Software Agents
What is a Software Agent

- No unique definition
- Many different types of applications ⇒ no consensus.
- Several key concepts have emerged
- Two popular definitions:
  - An agent is a component of software and/or hardware which is capable of acting exactly in order to accomplish tasks on behalf of its user (Nwana, 1996)
  - An agent is a computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives (Jennings, Sycara and Wooldridge, 1998)
What is a Software Agent (contd.)

- Agents typically possess several (or all) of the following characteristics
  - Autonomous
  - Adaptive/learning
  - Mobile
  - Persistent
  - Goal oriented
  - Communicative
  - Collaborative, cooperative
  - Self-starting / Long-lived
  - Flexible
  - Active/proactive

- Usually small in size. Do not constitute a complete application.
Why Agents?

• A natural abstraction

• Can be adapted for complex software projects

• Agents integrate diverse disciplines of CS

• Increased productivity and efficiency

• Powerful metaphor for designing and implementing complex software systems, since multiple cooperating agents can be used to solve very formidable problems.
Agents Classification

Autonomous Agents

- Biological Agents
- Robotic Agents
- Computational Agents

Software Agents

- Task-specific Agents
- Entertainment Agents
- Viruses

Artificial Life Agents
Client-Server vs. Mobile Agents

Traditional

Mobile Agent-Based
What problems does this solve?

• Agent technology also attempts to solve the age-old problem of getting a computer to do the real thinking for us.

• Mobile agents solve the nagging client/server network bandwidth problem.

• Agent architectures also solve the problems created by intermittent or unreliable network connections.
Mobile Agent: Basic Principles

• Trend coined by **Java**
• *shipping code to data source* vs. *remote communication*
• Two MA approaches:
  • *Remote execution*
    • Agent transfer (code and data) before Task execution
    • Agent stays at target system until task is performed
  • *Migration*
    • Agent transfer during Task(s) execution, i.e. Agent performs specific task(s) at different nodes (Agent decides when and where to go)
• In addition to the *code and data*, the *execution state also* has to be transferred
Basic Definitions
Basic Definitions

- An **agent system** can create, interpret, execute, transfer and terminate agents.

- A **host** can contain several agent systems; each is uniquely identified by its name and address.
Some Definitions

• agents have **locations** (names of their current places)

• agents have **names** (the agent’s authority and identity; a unique value)
There are two kinds of agents:

- **stationary agent** is permanently attached to a place; often resource managers, server programs or search engines

- **mobile agent** can move from one place to another
MAL (Definitions)

- An **executing unit**, EU is a single process (or thread) image of execution.
- EU is always considered in the context of a place, which contains components; either EUs, or resources, such as files.
- A **MAL** is a language designed for distributed systems, which supports EUs migrating between various places.
MAL (Definitions)

- An EU consists of
  - a **static** code segment
  - a program state:
    - **data space** containing accessible resources
    - **execution state** containing system information such as program counter and return address.
- The EU has a **distributed state** if its data space exists in more than one place.
MAL (Definitions)

- **Strong mobility** means that the entire code and execution state of EUs can be moved.
- **Weak mobility** means that the agent is restarted from the same initial state after it is restarted at the new place.
Benefits of Mobile Agents

- Reduce **human load**
- Handle **information overload**
- Provide **automated help** to untrained users
- Provide a new, more powerful methodology to develop **complex software systems**
- Asynchronous / autonomous **task execution**
- Reduction of network traffic and client processing power
- **Automation** of distributed **task processing**
- Decentralized / local task processing (control & management)
- **Flexibility**
Agent based Applications

- Business process agents
  - Data-driven workflow management
- Information management agents
  - Email filtering agents
  - Web browsing assistants
  - Notification agents
  - Resource discovery agents
- Agent Projects at SIU
  - MPI-Agent Based
  - Agent Based System Administration
  - Domain-Based Knowledge Discovery
Agent platforms
Building an Agent

Agent Architecture
Building an Agent

• What do we need to think about to build an agent?
  – Architecture
  – Security
Agent Architecture

• An agent must have its own unique identity
• An agent host must allow multiple agents to co-exist and execute simultaneously
• Agents must be able to determine what other agents are executing in the agent host
• Agents must be able to send and receive messages to/from other agents
Agent Host Architecture

- An agent host must allow agents to communicate with each other and the agent host
- An agent host must be able to negotiate the exchange of agents
- An agent host must be able to freeze an executing agent and transfer it to another host
- An agent host must be able to thaw an agent transferred from another and allow it to resume execution
- The agent host must prevent agents from directly interfering with each other
Agent architecture: The objects

- From the requirements, we can determine what classes will be needed in the system.
  - an Agent class
  - an AgentHost class
  - an AgentInterface class
  - an AgentIdentity class
Classes in detail...

