

Agents and Components: Comparison and Perspectives

A Study of the Evolution of Software Abstractions

Jean-Pierre Briot

Jean-Pierre.Briot@lip6.fr
<http://www-desir.lip6.fr/~briot/>

Laboratoire d'Informatique de Paris 6 (LIP6)
Université Pierre et Marie Curie (Paris 6) – CNRS



Laboratório de Engenharia de Software (LES)
Departamento de Informática, PUC-Rio



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
- Application to simulation
 - An Experience in Using Components to Construct and Compose Agent Behaviors for Agent-based Simulations

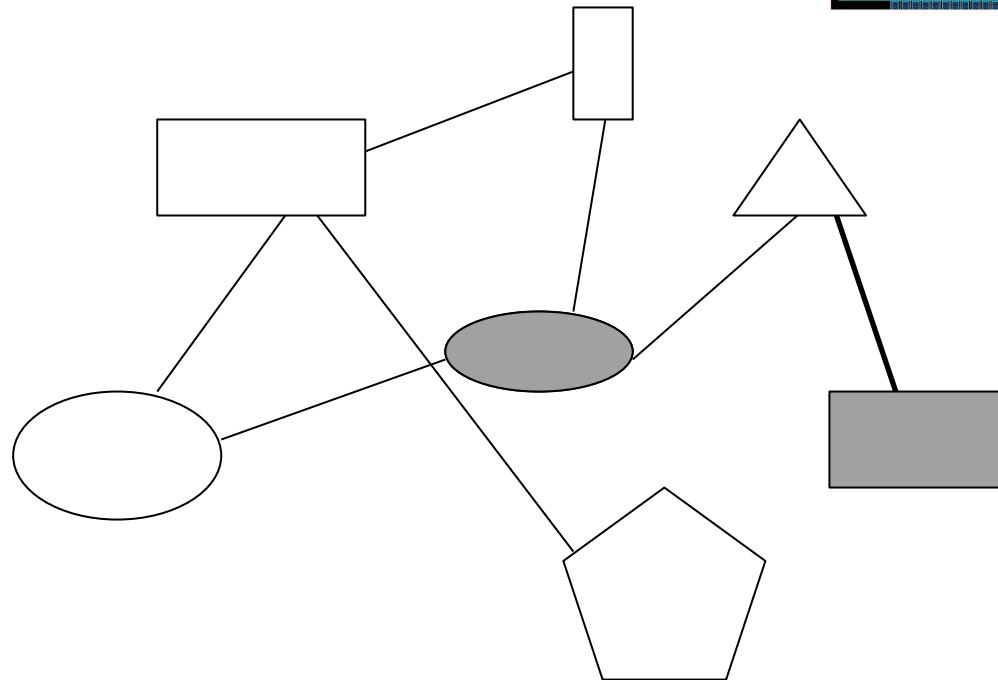
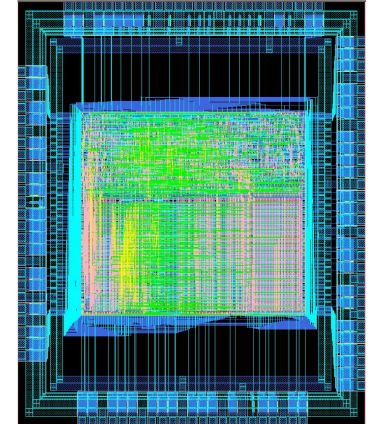
Outline



- 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) – Software view

- Limits of direct control/programming approach

- e.g., autonomous space probe, Internet search,
- world-level distributed computing

- 1) Delegation of mission – Initiative

- Autonomy

- Agents: autonomous entities/software components

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

- Limits of single entity solution

- One single agent cannot be omniscient and omnipotent

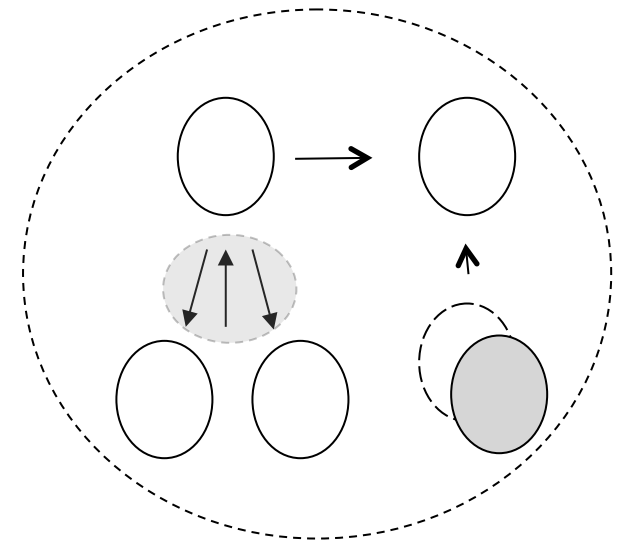
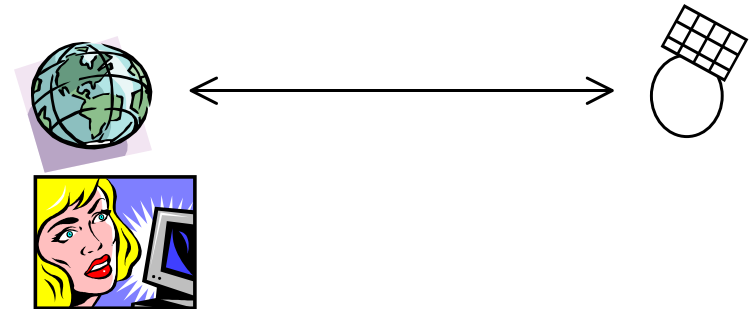
- 2) (Multi-)Agents: collaborative entities/software components

- Coordination

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

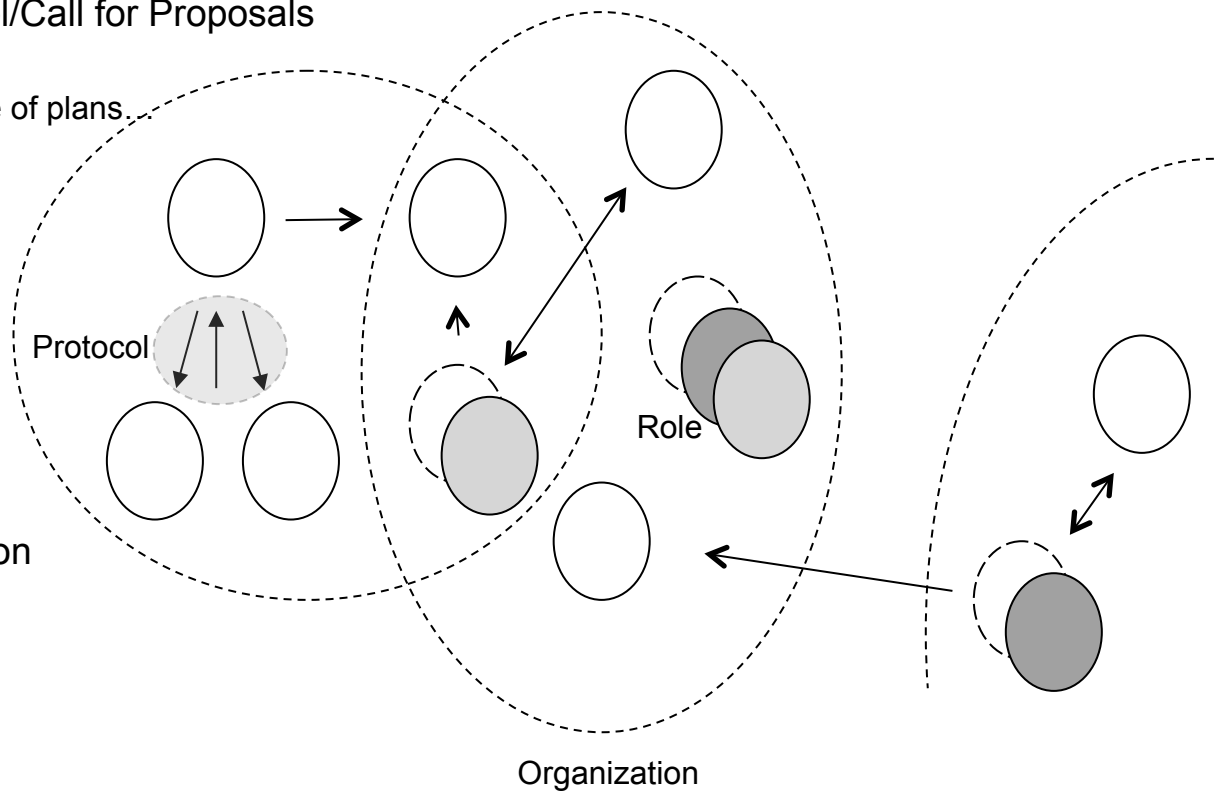
- Adaptative vs Defensive approach (static verification)

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



Multi-Agent Systems (MAS)

