CogSysI Lecture 10: Introduction to MAS

Intelligent Agents
WS 2006/2007

Part III: Multi-Agent Systems

Introduction to MAS
Five Trends in Computing

- **Ubiquity**
  Introduce processor power into many devices and places

- **Interconnection**
  Computer systems are networked into large distributed systems

- **Intelligence**
  Complexity of tasks which can be automated and delegated to a computer is growing steadily

- **Control**
  In more and more domains control is given over from human to computer; e.g. autopilot
Human-orientation
Programming paradigms: machine-oriented to more human-oriented abstractions; HCI: from setting switches to direct manipulation via GUI

New field in computer science: Multiagent Systems
The Idea of MAS

- An agent is a computer system that is capable of independent action on behalf of its owner or user.
- A multiagent system consists of a number of agents which interact, typically by exchanging messages via some computer network infrastructure.
- Different agents might represent users/owners with different goals/motivations.
- Therefore, to successfully interact, agents require the ability to
  - Cooperate
  - Coordinate
  - Negotiate

with each other (similar to interaction of people in everyday life)
Key Research Questions

- Micro/Agent Design: how to build agents capable of independent, autonomous action
- Macro/Society Design: how to build agents capable of interacting with other agents, esp. if they have different goals/interests?
- Standard AI: focus on intelligent individual
- MAS: Social abilities
  - Emergence of cooperation in a society of self-interested agents
  - Language to communicate beliefs and aspirations
  - Conflict recognition and resolution
  - Coordination of activities to reach common goals
Example Scenarios

- NASA Deep Space 1 mission (1998): space probe with an autonomous, agent-based control system which can make some decisions by itself (before: control decisions were completely done by a 300 person ground crew)

- Autonomous air-traffic control systems: recognition of failure of other control systems and cooperation to track and deal with attended flights (e.g. DVMT, Durfee; OASIS)

- Last minute holiday package via PDA, using a negotiating agent
MAS is Interdisciplinary Research

- Software Engineering: Agent paradigm (going beyond OO)
- Social Sciences: Using theories, gaining insights by simulation of artificial societies
- AI: use planning, reasoning, learning technologies; study intelligent behavior in dynamic, interactive environments
- Game theory: use theories and techniques for negotiation
Definition: Agent

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.

Compare to

- Control systems: e.g. thermostat
- Software demons: e.g. xbiff
Environments

- Accessible vs. inaccessible
  obtaining complete, accurate, up-to-date information about the environments state

- Deterministic vs. non-deterministic
  each action has a single, guaranteed effect, no uncertainty about the result of an action; note: highly complex deterministic environments must be handled as non-deterministic

- Static vs. dynamic
  environment remains unchanged except by performance of the agent

- Discrete vs. continuous
  fixed, finite number of actions and percepts

→ Open Env: inaccessible, non-deterministic, dynamic, continuous
Reactive Systems

- Two sources of complexity of MAS: characteristics of environment and nature of interaction between agent and environment.

- Reactive system: maintainance of interaction with environment, must be studied and described on a behavioral level (not a functional level, i.e. in classical terms of pre- and postconditions).

- Example: reactive planning systems.

- Local decisions have global consequences.

- Example: printer controller.
  Simple rule: first grant access to process $p_1$ and at some later time to process $p_2$ is unfair, because it might never grant access to $p_2$. 
Intelligent Agent

An intelligent agent is a computer system with the ability to perform actions independently, autonomously, and flexible (on behalf of a user or an owner).

Flexibility means: being
- reactive
- pro-active
- social
Demands and Examples

- Performing a useful activity on behalf of humans or organizations (cleaning roboter)
- Coexist/interact with humans (cleaning roboter)
- Be aware of social rules and norms (transportation robot)
- Coordinate activities (team of cleaning robots)
- Cooperate or compete (RoboCup)
- Entertain or educate people (games, tutor systems)
Acting Reactively

- If the environment of a program is static (known in advance), the program cannot fail (Compile-Time, Runtime)

- In the real world, changes occur, information is incomplete (dynamic system).

