Objectives

- Comparing components and agents
- Independent approaches...
- ...but some common goals for software:
  - Composable
  - Adaptable
  - "Better"
- Considering them within the history/evolution of programming
- What can agents bring to components?
  - Semantic coupling vs syntactic coupling
  - Autonomy
  - Adaptability
- What can components can bring to agents?
  - Self-containedness
  - Conformance control
  - Building blocks
Outline

- Components
- Agents and Multi-Agent Systems (MAS)
- Evolution of programming
- What agents can bring to components?
  - Autonomy/Evolvability
  - Assistance to Assemblage
    » Ex: The COGENTS project
- What components can bring to agents?
  - Self-containedness
  - Architectural support
    - macro-level, ex: role/agent conformance control
    - micro-level: agent architecture
- Component-based agent architectures
  - Various decomposition rationales (levels, modules, behaviors...)
  - Ex: behavior decomposition: the MALEVA agent component model
- Conclusion

(Software) Components

- Software components
- Inspiration from electronics - Integrated Circuits
- Objective: composition and reuse of software components
- Objective: ease
  - Replacement
  - Addition
  - Removal
    of
    - Components
    - Connectors
Outline

• Components

• Agents and Multi-Agent Systems (MAS)

• Evolution of programming

• What agents can bring to components?
  – Autonomy/Evolvability
  – Assistance to Assemblage
    » Ex: The COGENTS project

• What components can bring to agents?
  – Self-containedness
  – Architectural support
    – macro-level, ex: role/agent conformance control
    – micro-level: agent architecture

• Component-based agent architectures
  – Various decomposition rationales (levels, modules, behaviors...)
  – Ex: behavior decomposition: the MALEVA agent component model

• Conclusion

Agents and Multi-Agent Systems (MAS) - AI View

• Limits of direct control/programming approach
  – e.g., autonomous space probe, Internet search,
  – world-level distributed computing

• Delegation of mission - Initiative

• Agents: autonomous entities
  • rational, deliberative...

• Multi-Agent System: distributed interacting agents
  – Distributed AI (e.g., RoboCup) VS Traditional AI (e.g., chess)

• Assistant agent VS single artificial expert (Traditional AI)
Agents and Multi-Agent Systems (MAS) - Software view

- **Limits of direct control/programming approach**
  - e.g., autonomous space probe, Internet search, world-level distributed computing

- **Delegation of mission - Initiative**

- **Agents:** autonomous entities/software components
  - Reactive or/and proactive (e.g., goal-driven)

- **Knowledge-level coupling vs data-level (typing) coupling**

- **Adaptive vs Defensive approach (static verification)**

- **Bottom-up (emergent) VS/AND top-down (Architecture Description Languages) design/organization**

---

Multi-Agent Systems (MAS)

- **Autonomous entities**
  - Reactive or/and proactive (e.g., goal-driven)

- **Coordination**
  - Protocols
    - coordination, negotiation, auction...
    - e.g., Contract Net Protocol/Call for Proposals
  - Shared knowledge,
    - e.g., joint intentions, exchange of plans...

- **Organizations**
  - Division of labor (roles)
  - Inter-agent dependencies
  - Collective actions
  - Regulation (e.g., norms)

- **Meta-level**
  - Reasoning about and acting upon
    - Action
      - Individual
      - Collective
    - Interaction
    - Coordination
    - Organization
**Outline**

- Components
- Agents and Multi-Agent Systems (MAS)
- Evolution of programming

**What agents can bring to components?**
- Autonomy/Evolvability
- Assistance to Assemblage
  - Ex: The COGENTS project

**What components can bring to agents?**
- Self-containedness
- Architectural support
  - macro-level, ex: role/agent conformance control
  - micro-level: agent architecture

**Component-based agent architectures**
- Various decomposition rationales (levels, modules, behaviors...)
- Ex: behavior decomposition: the MALEVA agent component model