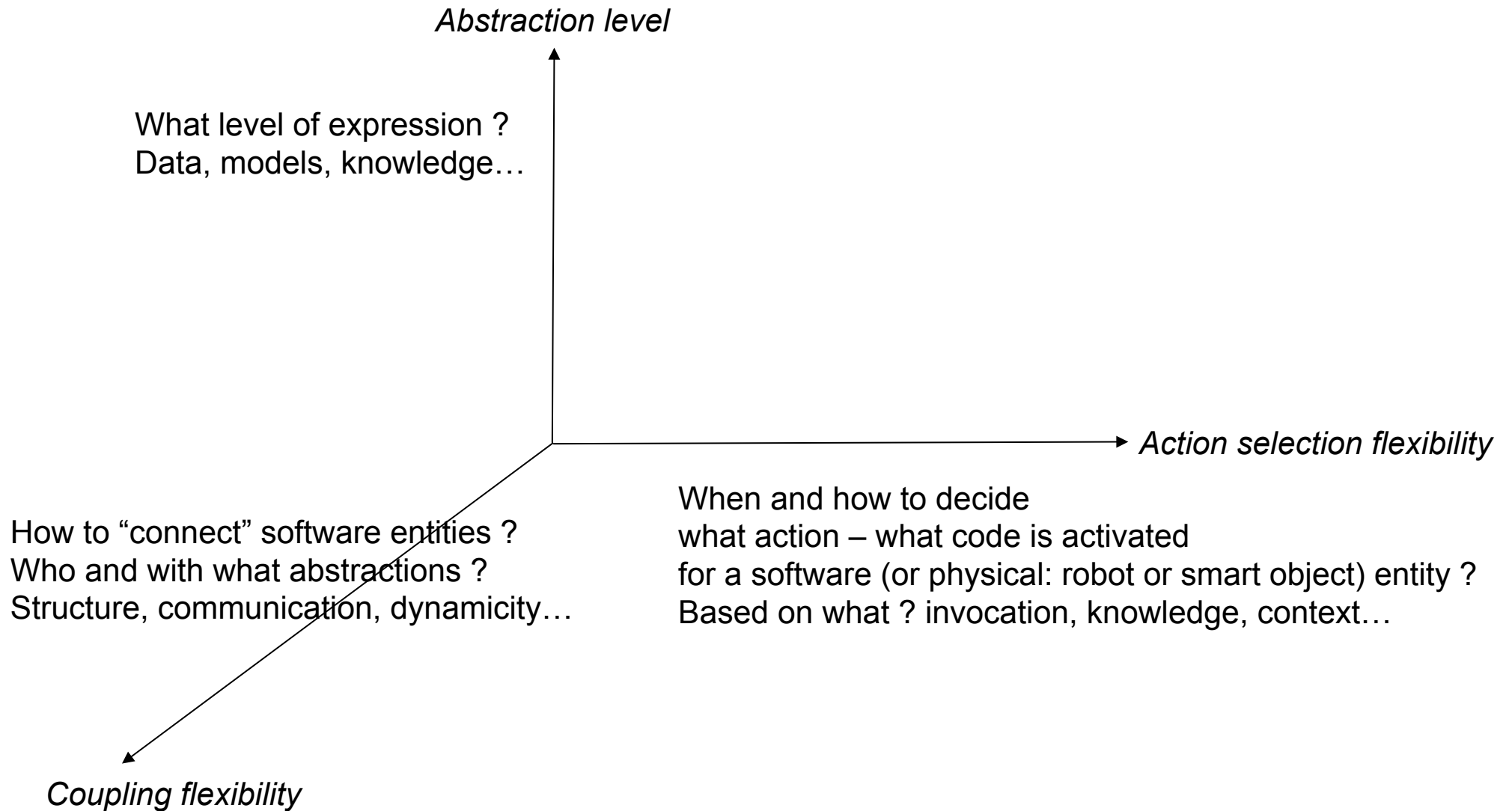
- Autonomous entities
 - Reactive or/and proactive (e.g., goal-driven)
- Coordination
 - Protocols
 - » coordination, negotiation, auction...
 - » e.g., Contract Net Protocol/Call for Proposals
 - Shared knowledge,
 - » e.g., joint intentions, exchange of plans.
- Organizations
 - Division of labor (roles)
 - Inter-agent dependencies
 - Collective actions
 - Regulation (e.g., norms)
- Meta-level
 - Reasoning about and acting upon
 - » Action
 - Individual
 - collective
 - » Interaction
 - » Coordination
 - » Organization
 - For control, dynamic adaptation...



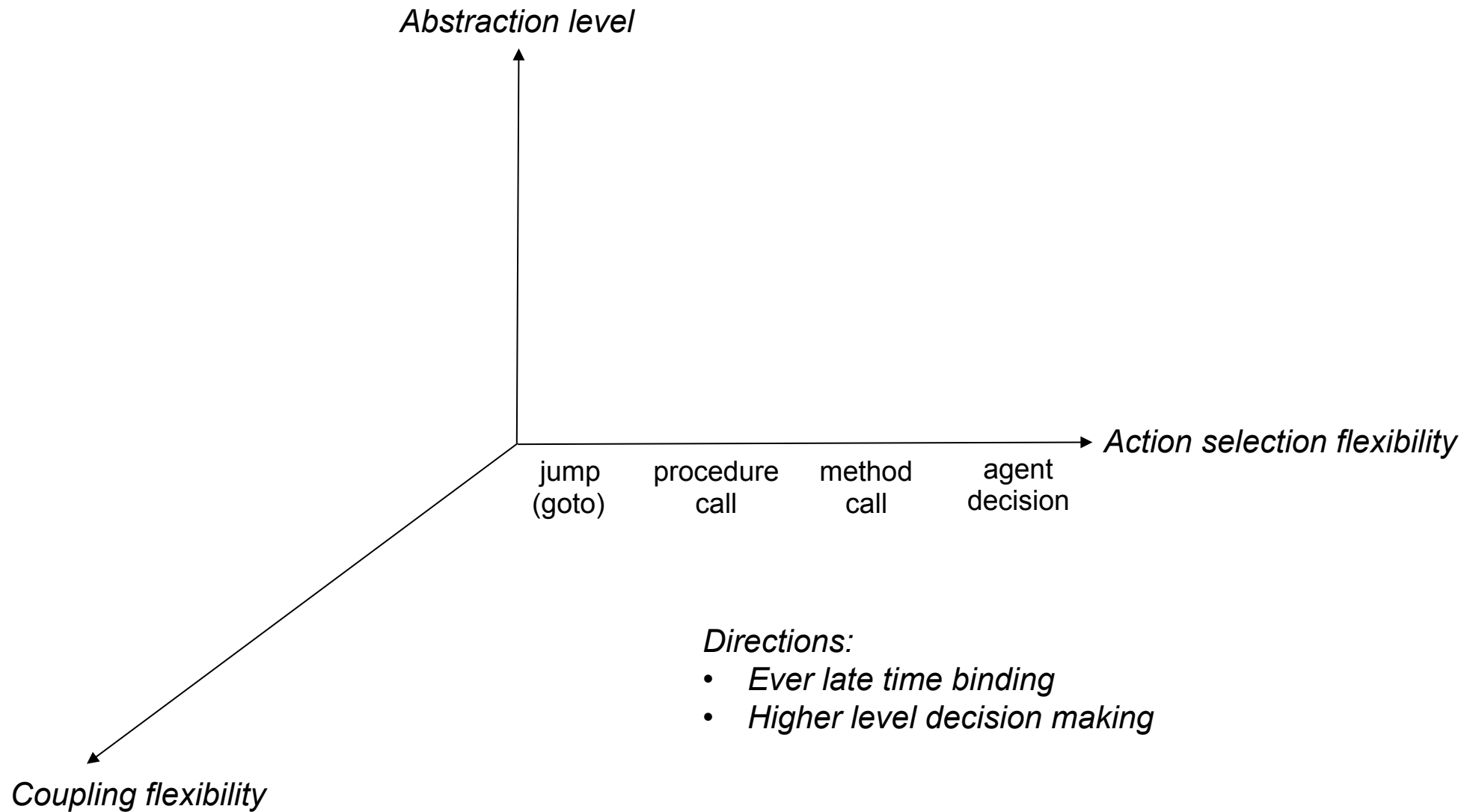
Outline

- 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

Evolution of programming Referential (3 dimensions)



Evolution of programming



1st Axis - Action selection

	<i>"Monolithic" programming</i>	<i>Modular programming</i>	<i>Object-oriented programming</i>	<i>Agent-oriented programming</i>
	e.g., Fortran	e.g., Pascal	e.g., Java	e.g., AgentSpeak
<i>Behavior</i>	global	<u>modular</u>	modular	modular
<i>State</i>	global & external	modular & external	modular & <u>internal</u>	modular & internal
<i>Invocation (and action selection)</i>	global & static & numeric jump (goto)	external & static & <u>symbolic</u> procedure call	external & <u>dynamic</u> (class vs type) method call	<u>internal</u> & dynamic (<u>action vs</u> <u>design time</u>) ex: goal-driven

(adapted from [Odell 99])