- A reactive system continuously interacts with its environment and reacts in time to changes

- Example: Java-Listener, BUT: here reactions do NOT take into account the current state of the environment, they are determined in advance

- Reactive systems can be modelled relative straight-forward: e.g. as stimulus-response rules
Acting Pro-actively

- means: generate goals autonomously, try to reach goals
- not only event-driven behavior but: act on one’s own initiative
Acting Socially

- Real world is a multi-agent environment
- When trying to reach goals, others must be taken into account
- Some goals can only be reached through cooperation
- In some situations exit conflicts and competition (e.g. internet auctions)
- Social skills of agents: ability to model goals of other agents when trying to reach one’s own (local) goals, ability to interact (i.e. cooperate and coordinate)
Further Features

- Mobility: ability to move in a computer net or in another environment
- Adaptivity/learning: Improving performance over time
- Rationality: do not act in a way which hinders to fulfill one’s goals
Example: Tileworld

- Dynamic environment: holes appear/disappear
- Agent must recognize changes and modify its behavior
Agents as Intentional Systems

- Endowing agents with “mental” states: beliefs, desires, wishes, hopes
- Folk psychology: attributing attitudes for predicting and explaining other people's behavior
- Intentional systems (Dennett):
  - First order: having beliefs, desires, etc.
  - Second order: having beliefs and desires about beliefs and desires of its own and others
- Compare to physical systems: for predicting that a stone will fall from my hand I do not attribute beliefs and desires but mass or weight
Abstract Architecture

- Environment: finite set of discrete states $E = \{e, e', \ldots\}$
  - Assumption: continuous env can be modelled by a discrete env to any degree of accuracy

- Repertoire of possible actions of an agent:
  $Ac = \{\alpha, \alpha', \ldots\}$

- Interaction of agent and environment: run $r$, as a sequence of interleaved environment states and actions
  - $R$: set of all possible finite sequences over $E$ and $Ac$
  - $R^{Ac}$: subset of $R$ ending with an action
  - $R^E$: subset of $R$ ending with an environment state
Abstract Architecture

- State transformer function: $\tau : R^Ac \rightarrow P(E)$
- Termination: $\tau(r) = \emptyset$
- Environment $Env = (E, e_0, \tau)$
- Agent: $Ag : R^E \rightarrow Ac$
- Set of runs of an agent in an environment $R(Ag, Env)$
- Behavioral equivalence: $R(Ag_1, Env) = R(Ag_2, Env)$
Purely Reactive Agents

- No reference to their history, next state is only dependent on the current state

\[ Ag : E \to Ac \]

- Example: thermostat
Perception

- $see : E \rightarrow Per$
- $action : Per^* \rightarrow Ac$
- Agent $Ag = (see, action)$
- Equivalence relation over environment states: $e \sim e'$ if $see(e) = see(e')$
- If $| \sim | = |E|$, the agent is omniscient
- If $| \sim | = 1$, the agent has no perceptual ability
Agents with State

- Internal states $I$
- $action : I \rightarrow Ac$
- $next : I \times Per \rightarrow I$

State-based agents as defined here are not more powerful than agents as defined above.

Identical expressive power: Every state-based agent can be transformed into a standard agent that is behaviorally equivalent.
Utility Functions

- Telling an agent *what* to do without telling it how to do it
- Indirectly via some performance measure
- Associate utility with states of environment, prefer actions leading to states with higher utilities
- Utility can be defined over states or over runs
  \[ u : E \to \mathbb{R} \text{ or } u : R \to \mathbb{R} \]
Task Environments

- \( \Psi : R \rightarrow \{0, 1\} \)
  is 1 (true) if a run satisfies some specification and 0 (false) otherwise

- Task environment: \( \langle Env, \Psi \rangle \)

- specifies the properties of the environment and the criteria by which an agent will be judged to have succeeded in its task

- Definition of success
  - pessimistic: \( \forall r \in R(Ag, Env) \) it has to hold \( \Psi(r) \)
  - optimistic \( \exists r \in R(Ag, Env) \) where it holds \( \Psi(r) \)

- Two kinds of tasks:
  - Achievement: relation to planning
  - Maintainance
Deductive Reasoning Agents

- The usual problems of knowledge engineering
  - Transduction problem: Translating world in adequate symbolic description
  - Representation problem: providing a representation such that agents can reason with it for the results to be in time and useful

- Agents as theorem provers
- Logic-based agents: “deliberate agents”
- Specialized languages, e.g. MetateM, based on temporal logic
Deliberate Agents