**Conclusion**
1st Axis - Action selection

<table>
<thead>
<tr>
<th>Behavior</th>
<th>&quot;Monolithic&quot; programming</th>
<th>Modular programming</th>
<th>Object-oriented programming</th>
<th>Agent-oriented programming</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g., Fortran</td>
<td>e.g., Pascal</td>
<td>e.g., Java</td>
<td>e.g., AgentSpeak</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Non modular</th>
<th>modular</th>
<th>modular</th>
<th>modular</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>external</td>
<td>external</td>
<td>internal</td>
<td>internal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Invocation (and action selection)</th>
<th>jump (goto)</th>
<th>procedure call</th>
<th>method call</th>
<th>agent decision (ex: goal-driven)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>external</td>
<td>external</td>
<td>external</td>
<td>internal</td>
</tr>
</tbody>
</table>

Evolution of programming

Abstraction level

Action selection flexibility ("ever late time binding")

Jean-Pierre Briot
Seminário de pesquisa LES/DI/PUC-Rio 25/11/05

11

12
2nd Axis - Coupling flexibility

<table>
<thead>
<tr>
<th></th>
<th>objects</th>
<th>components</th>
<th>agents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>structure</strong></td>
<td>implicit, internal</td>
<td>explicit, external</td>
<td>implicit, external</td>
</tr>
<tr>
<td></td>
<td>(object references)</td>
<td>(connectors)</td>
<td>(indexed by organizational roles)</td>
</tr>
<tr>
<td><strong>communication</strong></td>
<td>procedure call</td>
<td>unidirectional</td>
<td>protocol</td>
</tr>
<tr>
<td></td>
<td>(bidirectional, return</td>
<td>(events) or bidirectional</td>
<td></td>
</tr>
<tr>
<td></td>
<td>value)</td>
<td></td>
<td>protocol</td>
</tr>
<tr>
<td><strong>synchronization</strong></td>
<td>synchronous</td>
<td>synchronous or asynchronous</td>
<td>protocol</td>
</tr>
</tbody>
</table>

Evolution of programming

![Diagram showing evolution of programming](image)

**Abstraction level**
- agents, intentions, plans
- models, ontologies
- objects, messages
- data structures
- bits

**Action selection flexibility**
- Fortran
  - modules
  - jump (goto)
- procedure call
- method call
- agent decision

**Coupling flexibility**
- actors
- components
- agents
3rd Axis - Abstraction level

- Agents, not purely data/procedural
  - knowledge (beliefs, goals...)

- Semantic/Knowledge-level coupling rather than data-type-level coupling

- Communication (e.g., FIPA ACL vs OMG CORBA)
  - content language (e.g., KIF, FIPA SL)
  - performative (intention of communication, e.g., inform, recruit)
  - ontology
  - protocol

Outline

- Components
- Agents and Multi-Agent Systems (MAS)
- Evolution of programming

  - What agents can bring to components?
    - Autonomy/Evolvability
    - Assistance to Assemblage
      - Ex: The COGENTS project

  - What components can bring to agents?
    - Self-containedness
    - Architectural support
      - macro-level, ex: role/agent conformance control
      - micro-level: agent architecture

  - Component-based agent architectures
    - Various decomposition rationales (levels, modules, behaviors...)
    - Ex: behavior decomposition: the MALEVA agent component model

  - Conclusion
What agents could bring to components?

- More flexibility for assembling (match-making)
- Mechanisms (reorganization) for dynamic reconfiguration
- More “intelligent” behavior (intelligent/adaptive cooperative components)

Example: components match-making

Petrochemical process engineering (design, simulation, control)
Initial step: Interoperability:
CAPE-OPEN Project [Braunschweig et al. 02]

- Componentification of Process units
- Interoperability
- Interfaces standards
- OTS Components

Second step: Assistance to Assemblage:
COGENTS Project [Braunschweig et al. 04]

- Match-making
- Assistance for assemblage
- Instantiation (actual Software products)

E.g., LARKS matchmaking [Sycara et al. 98]
Outline

• Components

• Agents and Multi-Agent Systems (MAS)

• Evolution of programming

• What agents can bring to components?
  – Autonomy/Evolvability
  – Assistance to Assemblage
    » Ex: The COGENTS project

