole system)
Key benefits of mobile code

• Service customization
  ➢ Traditional Client/Server systems provide a fixed set of services – that is regularly upgraded by new versions
  ➢ An alternative way is a simple server with dynamically extensible functionality provided by the client

• Deployment and maintenance
  ➢ The new version of the software is installed on a server
  ➢ Clients may load and dynamically link code fragments on demand – lazy propagation

• Autonomous Components
  ➢ Communication channels have often low-bandwidth and low-reliability (e.g. wireless)
  ➢ Constant connection is therefore not available (no C/S)
  ➢ The user sends a single request to a stationary agent
  ➢ The stationary agent makes the job, while e.g. the mobile equipment might be switched off

• Fault tolerance
  ➢ In the client/server model the global state is distributed
  ➢ Mobile agents just take their local state with themselves

• Protocol Encapsulation
  ➢ Traditionally communication protocols must be installed at both peer entities
  ➢ With mobile code one basic protocol suffices
  ➢ Additionally, more sophisticated protocols can be downloaded with the help of the basic protocol

• Software Engineering
  ➢ Mobile agents are well-suited to make rapid prototypes of a distributed system
  ➢ Instead of a full installation, only a basic environment is installed
  ➢ Prototyping code is distributed as mobile agents
Application Domains

- Distributed Information Retrieval
  - Classical example for the MA paradigm
    - A mobile agent visits different hosts
    - It applies dynamic routing based on actual information
    - It performs search and filtering at the source of information, such decreasing network traffic

- Active Documents
  - Traditionally passive documents (as e-mail or Web pages) may be turned to be active by taking some code with
  - E.g. an e-mail takes the presentation software with it, thus allowing a presentation close to the original
  - E.g. an application that uses graphic forms to express queries to a remote database
    - The user requests the active document (CoD) and uses it as an interface (e.g. with WWW+Java applets)

- Advanced Telecommunication Services
  - Videoconference, video on demand, telemeeting etc.
  - They need dynamic reconfiguration and customization
  - E.g. the setup, signaling and presentation services could be dispatched to the users by a broker (REV)

- Remote Device Control and Configuration
  - E.g. network management
  - Configuration and monitoring code can be shipped (REV)

- Workflow Management
  - A mobile agent leads the workflow document through all stages of its processing in a distributed environment

- Active Networks (CoD)
  - E.g. routers can be dynamically “reprogrammed”

- Electronic Commerce
  - User transactions are carried out by a mobile agent
8.2. Mobile Agents

Mobile agents are defined as active (autonomous) objects (or clusters of objects) that have behavior, state and location. They can but need not to be intelligent (in the sense of AI).

8.2.1. Types of Agent Communication

- Mobile Agent ↔ Service Agent Interaction
  - Client/server like
  - Ideally an RPC-like mechanism is provided
- Mobile Agent ↔ Mobile Agent Interaction
  - Peer-to-peer, rather than client/server like
  - Often not just request/response, general message passing
- Anonymous Agent Group Interaction
  - The sender often does not know the receiver
    - E.g. a group of agent is working on a problem
    - The sender knows the group but not the individual agent
  - Group communication can be used to this purpose
  - Event channels can be used as well
  - The interaction between senders and receivers goes through event channels – this is additional level of indirection
8.2.2. Some available mobile agent systems

<table>
<thead>
<tr>
<th>Languages</th>
<th>Mobility</th>
<th>Source</th>
<th>Avail.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tacoma</td>
<td>Tcl, C, ...</td>
<td>REV</td>
<td>Uni Cornell</td>
</tr>
<tr>
<td>J. Servlets</td>
<td>Java</td>
<td>REV(push)</td>
<td>Sun</td>
</tr>
<tr>
<td>J. Applets</td>
<td>Java</td>
<td>COD (pull)</td>
<td>Sun</td>
</tr>
<tr>
<td>Active X</td>
<td>C, C++, ...?</td>
<td>COD</td>
<td>Microsoft</td>
</tr>
<tr>
<td>Aglets</td>
<td>Java</td>
<td>Weak migr</td>
<td>IBM</td>
</tr>
<tr>
<td>Mole</td>
<td>Java</td>
<td>Weak migr</td>
<td>Uni Stuttgart</td>
</tr>
<tr>
<td>Odyssey</td>
<td>Java</td>
<td>Weak migr</td>
<td>General Magic</td>
</tr>
<tr>
<td>Voyager</td>
<td>Java</td>
<td>Weak migr</td>
<td>ObjectSpac</td>
</tr>
<tr>
<td>AgentTcl</td>
<td>Tcl</td>
<td>Strong migr</td>
<td>Dartmouth</td>
</tr>
<tr>
<td>Ara</td>
<td>Tcl, C, Java</td>
<td>Strong migr</td>
<td>Uni Kaiserslaut</td>
</tr>
<tr>
<td>Telescript</td>
<td>Telescript</td>
<td>Strong migr</td>
<td>General Magic</td>
</tr>
<tr>
<td>Emerald</td>
<td>Emerald</td>
<td>Strong migr</td>
<td>Uni …</td>
</tr>
</tbody>
</table>
Example TicTacToe

