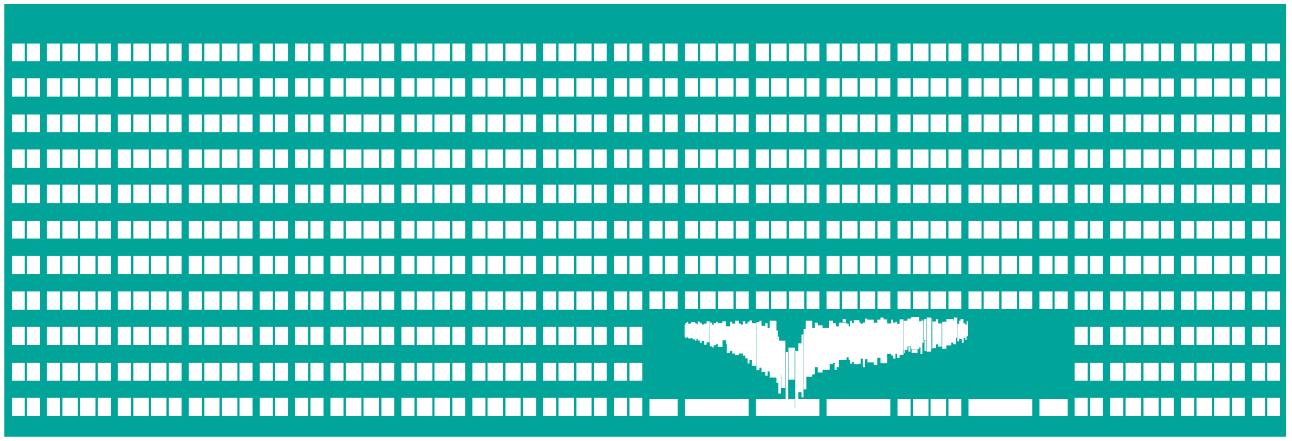
VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Mobile code
Multi-agent systems



MS (Mobile Computing)
Lecture 6

Mobile Code

Code mobility

- Capability to dynamically change the bindings between code fragments and the location where they are executed.
- Main aspects of mobile code system (MCS):
  - mobility mechanisms – enable application to move executable code between computational environments
  - design paradigms – benefit from MM, describe concepts of distributed applications which rely on mobile code
  - application domains – define where mobile code is useful and improves distributed application performance

Mobility mechanisms (1)

1. Mobility of code and execution state – executing unit (EU) = code + current computation state (CS). Move to different *computation environment* (CE):
   - Strong mobility – whole execution unit:
     - *Migration* – EU is suspended & after the transfer of code to the new CE, resumed again
     - *Remote cloning* – does not detach the original EU from its CE,
   - Weak mobility – code only.
     - Direction: fetch – to the EU × ship – from the EU
     - Nature of the code: stand-alone × only a fragment.
     - Synchronization: synchronous × asynchronous
     - Execution: immediate × deferred

Mobility mechanisms (2)

2. Data space management – executing code references resources managed by local computational environment. Moved code must hold valid references in new CE → when move occurs, modify bindings to the resources.
   - Resources split into:
     - transferable – free × fixed – huge file is t., but undesirable
     - non transferable – e.g. printer
   - Resource binding to EU:
     - by identifier – uniquely identified resource, not replaceable, must be moved or a *network reference* must be used (complicated, data space is distributed over the network)
     - by value – copy (most convenient) × move (esp. # EU >1)
     - by type – re-bind to a resource of same type on targ. EU.

Design paradigms

- Distribution of know-how (code) and resources (data) over sites, i.e. representation of location.
- Also deals with computational component which triggers the execution of the know-how.
- Paradigms:
  - remote evaluation – local computational component sends the know-how to a comp. component which resides on the same site as the resource and receives result.
  - code on demand (COD) – local component requests know-how from remote site to process the resource locally
  - mobile agent – similar to remote evaluation, sends the entire (existing and running) computational component to the remote site.

Application domains (1)

- *Distributed information retrieval* – gathers information matching some specified criteria from a set of information sources dispersed in the network by moving the code to huge databases.
- *Active documents* – traditionally passive data, like E-mail or Web pages, are enhanced to execute programs partly related with the document contents, enabling enhanced presentation and interaction. Typically COD is used (Java, JavaScript, …)
- *Advanced telecommunication services* – videoconferences, video on demand, or telemeetings require a specialized “middleware” providing mechanisms for dynamic reconfiguration and user customization. When dealing with mobile users, autonomous components can be used.