• What components can bring to agents?
  – Self-containedness
  – Architectural support
    – macro-level, ex: role/agent conformance control
    – micro-level: agent architecture

• Component-based agent architectures
  – Various decomposition rationales (levels, modules, behaviors...)
  – Ex: behavior decomposition: the MALEVA agent component model

• Conclusion

"Self-containedness"

• Includes all the code

• "Ready to use"

• “Ready to deploy”

• Includes documentation
Architectural support

- At the macro / system / organizational level

- At the micro / (single) agent level

System level architecture

- Software architectures (and components)
  - explicit
  - rational
  - explicit coupling
    - data-level (interfaces, typing)
    - communication-level (connectors)

- Agent organizations (cognitive)
  - explicit
  - rational
  - semantic/knowledge coupling
    - reified
    - evolutive (reorganization)

- Agent organizations (reactive)
  - bottom up / emergent (e.g., ant societies)
  - and conformant / top-down ([Cardon 99])
Conformance of an agent to a role

How can we make sure (or estimate) that an agent may (or will be able to) fulfill a role?

Checking the conformance of an agent to a role

- **Role**
  - place holder
  - requirements / capabilities
    - structure
      - procedures
      - knowledge
      - coordination
      - physical (e.g., for locomotion)
    - activity
      - behavior
      - coordination
      - regulation

- **Conformance problem**
  - static
    - procedures signatures / typing
    - contracts
    - compatibility with other roles already acquired (MOISE+ [Hübner et al. 02])
  - dynamic
    - possible dynamic acquisition (procedures, knowledge, protocols)
    - integration test [Rodrigues 05]
    - deontic specification (MOISE+ [Hübner et al. 02])
    - monitoring/evaluation mechanisms
Outline

• Components

• Agents and Multi-Agent Systems (MAS)

• Evolution of programming

• What agents can bring to components?
  – Autonomy/Evolvability
  – Assistance to Assemblage
    » Ex: The COGENTS project

• What components can bring to agents?
  – Self-containedness
  – Architectural support
    – macro-level, ex: role/agent conformance control
    – micro-level: agent architecture

• Component-based agent architectures
  – Various decomposition rationales (levels, modules, behaviors...)
  – Ex: behavior decomposition: the MALEVA agent component model

• Conclusion

Architectural support for an agent (agent level)

• Central issue: action selection (as for robots)

• More complex than for objects/components:
  – not just procedural (e.g., reasoning)
  – various inputs (environnement, communication...)
  – pro-activity (vs simple reactivity)
  – various levels (self, agents, organization)
  – knowledge (vs data)

• Architecture of an agent:
  • the software structure in charge of that action selection
  • functions of the agent and their interactions
Rationale (tentative typology) for (one) agent architectural decomposition

- Analog to software architectural styles (layers, pipes&filters...)

- (computational) Cycle
  - e.g., perception, mental state update, generating commitments, action
    » e.g., AOP architecture

- Viewpoints and types of processing
  - e.g., interaction, organization, environment
    » e.g., Volcano architecture

- Levels
  - e.g., world level, individual level, social level
    » e.g., InteRRaP architecture

- Behaviors
  - e.g., gradient following, obstacle avoidance, random move...
    » e.g., subsumption architecture

Cycle decomposition: “Horizontal” modular architectures

- one layer

- decision/action cycle

```
perception → mental states update → commitments generation → action
```

environment
Ex. of Cycle decomposition:
AOP (Agent Oriented Programming) Architecture

[Shoham 93]
data-driven

Ex. of Viewpoint decomposition: Vulcano [Ricordel 02]

- Vowels decomposition model [Demazeau 01]:
- A(gent)
- E(nvironnement)
- I(teraction)
- O(rganization)
- Interfacing wrappers/adapters (ad-hoc)
**MAST [Vercouter 04]**

- provided roles and required roles
- sent events and handled events
- delegation
- priority

![Diagram of MAST](image)

**DESIRE [Brazier et al. 95-01]**

- formal specification
- Generic Agent Model (GAM)
- retro-engineering of some architectures
- (e.g., BDI) and applications (e.g., ARCHON)