- \( D \) as a set of logical formulae, internal state of an agent is \( \delta \in D \)
- \( \rho \) as set of deduction rules
- \( \delta \rightarrow_{\rho} \varphi \): formula \( \varphi \) can be proved from database \( \delta \) using deduction rules \( \rho \)
- Goal: Deriving a formula \( Do(\alpha) \) either as best action or as action which is not explicitly forbidden

function action(\( \delta : D \))

for each \( \alpha \in Ac \) do
  if \( \delta \rightarrow_{\rho} Do(\alpha) \) then return \( \alpha \)

for each \( \alpha \in Ac \) do
  if \( \delta \not\rightarrow_{\rho} \neg Do(\alpha) \) then return \( \alpha \)

return null
Vacuum World

Predicates:
In(x,y)
Dirt(x,y)
Facing(d)
(d ∈ {north, east, south, west})

Navigation:
In(0,0) ∧ Facing(north) ∧ ¬Dirt(0,0) → Do(forward)

Cleaning:
In(x,y) ∧ Dirt(x,y) → Do(suck)
Concurrent MetateM

- Michael Fisher, 1994
- Language for direct execution of logical formulae (executable temporal logic)
- Near the “ideal” of agents as deductive theorem provers
- Concurrently executing agents, communication via asynchronous broadcast message passing
- Components of an agent
  - Interface: defines interaction with other agents with *id*, set of *environment propositions* (accepted messages), set of *component properties* (messages the agent can send)
  - Stack: *(pop, push)* [popped, full]
  - Computational engine (executable temporal logic)
Executable Temporal Logic

- Agent specification as set of program rules of the form antecedent about past => consequent about present and future
- Declarative past and imperative future paradigm
# Temporal Connectives

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bigcirc \varphi$</td>
<td>$\varphi$ is true ’tomorrow’</td>
</tr>
<tr>
<td>$\bigotimes \varphi$</td>
<td>$\varphi$ was true ’yesterday’</td>
</tr>
<tr>
<td>$\lozenge \varphi$</td>
<td>at some time in the future, $\varphi$</td>
</tr>
<tr>
<td>$\Diamond \varphi$</td>
<td>always in the future, $\varphi$</td>
</tr>
<tr>
<td>$\lozenge \bullet \varphi$</td>
<td>at some time in the past, $\varphi$</td>
</tr>
<tr>
<td>$\Box \bullet \varphi$</td>
<td>always in the past, $\varphi$</td>
</tr>
<tr>
<td>$\varphi U \Psi$</td>
<td>$\varphi$ will be true until $\Psi$</td>
</tr>
<tr>
<td>$\varphi S \Psi$</td>
<td>$\varphi$ has been true since $\Psi$</td>
</tr>
<tr>
<td>$\varphi W \Psi$</td>
<td>$\varphi$ is true unless $\Psi$</td>
</tr>
<tr>
<td>$\varphi Z \Psi$</td>
<td>$\varphi$ is true since $\Psi$</td>
</tr>
<tr>
<td>Start</td>
<td>nullary operator, true only at the beginning</td>
</tr>
</tbody>
</table>
Example

rp(ask1, ask2)[give1, give2]:

- $\odot$ask1 $\Rightarrow$ $\Diamond$give1;
- $\odot$ask2 $\Rightarrow$ $\Diamond$give2;
- start $\rightarrow$ $\Box \neg (\text{give1} \land \text{give2})$.

rc1(give1)[ask1]:

- start $\rightarrow$ ask1;
- $\odot$ask1 $\rightarrow$ ask1.

rc2(ask1, give2)[ask2]:

- $\odot(\text{ask1} \land \neg \text{ask2}) \rightarrow$ ask2.
### Example Run:

<table>
<thead>
<tr>
<th>Time</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rp</td>
</tr>
<tr>
<td>1</td>
<td>ask1</td>
</tr>
<tr>
<td>2</td>
<td>ask1, ask2, give1</td>
</tr>
<tr>
<td>3</td>
<td>ask1, give2</td>
</tr>
<tr>
<td>4</td>
<td>ask1, ask2, give1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Practical Reasoning Agents

- What we want to achieve: deliberation
- How to achieve a state: means-end-analysis
The Procedural Reasoning System

- Georgeff and Lansky
- Belief-desire-intention architecture (BDI)

![Diagram of BDI architecture]

Beliefs → Interpreter → Plans

Desires → Interpreter → Intentions

data input

action output