- The AgentHost class defines the agent host. An instance of this class keeps track of every agent executing in the system. It works with other hosts in order to transfer agents.
Classes in detail...

- The Agent class defines the agent. An instance of this class exists for each agent executing on a given agent host.
Classes in detail...

• The AgentInterface class defines the agent interface. An instance of this class envelopes an agent and provides access to it via a well-defined interface. It is also the primary conduit for communication between agents. An AgentInterface instance is the only handle an agent gets to the other agents executing on a given host.
Classes in detail...

- The AgentIdentity class defines agent identity. An instance of this class uniquely identifies an agent. Agents use this information to identify the agents with whom they are interested in collaborating.
Mobile Agent System Checklist

• *Agent Mobility* - the ability to transport agents between computers
• *Agent Naming* - the ability to assign globally unique names to agents to distinguish one agent from another
• *Agent Authentication* - the ability to authenticate the identity of the owner (authority) of an agent
• *Agent Permissions* - the ability to assign permissions to agents that restrict access to data and unintended consumption of computer resources. Selected agents may have the ability to grant permissions to other agents or re-negotiate their own set of permissions
• *Agent Collaboration* - the ability to request and respond to requests for establishing a meeting with another agent. Agents should also have the ability to begin and end meetings with other agents and enforce rules for the meetings
Mobile Agent System Checklist

- **Agent Creation** - the ability for agents to create other agents locally and remotely. New agents may have the authority of the existing agent and either the same permissions or a subset of them.
- **Agent Life Cycle** - the ability to control the life-span of agents by age and resource consumption.
- **Agent Termination** - the ability to terminate agents gracefully, thereby allowing them to notify other agents they are collaborating with.
- **Agent Staging** - the ability to write to disk agents that must wait for long periods of time for events to occur.
- **Agent Persistence** - the ability to checkpoint agents to disk so that they survive crashes on their host computers.
Mobile Agent System Checklist

• *Agent Interaction* - the ability for related agents to interact. The means of interaction might depend upon whether the agents occupy the same or different computers.

• *Agent Management* - the ability to manage a collection of agents in the system

• *Agent Tracking* - the ability to track and locate agents that have migrated to other computers

• *Agent Debugging* - the ability to monitor and log agent activities and exceptions
JADE

- **JADE (Java Agent DEvelopment Framework)**
- Simplifies implementation of multi-agent system (MAS)
- Programming framework in Java
- High level programming for
  - controlling & configuring MASs
  - communication
  - mobility
JADE overview

- **Jade** includes
  - runtime environment
    - must be active on a host for agent execution
  - **Java library**
    - for programming agents
  - **Graphical tools**
    - administering & monitoring agent activity
Behavior class

- Behaviors store what an agent has to or can do
- For every task
  - Make a new behavior class
  - `extend jade.core.behaviours.Behaviour`
  - `implement the action(), done() methods`
  - `addBehaviour()` to add to the agent
    - anytime, anywhere
setup()

Agent has been killed (doDelete() method called)?

- Initializations
- Addition of initial behaviours

Get the next behaviour from the pool of active behaviours

b.action()

b.done()?

- Agent “life” (execution of behaviours)

Remove currentBehaviour from the pool of active behaviours

takeDown()

- Clean-up operations

Highlighted in red the methods that programmers have to implement
A selfish behavior

```java
public class OverbearingBehaviour extends Behaviour {
    public void action() {
        while (true) {
            // do something
        }
    }

    public boolean done() {
        return true;
    }
}
```
One-shot & cyclic behaviors

- **One-shot**
  - complete immediately
  - `action()` run only once

```java
public class MyOneShotBehaviour extends OneShotBehaviour {
    public void action() {
        // perform operation X
    }
}
```

- **Cyclic**
  - continuous operation

```java
public class MyCyclicBehaviour extends CyclicBehaviour {
    public void action() {
        // perform operation Y
    }
}
```
One-shot & cyclic behaviors

• One-shot
  • complete immediately
  • action() run only once

  ```java
  public class MyOneShotBehaviour extends OneShotBehaviour {
      public void action() {
          // perform operation X
      }
  }
  ```

• Cyclic
  • continuous operation

  ```java
  public class MyCyclicBehaviour extends CyclicBehaviour {
      public void action() {
          // perform operation Y
      }
  }
  ```
Generic behaviors

```java
generic behaviors

public class MyThreeStepBehaviour extends Behaviour {
    private int step = 0;
    public void action() {
        switch (step) {
            case 0:
                // perform operation X
                step++; 
                break;
            case 1:
                // perform operation Y
                step++; 
                break;
            case 2:
                // perform operation Z
                step++; 
                break;
        }
    }

    public boolean done() {
        return step == 3;
    }
}
```