- The simple TicTacToe class is located statically at the client sites and it may emit `askToPlay` calls, on user demand
- The sever site accepts and answers these calls
  - The 1. player must wait until a second is coming
  - If the 2. request arrives, the sever creates an agent and passes it to the 1. player (Player$_0$), via the `goTo` method
- The players now send the agent back and forth with the actual state of the game, via the `goToPlayer` method
- The agent starts running at the destination site at a method specified by the source site. The selection of the method depends on state of the game (e.g. `printRemis` for undecided)
- If the game is over (one player wins resp. remis) they shut down the voyager daemon

![Example TicTacToe Diagram]
public class TTTAgent implements ITTTAgent, Serializable {

public void goTo() {
    // Called by the server. It sends the agent to player_0
    try {
        // Agent's execution there starts at printFieldArrival
        Agent.of(this).moveTo(players[0], "printFieldArrival");
        . . .  }

public void printFieldArrival() { . . . playTurn(); }

public void playTurn() // Checks input and the effect of it
{ . . .
    if (game not ended yet) {
        goToPlayer("printFieldArrival");
        else if (remis) { isRemis() }
        else if (winner) { isWinning() }
    }

public void isWinning() // Print winning message
{ goToPlayer("printLooser");
    Voyager.shutdown();  }

public void printLooser() // Print losing message
{ Agent.of(this).setAutonomous(false);
    Voyager.shutdown();  }

public void goToPlayer(String callback) {
    // Agent is moved – incl. instance variables (state of the game)!
    // The actual value of callback depends on the state of the game.
    // For normal case, printFieldArrival, for remis printRemis ...
    try {
        Agent.of(this).moveTo(players[actual++ % 2], callback);
        . . .  }
} }
8.3. Partitioning an Application

- Non-distributed model of the application (UML)

- Distributed model of the application
8.4. Mobile Agents and the Future of the Internet
(D. Kotz and R. Gray)

8.4.1. Trends

- **Bandwidth**
  - Huge amount of bandwidth at “backbones”
    - 1.6 Tbps fiber in 2000 (Nortel)
  - Growing gap between high and low edge
    - 128Kbps – 1 Mbps for modem and wireless users

- **Mobile devices**
  - Rapidly growing market
    - Laptops, palmtops, electronic books, car telephones, pagers etc.
    - All these will have Internet access

- **Mobile users**
  - Pervasive services
    - E.g. web-based email services (hotmail)

- **Intranets**
  - Owned and managed by single organization
    - Little coordination is needed
    - Security is of less concern

- **Information overload**
  - Users are overwhelmed with available information
    - Search engines, filtering etc.

- **Customization**
  - For each user at both client and server site

- **Proxies**
  - Reduce information overload, customize service access
  - Buffering and caching
8.4.2. Mobile Agents in Coming

- Maybe no really new applications
- Maybe not much better performance, but
- Single framework for distributed, information-oriented applications
  - More useful applications with more useful features
- Amount of information and the diversity of users are growing
  - Need for personalized presentations and access methods
- Today such personalization is provided at the
  - Information source in a site-specific manner
  - Proxy Web site
  - Client software
- Consequences of the growing bandwidth-gap:
  - Client will try to avoid to transfer large amount of data
  - Functionality should migrate from client to proxy
  - Proxies will aggressively gather, pre-fetch, cache
- Mobile users will often disconnect and reconnect
  - Proxies
  - Mobile code to install client code dynamically
- Customization tools will be specified as mobile software
  - May run on the server or
  - On a dynamically selected proxy or
  - On the client site
- Multi-hop mobile code is needed
  - This needs mobile agents
8.4.3. Technical hurdles

- Performance and scalability
  - Mobile agents save network latency and bandwidth
  - They need generally more execution time
    - Interpretative execution
    - Security checks
    - Restoration of execution context
  - Promising techniques
    - Just-in-time compilation
    - Software fault isolation
    - Research in reducing migration overhead

- Portability and standardization
  - Portability among heterogeneous systems is generally supported (JVM)
  - Portability among mobile agent systems not
  - Standardization is needed
    - OMG MASIF
      - Addresses only cross-system communication and administration
      - Agent may not be able to migrate to the desired target only to the nearby “right” agent system
      - Standardization of execution environment and formats to encode state and code is needed

- Security
  - Protecting machines without limiting agent access rights more than necessary
    - Agents with limited life time (as long as “cash“ holds)
  - Protecting agents from malicious machines
    - More difficult
  - Protecting group of machines that are not under the same administrative control
8.4.4. Non-technical hurdles

- Acceptance
  - High initial costs for a new technology
  - Perceivable improvements must be achieved
  - Lack of a “killer” application
    - Most applications can be implemented easily and efficiently also without mobile agents
    - Gain is modest for single applications in isolation
    - Convincing arguments needed a set of applications
  - Support is necessary to
    - Write a survey of mobile-agent applications
    - Open some Internet sites

- Evolutionary path
  - Internet services will not be willing to “jump” to the new technology
  - Switch must be made incremental
  - A probable evolution could be
    - Applets are accepted and used
    - Proxy sites accepting mobile code – such proxies will probably provided by Internet service providers (ISPs)
    - Mobile code sent from the client (or the proxy) to the server (e.g. servlets) – needs additional security
    - Full-fledged mobile agent systems
  - Migration of the MA technology from intranets to the Internet
    - First it will appear in safe intranet environments on high-latency networks

- Revenue and image
  - Through MA the number of human visits at a Web page may decrease – ads will not be seen!
  - A “poor” agent may lead to bad image of the service