Early/Static vs Dynamic/Late Binding

Library

- Class A
 - Method m
 - » <Code m1>
 - Method p
 - » self.m

Refining the Library

- Class B subclassOfClass A
 - Method m
 - » <Code m2>

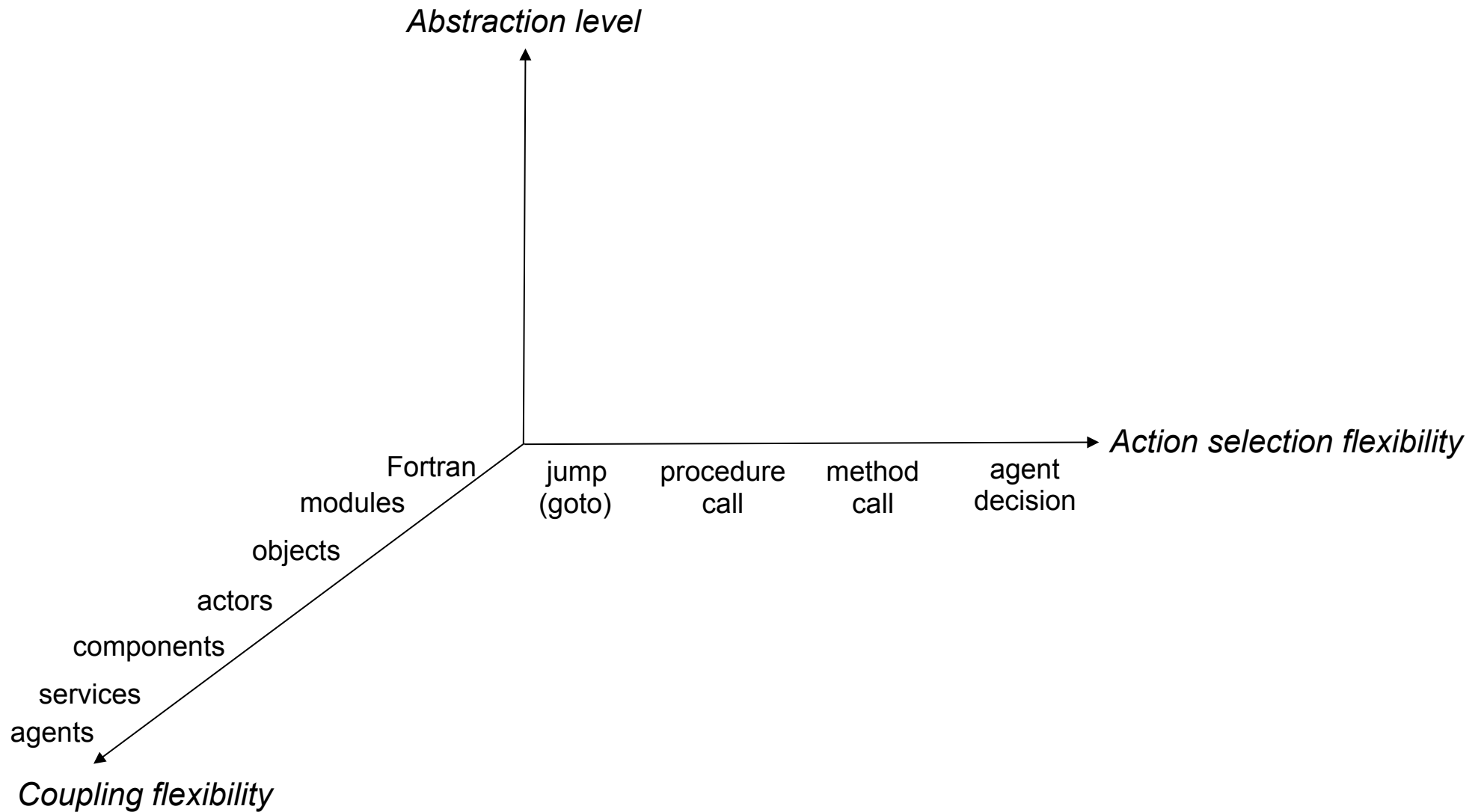
Using the Refined Library

- B : b
- B := new B
- b.p
 - ? m1 or m2 ?

ClientServer vs Autonomy

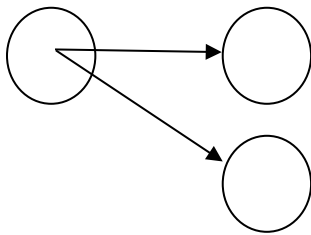
- object.message
- agent.message
- Agents can be thought of as objects that can say "No" – as well as "Go"
[Odell 2002]
 - Can defer processing of the request
 - Can take action without being requested to (Autonomy)
 - Can use arbitrary internal reasoning
 - » Including asking other agents, searching for information, etc.
- Objects : procedural
 - How to do has already been defined by the programmer
- Agents : Knowledge and social-based
 - May take initiative on how to proceed
- Emergence vs Determinism
- Interaction vs Algorithm [Wegner, 1997]