Application domains (2)

- *Remote device control & configuration* – code mobility can be used to design and implement monitoring components co-located with monitored devices and report events representing the evolution of the device state. In addition, the shipment of management components to remote sites can improve both performance and flexibility.
- *Workflow management & cooperation* – provide support for mobility of activities encapsulating their definition and state. E.g., a mobile component could encapsulate a text document undergoing several revisions, maintaining information about the document state, legal operations on its contents, and next scheduled step in the revision process based on the workflow.

Application domains (3)

- *Active networks* – managing the network through the traffic according to applications' needs. Two extremes:
  - A) *programmable switch* – COD approach providing dynamic extensibility of network devices through dynamic linking of code.
  - B) *capsule* approach – attach some code to every packet flowing in the network. It describes computation which must be performed on packet data, at each node.
- *Electronic commerce* – business transactions through the network among several independent (possibly competing) business entities involving negotiation possible need to access continuously evolving information. Need to customize the behavior of individual sides of negotiation based on negotiation protocol + to move application components close information sources.

Mobile code technologies (1)

- Scripting languages (old ones):
  - *Agent TCL* – TCL interpreter extended with support for strong mobility. EU (agent) is a Unix process running the language interpreter, sharing only resources provided by OS, considered as non-transferable. EUs can jump to another CE, fork a new EU at a remote CE, or submit code to a remote CE. Migration of a whole interpreter along with its code & ES, proactive remote cloning mechanism code shipping for standalone code (asynchronous & immediate).
  - *TACOMA* – TCL + weak mobility. EUs (agents) are Unix processes, dedicated run-time supporting agent check-in and check-out. Synchronous & asynchronous immediate execution Initialization data encapsulated in a “briefcase”, resources in CE contained in “cabinets”.

Mobile code technologies (2)

- *Safe TCL* – active e-mail, containing code in TCL: trusted & untrusted (highly limited, code of uncertain origin) interpreters.
- *M0* – stack-based interpreted language that implements the concept of *messengers* (EUs) transmitted between *platforms* (CEs) and executed upon reception. EU can submit code to another EU. Resources are transferable and fixed, M0 is a weak MCS providing shipping of standalone code & copy-based data space management.

- *JavaScript*, (*VBScript*, …) – code shipped with static content to provide dynamic modifications of data at host node. COD approach. May also include support for data space management (resource copying in AJAX)

Mobile code technologies (3)

- Standalone systems (old):
  - *ARA* – multilanguage MCS, strong mobility. EUs – agents – managed by language-independent system core + interpreters for supported languages = CE. place abstraction. Mobility: *proactive* migration; data space management: agents share only system resources.
  - *Obliq* – untyped, object-based, lexically scoped, interpreted language. EU = thread, CE = *execution engine*, remote execution, weak mobility, synchronous shipping of standalone code. Objects are transferable fixed resources. Local objects → network references

Mobile code technologies (4)

- *Telescript* – object-oriented language, large distributed applications. Security & strong mobility. Intermediate, portable language called Low Telescript between engines = CEs. Agents & places = EUs. Move go operation, proactive remote cloning. Places are stationary EUs that can contain other EUs. Data space management: ownership. DSM: by move.
- Stand-alone systems (newer):
  - *Java* – Java Virtual Machine (JVM) = CE, *class loader* mechanism, weak mobility using mechanisms for asynchronous fetching code fragments. Immediate & deferred execution. *Applets* embedded in web pages, *Enterprise Java Beans* (*EJB*). Many frameworks are based on Java and provide additional support for mobile code.
  - *Mobility-RPC* — Mobile agent, RE & COD, RPC, Java

Mobile code technologies (5)

- Java-based frameworks:
  - **Java Aglets** – *context* = abstraction of the CE, dispatch & retract primitives for code move, aglet is reexecuted from scratch after migration, but retains value of its object attributes which can provide initial state for its computation. Resource management: copy
  - Mole – Java API for weak mobility. Mole agents run as threads of Mole CE. A place provides access to the underlying OS through stationary service agents. Asynchronous, immediate code shipping. The code & data to be sent are determined based on island – a transitive closure over all the objects referenced by the main agent object. Islands cannot have object references to the outside, DSM by move.