![Diagram of DESIRE](image)
Modules decomposition, Ex: DIMA architecture [Guessoum 99]

ATN = Augmented Transition Network (automata)

Meta level
Adaptative control

Behavior level

Reactives Module

Control
Objects

MetaRules

Rules

Deliberative Module

Ex. of Level decomposition: InteRRap [Müller 94]

- 3 levels/layers activated in //
  - behavior - beliefs about the state of the environment
  - local planning - beliefs about oneself
  - cooperative planning - beliefs about and commitments with other agents
Ex. of Behavior decomposition: Subsumption architecture [Brooks 86]

- components activated in parallel
- competitive and hierarchical
- priorities and inhibitions:
  - taking over input of lower component
  - inhibiting output of lower component
- hard-wired

![Diagram of behavior decomposition]

Other: Evolvable architectures [Meyer et al. 98]

- Genetic programming
- Evolution of the development program
- instructions: DIVIDE, GROW, DRAW...

- Modular design/construction:
  - Black network: walk
  - Red network: obstacle avoidance
Reuse of architectural components

• Cycle
  – e.g., AOP
  – only little decomposition
  – often only conceptual, no implementation decoupling

• Viewpoints
  – e.g., Volcano
  – replacing a brick -> replace the adaptors

• Levels
  – e.g., InteRRaP
  – often only conceptual, no implementation decoupling

• Behaviors
  – e.g., Subsumption architecture
  – hard-wired
  – very difficult to evolve

Architectural model versus Component model

• Existence of an architecture restrains the possible combination of components
  – cons: constraints
  – pros: constraints ! (structure)
    » Reuse of (stable) architecture is more easy
    » Cf. Frameworks - “Is reusable only what has already been reused” [Johnson]

• But difficult to evolve the architecture itself (e.g., add a component)

Radical option:

• No more architecture

• Just a component model (like ex: JavaBeans)
Rationale for agent architectural decomposition (2)

• Tools/techniques
  – e.g., backpropagation, bayesian, time series, rules...
    » e.g., ABLE architecture

• Protocol components
  – e.g., Agentalk, SCD

• Behaviors
  – e.g., gradient following, obstacle avoidance, random move...
    » e.g., MALEVA component architecture

Tools (tool box) decomposition, ex: ABLE architecture [Bigus et al. 02]

IBM Autonomic computing programme

Java Beans-based implementation

• Data beans
  e.g., TimesSeriesFilter
• Learning beans
  e.g., BackPropagation
• Rule beans
  e.g., FuzzyForwardChaining
• Specific beans
  e.g., GenericSearch

E.g., rule beans
Reusing protocol components

• Extensions of the Contract Net Protocol (CNP)

• Agentalk [Kuwabara et al. 95]
  - inheritance (e.g., directed-award-CNP)
  - customization interface

![Script cnrnt-manager](image)

- added output message port

• SCD [Yoo et al. 98]
  - inheritance (e.g., time-out-CNP)
  - composition (e.g., iterated-CNP)

![Script cnrnt-manager](image)

- new component managing iteration of proposal

• also XMLaw [Carvalho et al. 04]
The MALEVA agent component model [Lhuillier et al. 98]

- Domain: multi-agent simulation
  - e.g., traffic simulation, eco-systems, population micro-simulation...

- Unit of decomposition: agent behavior

- Assembling behaviors into more complex behaviors
  - concept of composite component (behavior)

- Supports behavior dynamic change
  - e.g., from an egg, to a larva, to a worker ant

- Distinction between
  - data flow
  - control flow

- JavaBeans-based (re)implementation

General agent architecture

- Sensors
- Connexions
- Sub-behaviors
- Effectors
- Behavior
Data flow and control flow: ports and connexions

Sequence

Data flow and control flow: ports and connexions

Concurrency
A first example: Prey

- if the Prey detects a Predator, it flees away
- otherwise, it moves randomly

Control components

- dispatch of control flow

- Switch
  - reifies in a component
  - traditional conditional control structure
  - (if then else)
  - 
  - 
  - 
  - other control components:

  - control structures
    - e.g., Repeat
  - synchronization
    - e.g., Sync (synchronization barrier)
Reuse of a Prey: Predator

- if the Predator detects a prey, it follows the prey
- otherwise, it acts as a Prey (cannibalism among Predators)

Importance of composite component

- Two kinds of composition:
  - functional composition (assemblage)
  - structural composition (composite component, information hiding, black box)
2nd Example: Ant Nests Simulation

- Reengineering of MANTA simulation testbed [Drogoul 93]

- Redesign/construction of ant behaviors using MALEVA [Guillemet et al. 98]

Ex. of hierarchical behavior: Ant Worker

Living agent (pattern)
Default random move (pattern)
Following gradient
Dynamicity (dynamic change of behavior)

- e.g., egg -> larva -> ant

- behavior server meta-component
  - set up future behavior
  - check what components to keep, to add, to remove
  - install connexions

Advantages of explicit control flow

- decoupling activation logic from functionality

- more genericity

- fine grain control of intra-agent scheduling
  (specification of temporal dependencies)
  see next example/slides
3rd Example: population reproduction/evolution

Mating

Birth

Divorce

3 behaviors/components
(probablistic state change):

getMarried  newBaby  divorce

Issue for the designer of the model/simulation:
(Note: often not an expert programmer)

in what order should we activate these behaviors?

Impact: Scheduling bias
Specification of intra-agent temporal dependencies [Meurisse 04]

Possible scheduling bias on number of babies

Reingineering of existing behavioral code [Meurisse 04]

- a Java class (name)
- a method (name)
- method signature
  e.g., position Follow(position p)

- typed ports
- one FIFO for each data input port
Reuse: Design Patterns [Guillemet et al. 99]

• E.g., “Living Agent” (ageing agent) pattern
  - used for egg,
  - larva,
  - ant worker,
  - queen...

• Actually, we offer more
  • than just a design pattern:
  • a black box micro-framework - parameterized component
    - with (in this case) one hot-spot (Behavior)

From control flow graph to process algebra term

Even with the hierarchy of components (composite components), which helps at encapsulate some complexity of the control flow graph, specifying it is precise but low level

An alternative direction could then be in using a formalism (coordination language), to specify control coordination language (a very fine grained one)

Process algebra, e.g., CCS

Pi-calculus to handle dynamicity

a compact term

isPrey.Follow || isPredator.Flee || (isNoPrey.RandomMove + isNoPredator.RandomMove)
Outline

- Components

- Agents and Multi-Agent Systems (MAS)

- Evolution of programming

- What agents can bring to components?
  - Autonomy/Evolvability
  - Assistance to Assemblage
    - Ex: The COGENTS project

- What components can bring to agents?
  - Self-containedness
  - Architectural support
    - macro-level, ex: role/agent conformance control
    - micro-level: agent architecture

- Component-based agent architectures
  - Various decomposition rationales (levels, modules, behaviors...)
  - Ex: behavior decomposition: the MALEVA agent component model

Conclusion

Conclusion on MALEVA

- components can be useful to help at decomposing/recomposing agent architectures

- fine grained (behaviors)
  - but optimizations possible

- composite components hierarchy

- dynamic change of behaviors

- data flow and control flow for decoupling activation from functionality

- typed ports

- libraries of
  - behaviors
  - parameterized behaviors (e.g., ageing agent)
General Conclusion (Components & Agents)

• Dual movement:
  – Distributed systems/applications are getting more adaptable/dynamic
    » dynamic reconfiguration
    » more semantic support
    » e.g., GRID and MAS: “Brain meets brawn” [Foster et al. 03]
  – Agents and multi-agent systems have greater software maturity
    » deployment
    » configuration
    » life cycle

• Reuse is difficult (no free lunch)
  – components
  – but also:
    » inheritance, parameterization, frameworks, delegation
    » reflective architectures, aspects [Garcia et al. 04], meta-models [Silva et al. 04]...

• Alternative to distributed components: Web services
  – simpler infrastructure (e.g., vs Corba Component Model)
    » e.g., Web-service-based MAS interoperability [Melliti et al. 04]