Evolution of programming



2nd Axis - Coupling flexibility

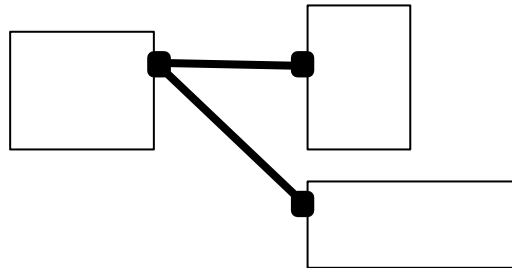
objects
& actors



references

implicit
& internal

components

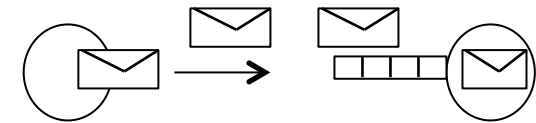


connectors

explicit
& external

components are compositional

actors



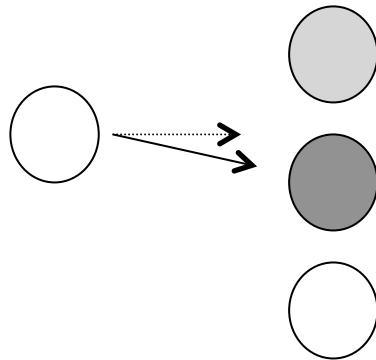
sending reception processing

message passing

temporal
decoupling

2nd Axis - Coupling flexibility

services



consultation... ...and selection

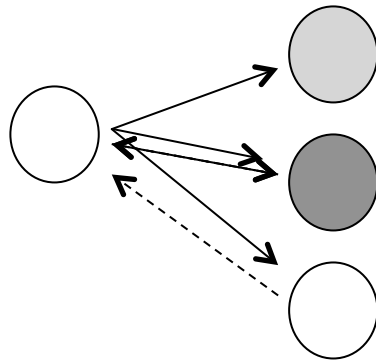
implicit

& dynamic

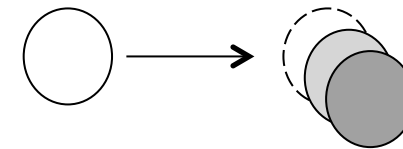
connexions managed by services

2nd Axis - Coupling flexibility

multi-agent
protocols



multi-agents
organizations



call

bids

selection

contract

role-indexed communication

indirect

& dynamic

indirect

& dynamic

connexions managed by agents

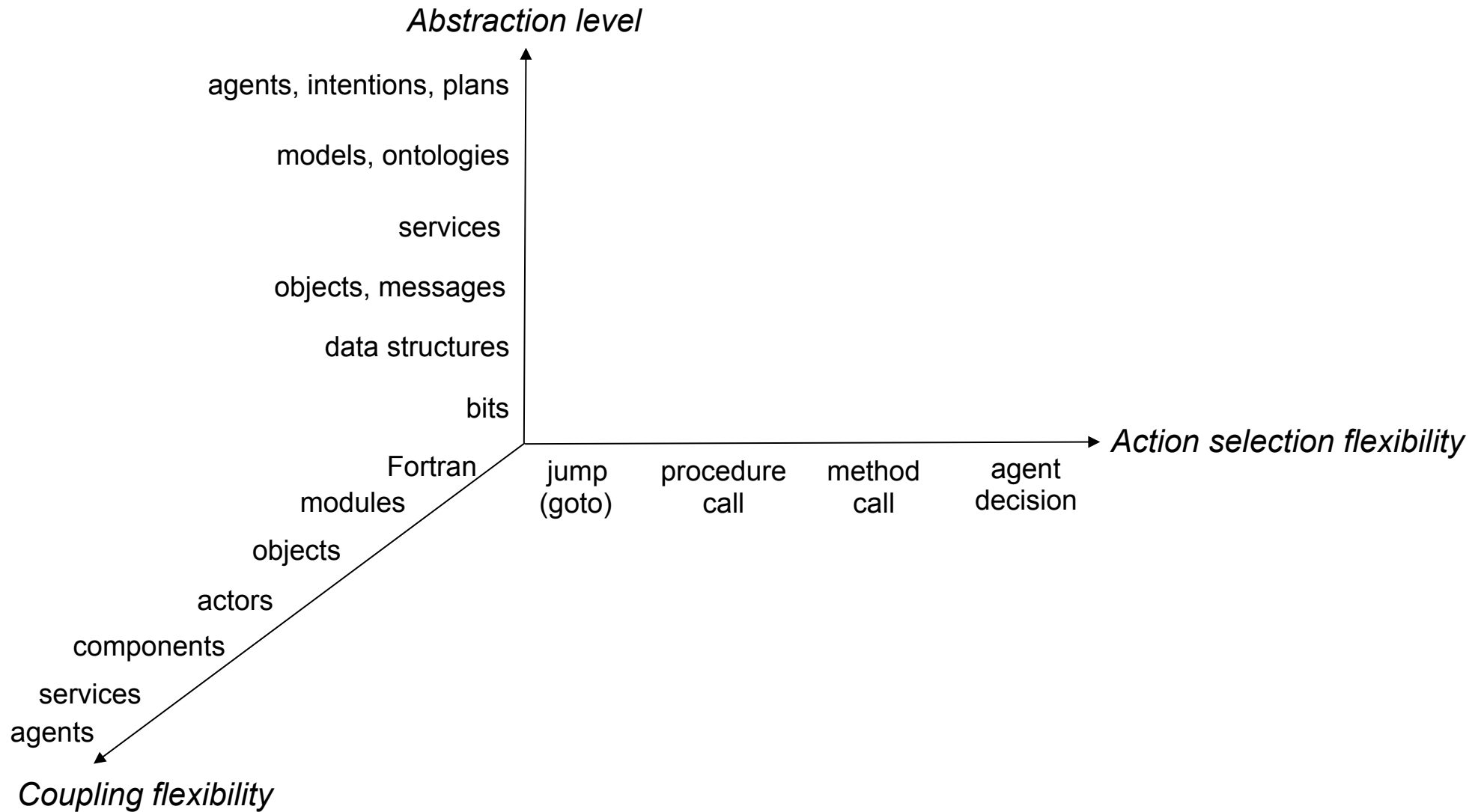
2nd Axis - Coupling flexibility

<i>coupling</i>	<i>objects</i>	<i>actors</i>	<i>components</i>	<i>services</i>	<i>agents</i>
<u><i>structure</i></u>	implicit & internal (<i>references</i>)	implicit & internal (<i>references</i>)	explicit & external (<i>connectors</i>)	indirect & volatil (<i>invocations</i>)	indirect & external (<i>roles</i>)
<u><i>communication</i></u>					
<i>receiver designation</i>	point-to-point & explicit	point-to-point & explicit	multipoint & explicit or implicit (<i>publish-subscribe</i>)	multipoint & dynamic (<i>discovery & selection</i>)	multipoint & explicit or implicit (<i>indexed by roles</i>)
<i>data transfer</i>	bidirectional (value return) & direct	unidirectional & direct	bi- or uni-directional (<i>events</i>) & direct	bi- or uni-directional (<i>events</i>) & direct	unidirectional & direct or indirect (<i>environment</i>)
<i>synchronization</i>	synchronous	asynchronous	synchronous or asynchronous	synchronous or asynchronous	synchronous & protocol

Flexible and Autonomous Coupling

- Traditional Programming
 - Coupling between software components is implicit and buried within the components
 - Programmer cannot change the coupling without reimplementing internals
- Component Programming
 - Coupling between software components is explicit and outside components
 - Programmer can change the coupling without reimplementing internals
 - Better reuse and flexibility
 - Explicit vision of system programming (Programming at large)
 - Evolving coupling is done at configuration time
- Service Programming
 - Coupling between components is managed by components themselves
 - They can decide dynamically who they will contract
 - » Discovery and Selection of service
- Multi-Agent Programming
 - Arbitrarily complex coupling protocols (Coordination)

Evolution of programming



3rd Axis - Abstraction level

- From machine-oriented to application-oriented high-level abstractions
- More explicitness (type of request, architecture, intention, plan...)
- More knowledge
- 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

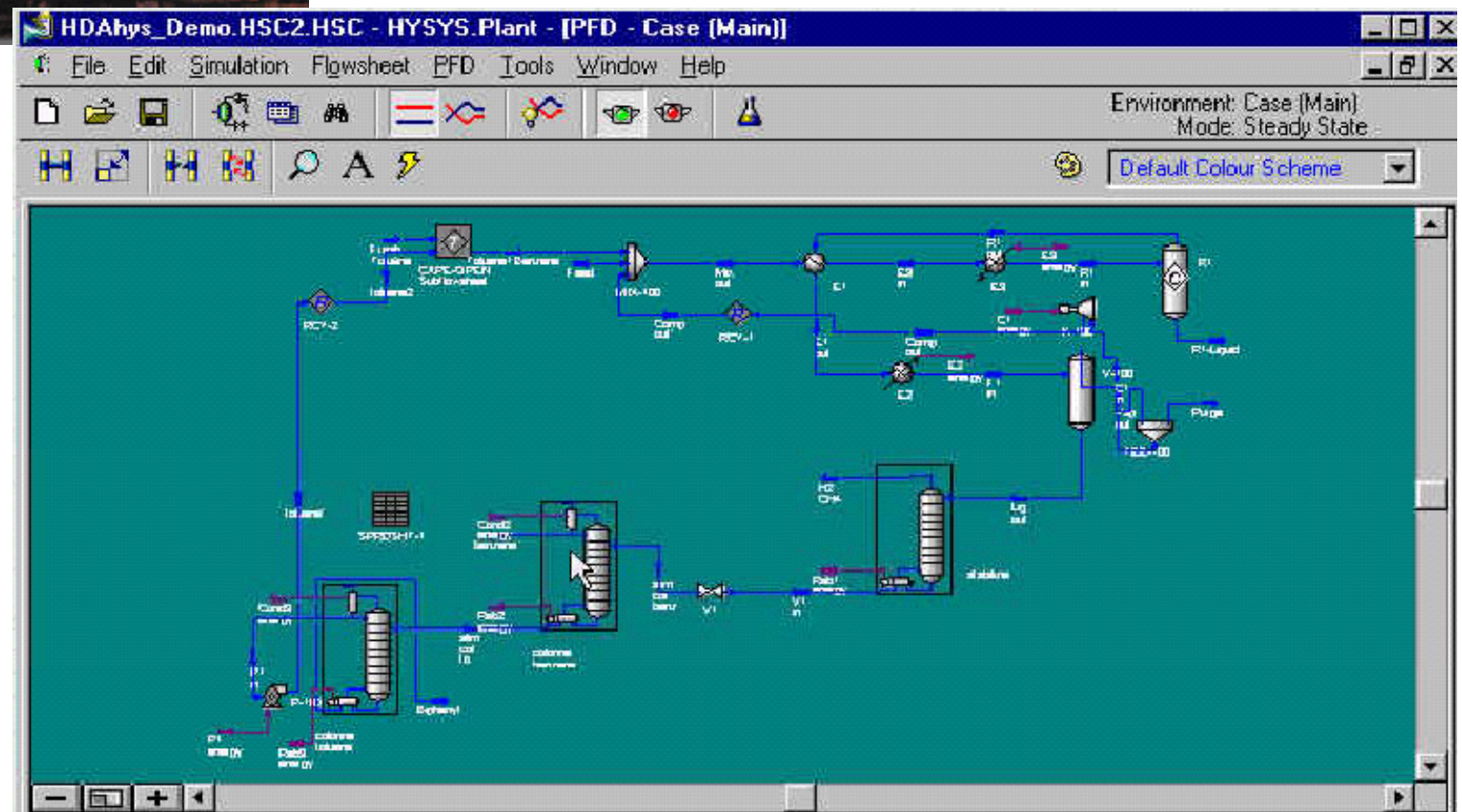
What agents could bring to components ?

- More flexibility for assembling (match-making)
 - Signatures (types) strict but low level
 - Components that plug but just don't play [Northrop 1991]
 - Higher-level descriptions
 - » contracts, concepts (ontologies)...
 - cf. semantic descriptions/coupling vs syntactic data-level descriptions/coupling
- Mechanisms (reorganization) for dynamic reconfiguration (and reassemblage)
 - Macro-level
 - Micro-level
- *More “intelligent” behavior (intelligent/adaptive cooperative components)*