Mobile code technologies (6)

- *Sumatra* – Java extension for implementation of resource-aware mobile programs with support for strong mobility of Java threads = Sumatra EUs. Proactive migration mechanisms, remote cloning & shipping of standalone code with synchronous, immediate execution. Threads or stand-alone code can be migrated separately from the objects they need. Dynamically created object aggregates that determine the unit of mobility & persistency - object groups. Objects belonging to a group must be explicitly checked in and out. Programmer has ability to modify dynamically the granularity of the unit of mobility. Data space management in an object-group is always by move; bindings to migrated objects owned by EUs in source CE are transformed into network references.

Multi-agent systems

Multi-agent system

- Multi-agent system (MAS) is composed of multiple interacting autonomous intelligent agents. It can be used to solve problems impossible or difficult to solve by individual agent or monolithic system (e.g. stock exchange, on-line trading, disaster management, social networks).
- Agents in a MAS have following characteristics:
  - Autonomy – agents are (at least partially) autonomous
  - Local views – no agent has a full global view of the system, or it might, but the system is too complex for an agent to make practical use of the knowledge.
  - Decentralization – no designated controlling agent.

Intelligent agent (1)

- Intelligent agent (IA) is an autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals.
- IA may also learn or use knowledge to achieve its goals.
- Classification:
  - Simple reflex agents – acts only on the basis of the current percept. Condition-action rule: **if** condition **then** action.
  - Model-based reflex agents – include current state stored inside the agent, “World View” model.
  - Goal-based agents – store information which situations are desirable.

Intelligent agent (2)

- Utility-based agents – defined measure of how desirable a particular state is obtained through the use of a utility function which maps a state to a measure of the utility of the state.
- Learning agents – learning allows agents to initially operate in unknown environments and to become more competent.
- Other types: Decision agents, Input agents (process & make sense of sensor inputs), Processing agents, Spatial agents (relate to the physical real-world), Believable agents, Physical/Embodied agents and Temporal agents

MAS application domains

- Embodied agents – physical interaction with environment: predator-prey, foraging, box pushing, robot soccer, cooperative navigation & target observation, herding
- Game-theory problems – coordination games, social dilemmas
- Real-world scenarios – traffic monitoring & control (cars, air), network management and routing, electricity distribution management, distributed medical care, supply chains, hierarchical MAS problems, modeling social interaction, scheduling of meetings

Robot soccer

- FIRA – Federation of International Robot-soccer Association
- Simulation or real-world based games
- The independent agent concept is not observed in many implementations of robot control
- Micro-robots (box with maximal dimension 7.5 cm, wheels) $\times$ humanoid robots
- Different playground and team sizes.

FIPA

Foundation for Intelligent Physical Agents (founded 1996, member of IEEE since 2005)

- FIPA specifications – a collection of standards which are intended to promote the interoperability of heterogeneous agents and the services that they can represent.
  - Architecture, content language, device ontology, QoS
  - Nomadic Application Support, Communicative Act Library, Agent Management
  - Interaction Protocols, Message transport
  - ACL

Remarks

- Multi-agent – modeled from 2+ agents, but 10-1000 and more is reasonable expectation
- Team heterogeneity – usually modeled in homogeneous environment, but heterogeneous agents are common
- Agent complexity – much higher internal complexity of agent in real-world applications
- Dynamic team assignment, dynamic scenarios – what happens?

- 100+ MAS development environments and counting

Examples of MAS with mobility

- Mobile-C – mobile agent platform, C/C++
- Tryllian Agent Development Kit (discontinued) – dynamic, smart software components that can work together with the outside world. ADK features:
  - Task oriented programming, secure mobile code
  - Communication between components – http(s) or jms
  - Integration with Java EE app. servers, web services
- For Wireless Sensor networks:
  - Agilla, actorNet
  - MAPS (Mobile Agent Platform for Sun SPOTs)
- Agent Factory – Standard & Micro Edition (AFSE/ME)
- Java Agent Development Framework (JADE) – container for agent, a set of containers forms a platform with *Main Container* and two special agents – AMS & DF.

JADE description

- Platform for running agents with:
  - Asynchronous agent programming model
  - Communication between agents either on the same or different platforms
  - Mobility, security, and other utilities
  - Agent framework hooked to JAVA GUI applications
- Features:
  - Support for mobile devices, ontologies
  - Graphical runtime environment
  - FIPA compliant with predefined interaction protocols
  - Agent Management System (AMS) and Directory Facilitator (DF) have a common subset of actions