Evolution of Programming Languages: Away from Objects

Distributed Systems
Sistemi Distribuiti

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2012/2013
Evolution of Programming Languages: The Picture

- [Odell, 2002]
Evolution of Programming Languages: Dimensions

**Historical evolution**
- Monolithic programming
- Modular programming
- Object-oriented programming
- Agent programming

**Degree of modularity & encapsulation**
- Unit behaviour
- Unit state
- Unit invocation
Monolithic Programming

- The basic unit of software is the whole program
- Programmer has full control
- Program’s state is responsibility of the programmer
- Program invocation determined by system’s operator
- Behaviour could not be invoked as a reusable unit under different circumstances
  - modularity does not apply to unit behaviour
Monolithic Programming

Encapsulation? There is no encapsulation of anything, in the very end

<table>
<thead>
<tr>
<th>Unit Behavior</th>
<th>Monolithic Programming</th>
<th>Modular Programming</th>
<th>Object-Oriented Programming</th>
<th>Agent Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nonmodular</td>
<td>Modular</td>
<td>Modular</td>
<td>Modular</td>
</tr>
<tr>
<td>Unit State</td>
<td>External</td>
<td>External</td>
<td>Internal</td>
<td>Internal</td>
</tr>
<tr>
<td>Unit Invocation</td>
<td>External</td>
<td>External (CALLed)</td>
<td>External (message)</td>
<td>Internal (rules, goals)</td>
</tr>
</tbody>
</table>
The Prime Motor of Evolution

Motivations

- Larger memory spaces and faster processor speed allowed program to became more complex

Results

- Some degree of organisation in the code was required to deal with the increased complexity
Modular Programming

- The basic unit of software are structured loops / subroutines / procedures / ...
  - this is the era of procedures as the primary unit of decomposition
- Small units of code could actually be reused under a variety of situations
  - modularity applies to subroutine’s code
- Program’s state is determined by externally supplied parameters
- Program invocation determined by CALL statements and the likes
### Encapsulation

Encapsulation applies to *unit behaviour* only.

<table>
<thead>
<tr>
<th></th>
<th>Monolithic Programming</th>
<th>Modular Programming</th>
<th>Object-Oriented Programming</th>
<th>Agent Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit Behavior</strong></td>
<td>Nonmodular</td>
<td>Modular</td>
<td>Modular</td>
<td>Modular</td>
</tr>
<tr>
<td><strong>Unit State</strong></td>
<td>External</td>
<td>External</td>
<td>Internal</td>
<td>Internal</td>
</tr>
<tr>
<td><strong>Unit Invocation</strong></td>
<td>External (CALled)</td>
<td>External (message)</td>
<td>Internal (rules, goals)</td>
<td></td>
</tr>
</tbody>
</table>
Object-Oriented Programming

- The basic unit of software are objects & classes
- Structured units of code could actually be reused under a variety of situations
- Objects have local control over variables manipulated by their own methods
  - variable state is persistent through subsequent invocations
  - object’s state is encapsulated
- Object are passive—methods are invoked by external entities
  - modularity does not apply to unit invocation
  - object’s control is not encapsulated
Object-Oriented Programming

Encapsulation? Encapsulation applies to unit behaviour & state
Agent-Oriented Programming

- The basic unit of software are agents
  - encapsulating everything, in principle
    - by simply following the pattern of the evolution
  - whatever an agent is
    - we do not need to define them now, just to understand their desired features

- Agents could in principle be reused under a variety of situations
- Agents have control over their own state
- Agents are active
  - they cannot be invoked
  - agent’s control is encapsulated
Agent-Oriented Programming

Encapsulation? Encapsulation applies to unit *behaviour, state & invocation*

<table>
<thead>
<tr>
<th>Monolithic Programming</th>
<th>Modular Programming</th>
<th>Object-Oriented Programming</th>
<th>Agent Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit Behavior</td>
<td>Nonmodular</td>
<td>Modular</td>
<td>Modular</td>
</tr>
<tr>
<td>Unit State</td>
<td>External</td>
<td>Internal</td>
<td>Internal</td>
</tr>
<tr>
<td>Unit Invocation</td>
<td>External (CALled)</td>
<td>External (message)</td>
<td>Internal (rules, goals)</td>
</tr>
</tbody>
</table>
Before we define agents...

- ...agents are *autonomous* entities
  - encapsulating their thread of control
  - they can say “Go!”
- ...agents cannot be invoked
  - they can say “No!”
  - they do not have an interface, nor do they have methods
- ...agents need to encapsulate a criterion for their activity
  - to self-govern their own thread of control
Dimensions of Agent Autonomy

Dynamic autonomy

- Agents are *dynamic* since they can exercise some degree of activity
  - they can say “Go!”
- From passive through reactive to active

Unpredictable / non-deterministic autonomy

- Agents are *unpredictable* since they can exercise some degree of deliberation
  - they can say “Go!”, they can say “No!”
  - and also because they are “opaque” — may be unpredictable to external observation, not necessarily to design
- From predictable through partially predictable to unpredictable
Objects vs. Agents: Interaction & Control

Message passing in object-oriented programming
- Data flow along with control
  - Data flow cannot be designed as separate from control flow
- A too-rigid constraint for complex distributed systems...

Message passing in agent-oriented programming
- Data flow through agents, control does not
  - Data flow can be designed independently of control
- Complex distributed systems can be designed by designing information flow
Agents communicate

- Interaction between agents is a matter of exchanging information
  - toward Agent Communication Languages (ACL)
- Agents can be involved in conversations
  - they can be involved in associations lasting longer than the single communication act
  - differently from objects, where one message just refer to one method
Philosophical Differences [Odell, 2002]

Decentralisation
- Object-based systems are completely pre-determined in control. Control is essential centralised at design time
- Agent-oriented systems are essentially decentralised in control

Multiple & dynamic classification
- Once created, objects typically have an unmodifiable class
- After creation, agents can change their role, task, goal, class, ..., according to their needs and to the ever-changing structure of the surrounding environment
Instance-level features

- Objects are class instances whose features are essentially defined by classes themselves once and for all.
- Agents features can change during execution, by adaptation, learning, ...

Small in impact

- Loosing an object in an object-oriented system makes the whole system fail, or at least raise an exception.
- Loosing an agent in a multi-agent system may lead to decreases in performance, but agents are not necessarily single points of failure.
Philosophical Differences [Odell, 2002] III

Small in time

- Garbage collection is an extra-mechanism in object-oriented languages for taking advantage of disappearing objects.
- Disappearing agents can simply be forgotten naturally, with no need of extra-mechanisms.

Small in scope

- Objects can potentially interact with the whole object space, however their interaction space is defined once and for all at design time: this defines a sort of local information space where they can retrieve knowledge from.
- Agents are not omniscient and omnipotent, and typically rely on local sensing of their surrounding environment.
### Emergence
- Object-based systems are essentially predictable
- Multi-agent systems are intrinsically unpredictable and non-formalisable and typically give raise to emergent phenomena

### Analogies from nature and society
- Object-oriented systems have not an easy counterpart in nature
- Multi-agent systems closely resembles existing natural and social systems
Towards the Coexistence of Agents and Objects

Final issues from [Odell, 2002]

- Should we *wrap* objects to *agentify* them?
- Could we really *extend objects* to make them agents?
- How are we going to *implement the paradigm shift*, under the heavy weight of legacy?
  - technologies, methodologies, tools, human knowledge, shared practises, ...

Answers are to be found in the remainder of the course

- So, stay tuned!
Evolution of Programming Languages: Away from Objects

Distributed Systems
Sistemi Distribuiti

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2012/2013