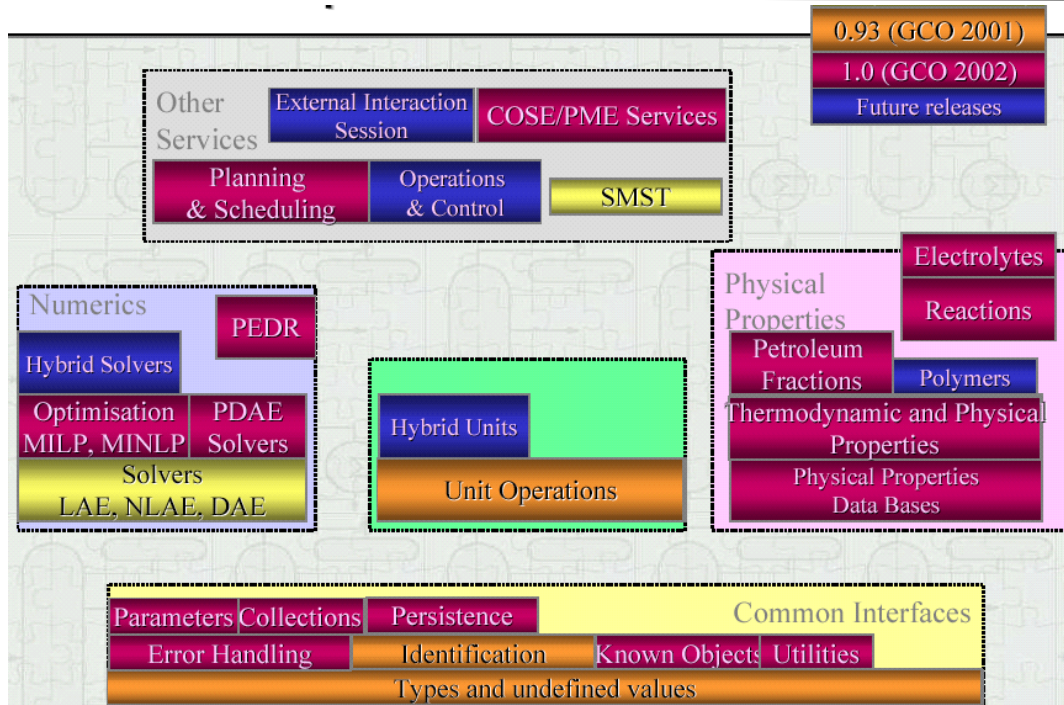
Example: components match-making



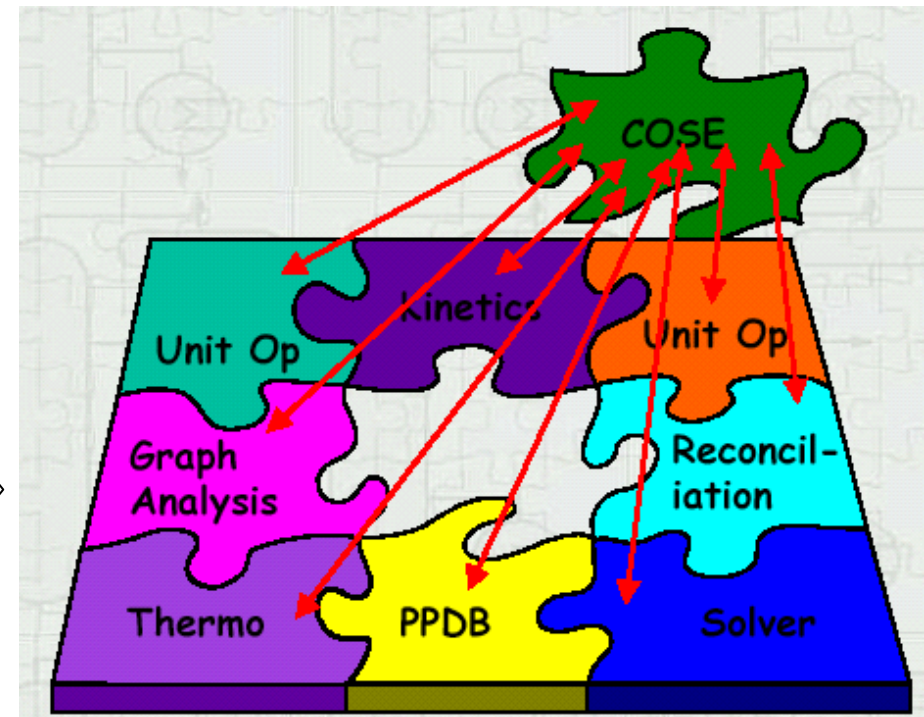
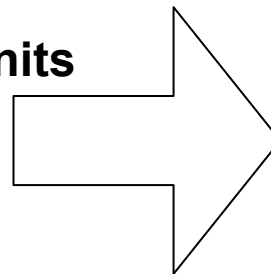
Petrochemical process engineering (design, simulation, *control*)



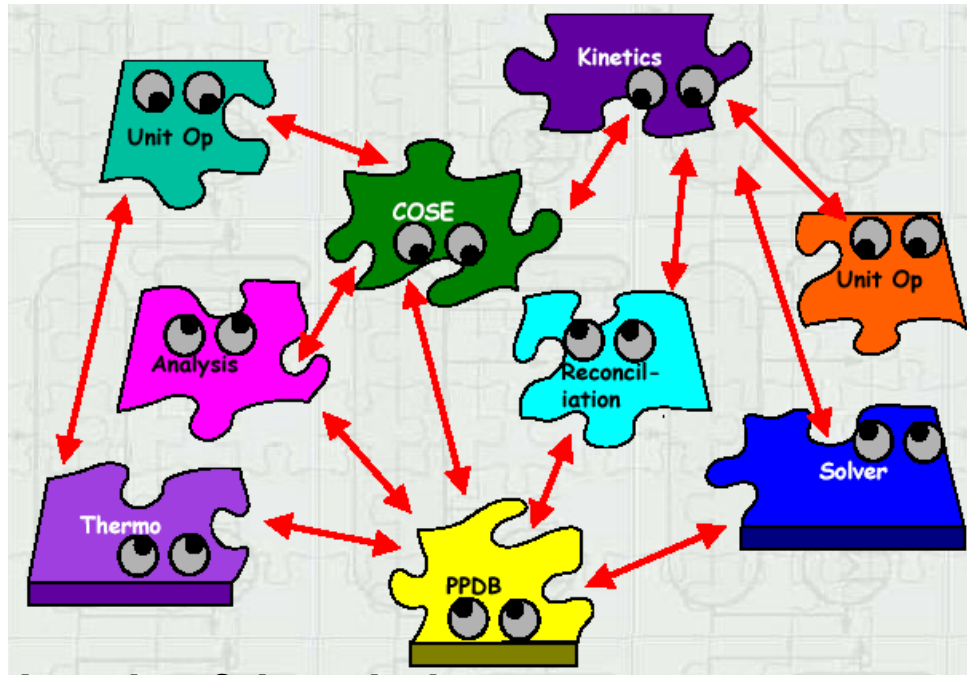
Initial step: Interoperability: CAPE-OPEN Project [Braunschweig et al. 02]



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

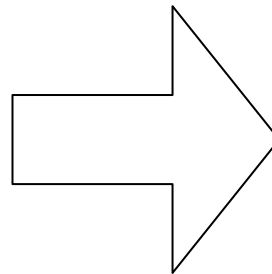


Second step: Assistance to Assemblage: COGENTS Project [Braunschweig et al. 04]

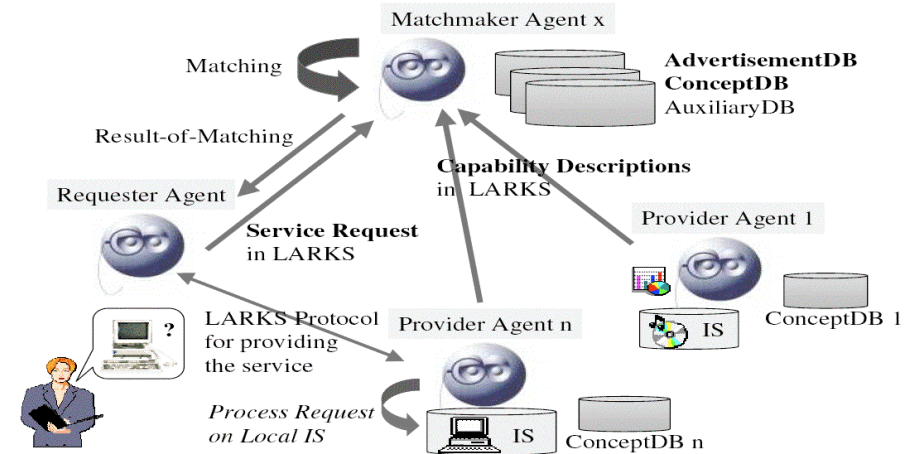


- Levels of description
(signatures, assertions, ontologies: Onto-CAPE)
- Match-making
- Assistance for assemblage
- Instantiation
(actual Software products)

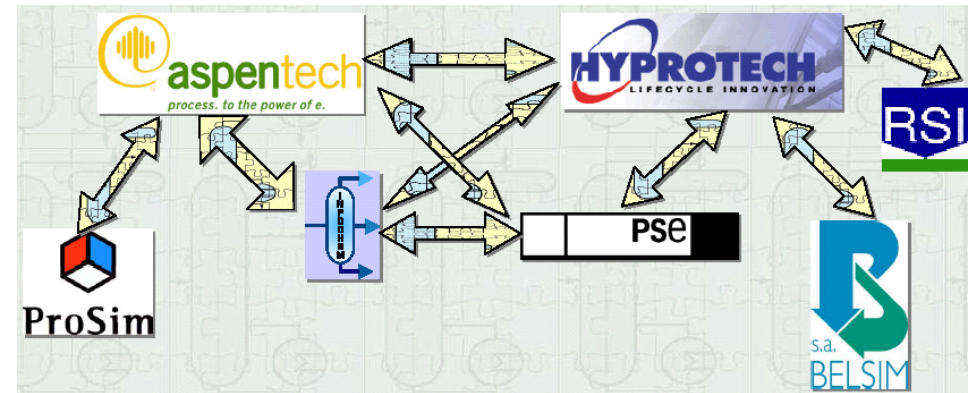
Jean-Pierre Briot



Agents and Components

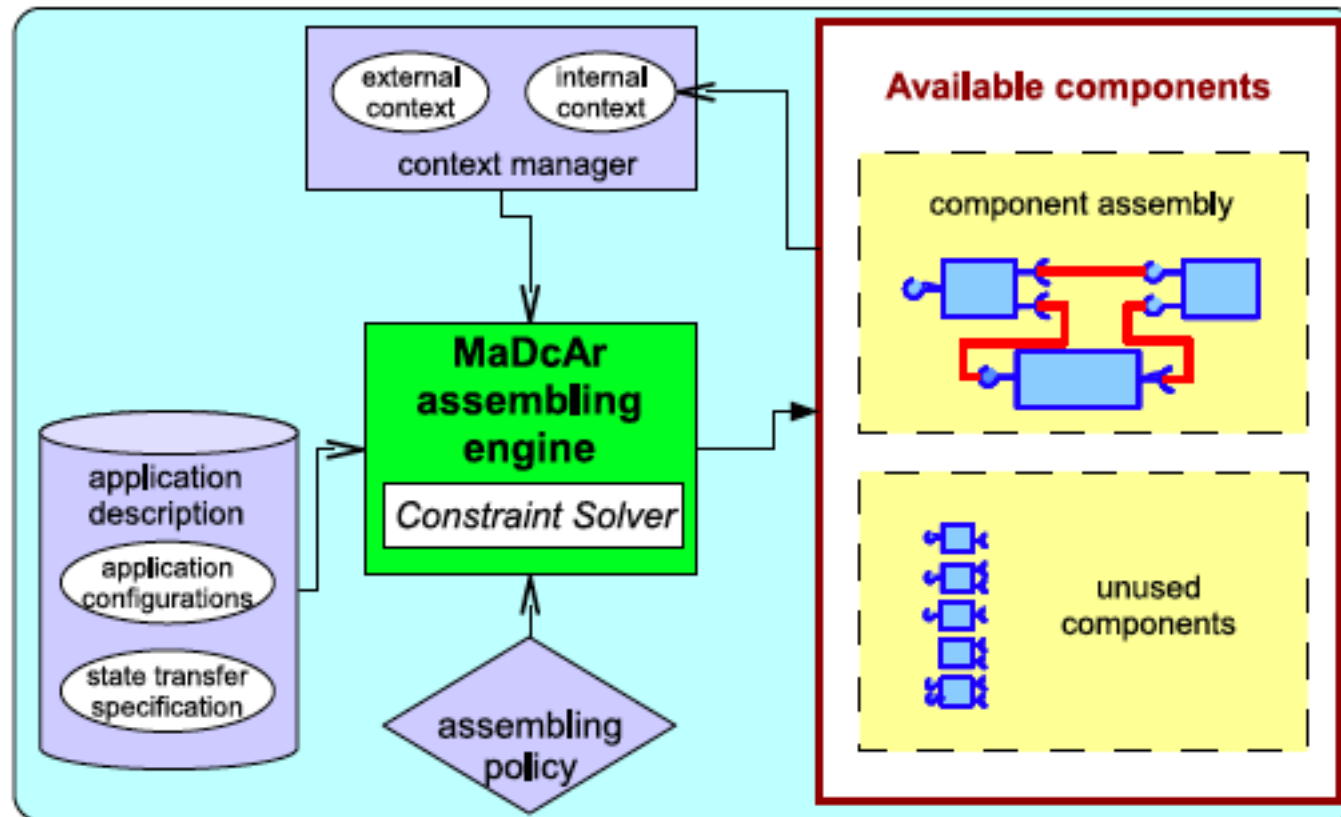


E.g., LARKS matchmaking [Sycara et al. 98]



Other Ex: Dynamic Reassembly of Components

- Ex : MadCar [Grondin et al. 2004]
 - Decentralized control
 - High-level general reassembly model/mechanism
 - Based on concepts of configuration, roles and policies



—> data flow

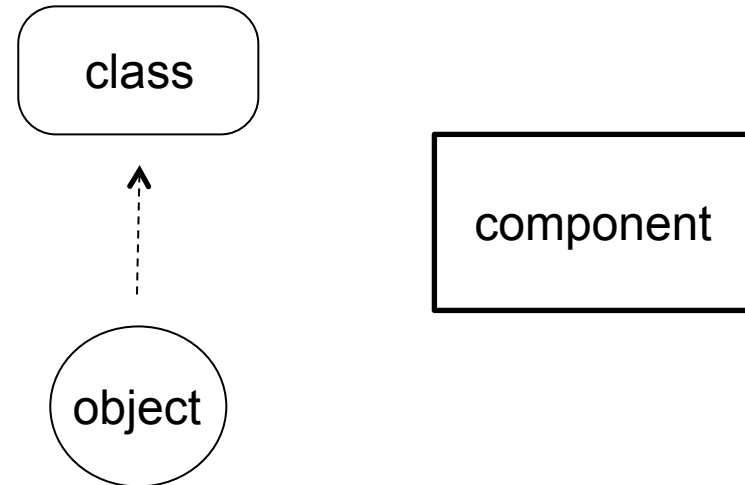
—> modification
Agents and Components

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

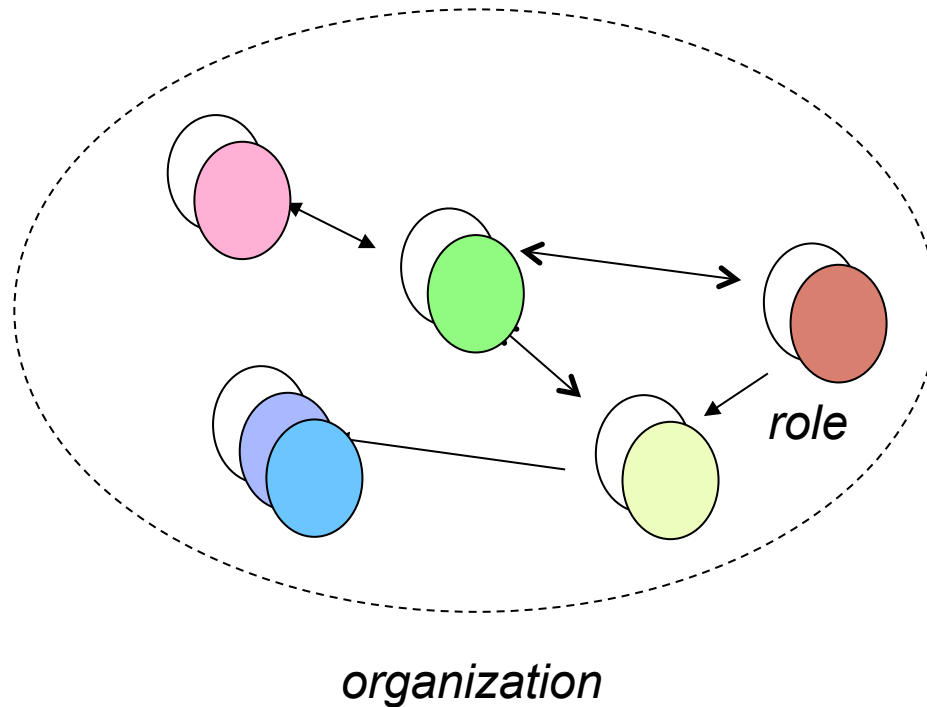
"Self-containedness"

- Includes all the code
- "Ready to use"
- “Ready to deploy” [Melo et al. 2004]
- Includes documentation
- Assistance to deployment (e.g., OLAN/AAA [Belissard et al. 2006])

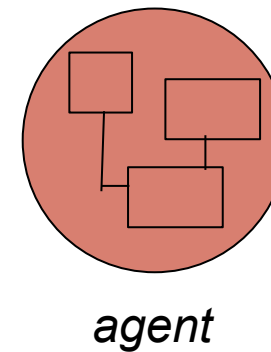


Architectural support

- At the macro / system / organizational level



- At the micro / (single) agent level

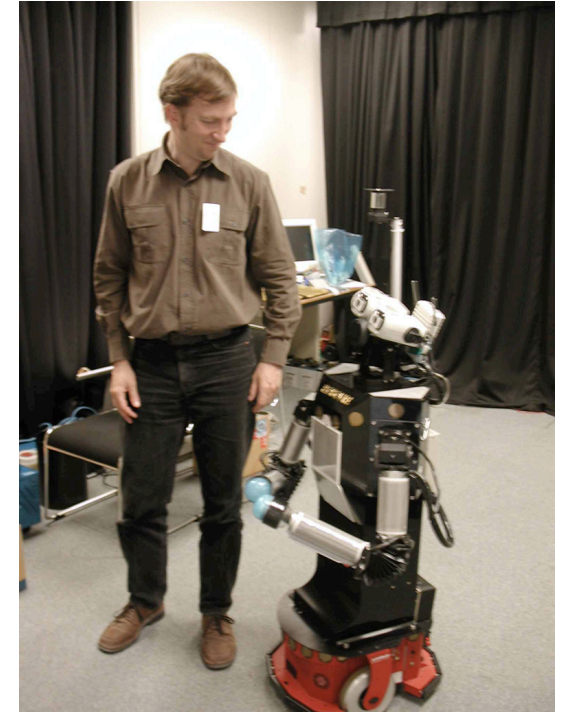
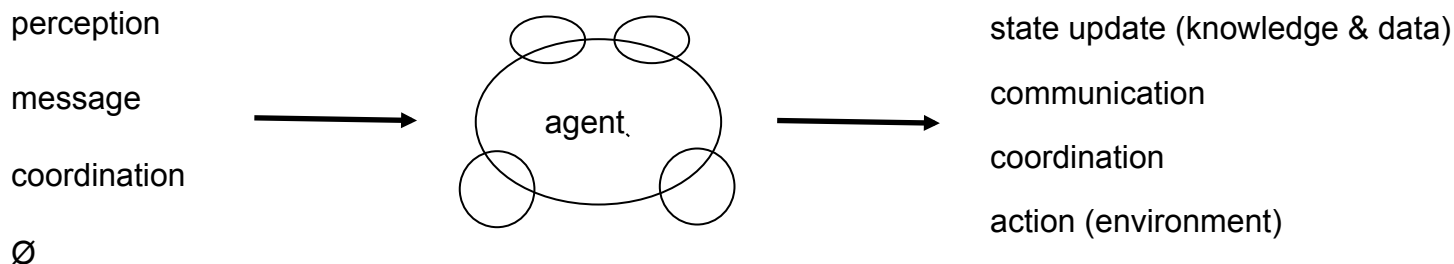


Outline

- 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

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)



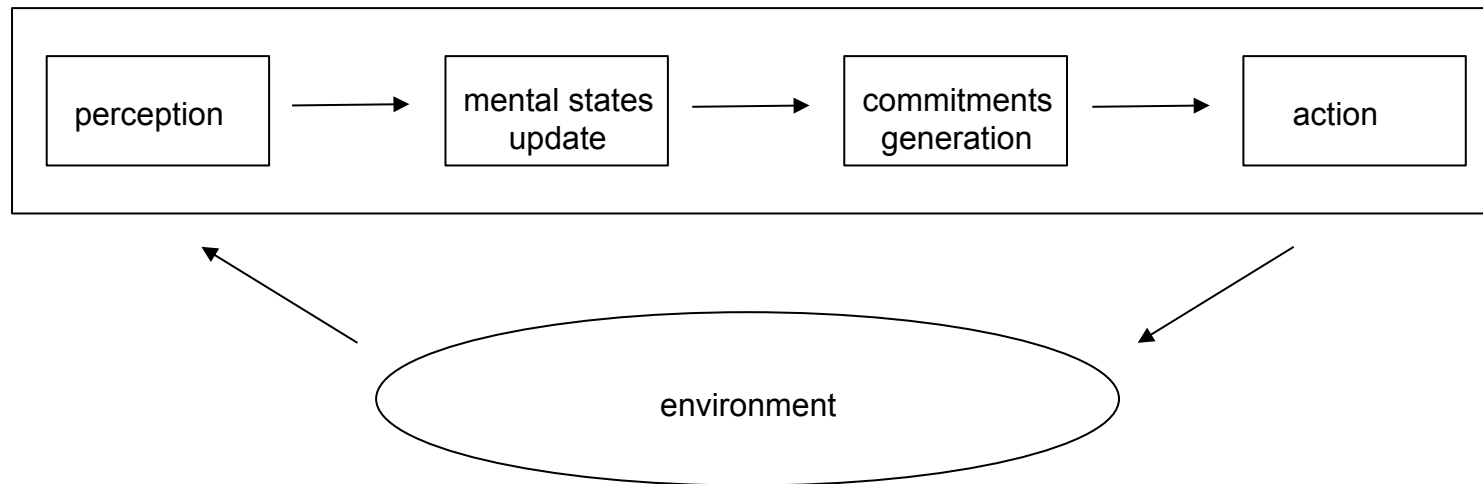
- **Architecture of an agent:**
 - the software structure in charge of that action selection
 - functions of the agent and their interactions

Rationale (tentative typology) for (one) agent architectural decomposition

- *Analog to software achitectural 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

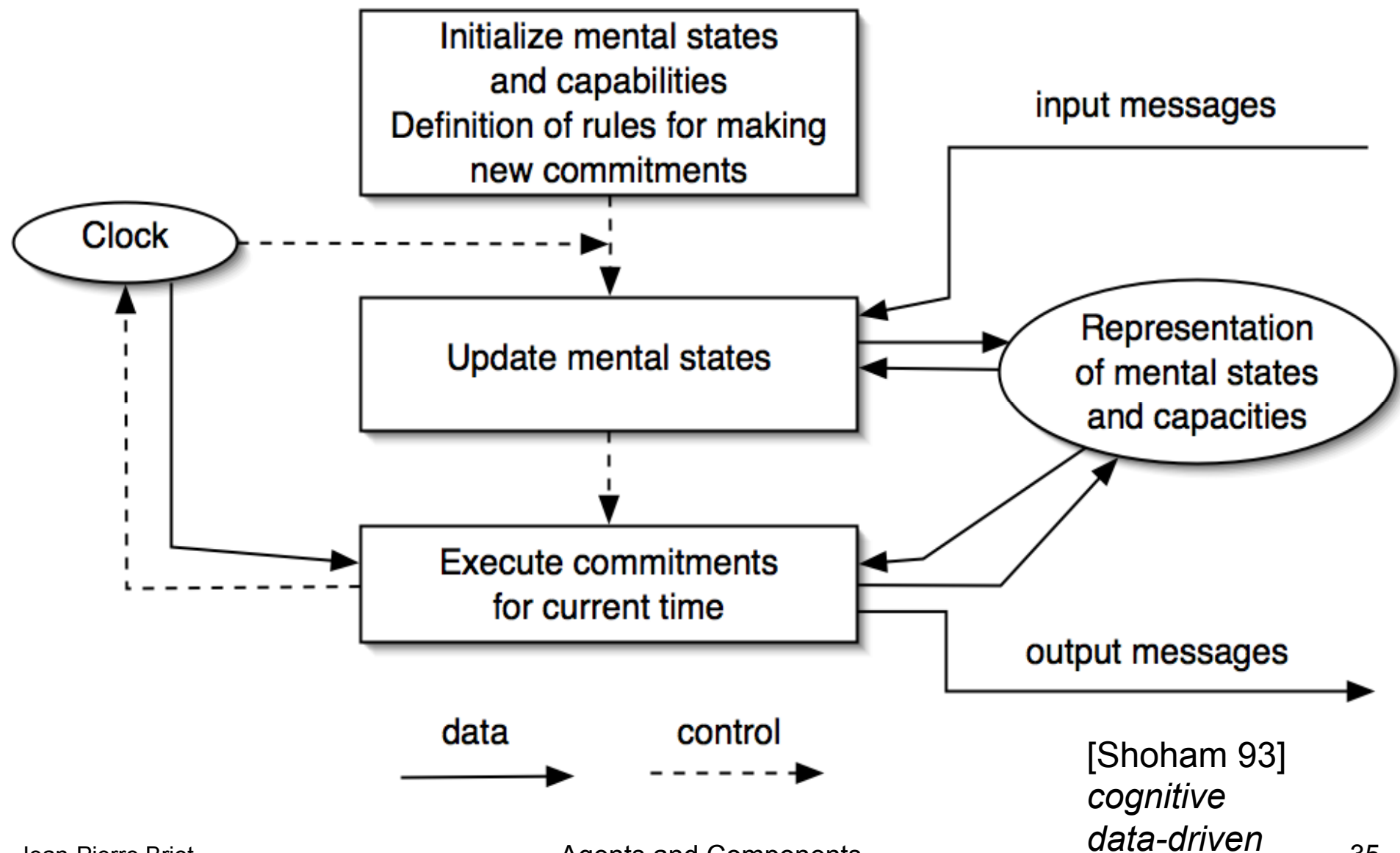
Cycle decomposition: “Horizontal” modular architectures

- one layer
- decision/action cycle



*reactive
data-driven*

Ex. of Cycle-based Decomposition: AOP (Agent Oriented Programming) Architecture

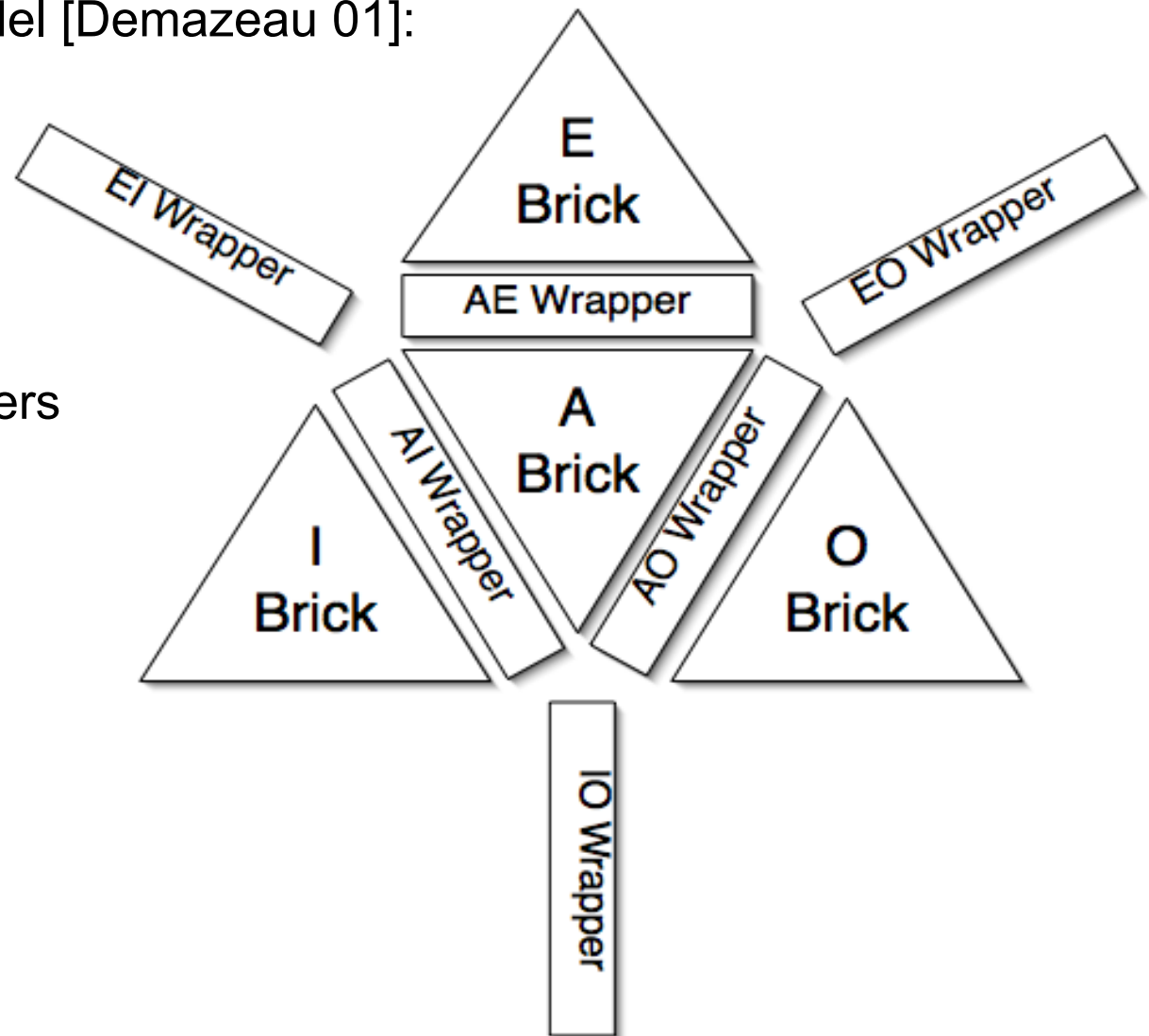


Ex. of Viewpoint decomposition: Vulcano [Ricordel 02]

- Vowels decomposition model [Demazeau 01]:

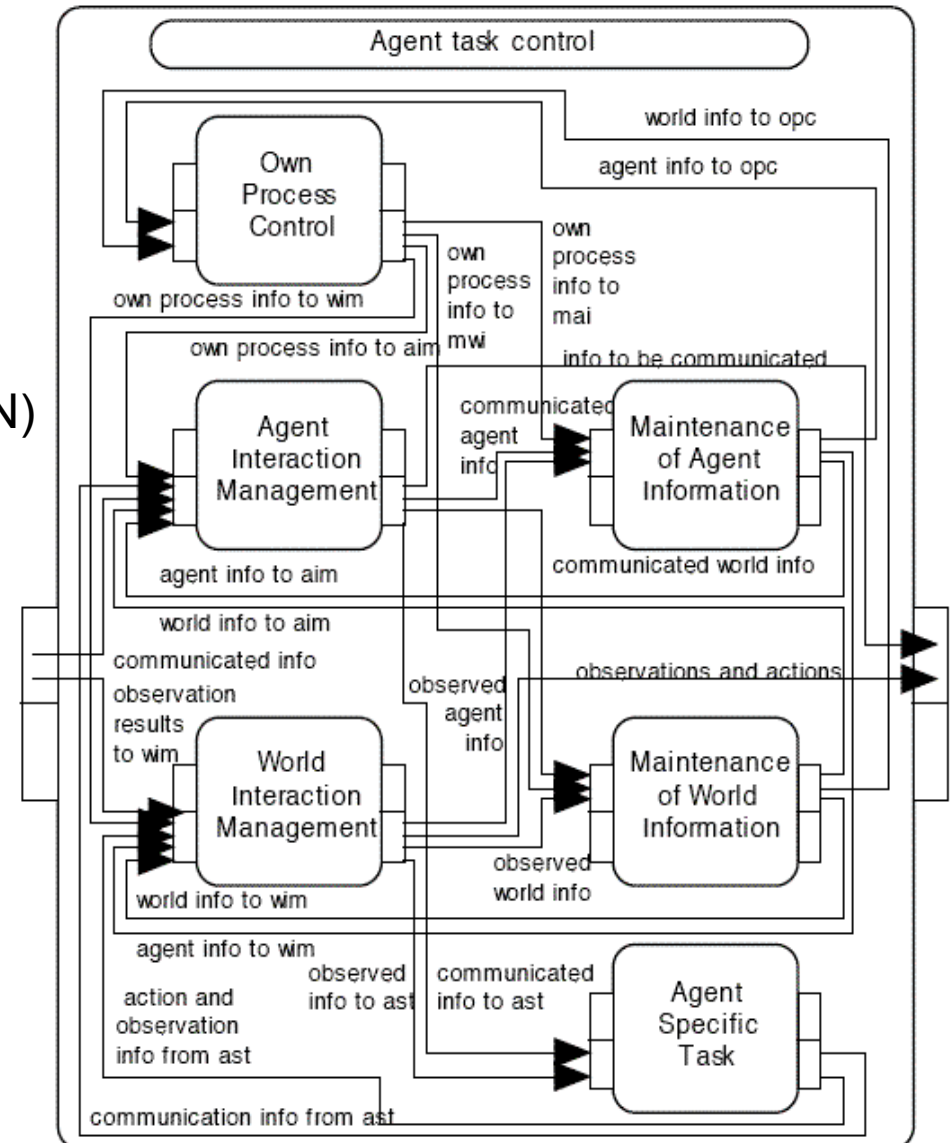
- A(gent)
- E(nvironnement)
- I(nteraction)
- O(rganization)

- Interfacing wrappers/adapters
(*ad-hoc*)



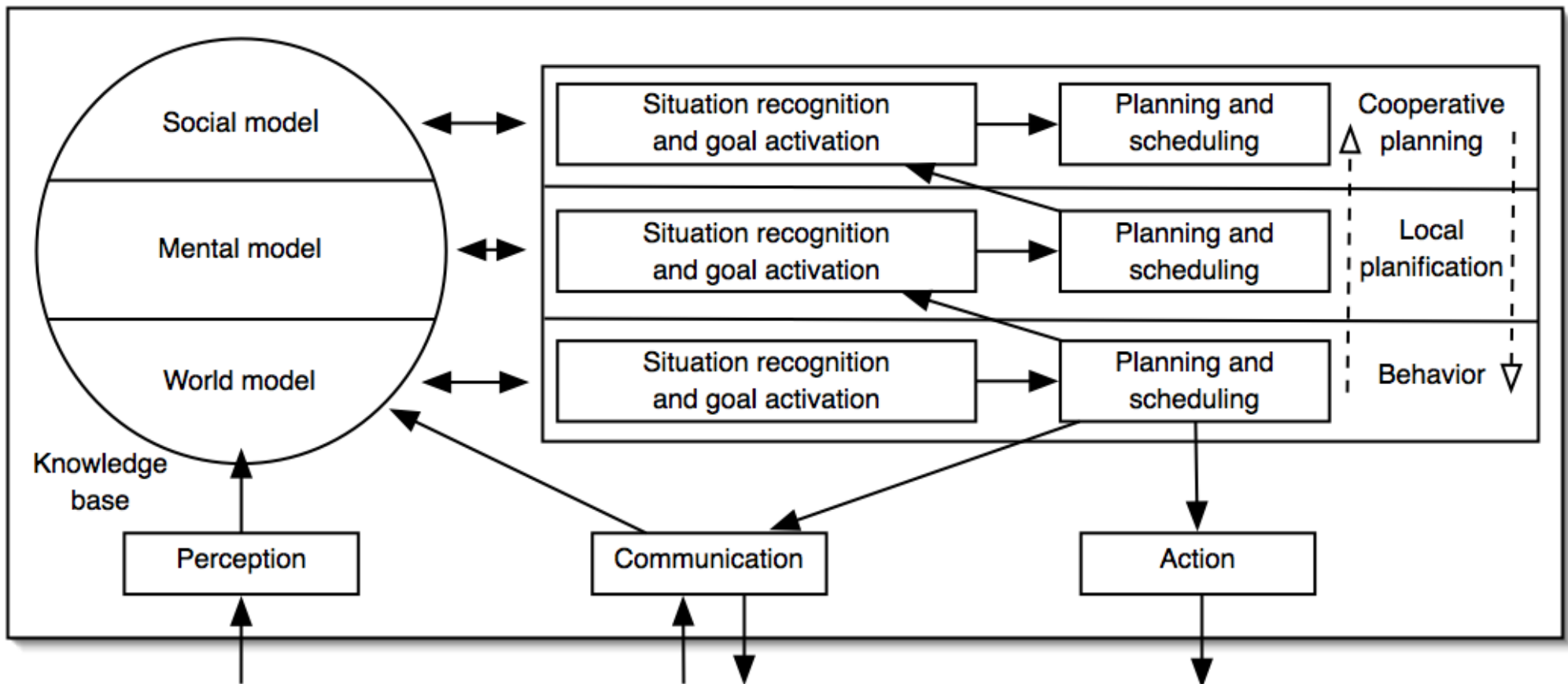
DESIRE [Brazier et al. 95-01]

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



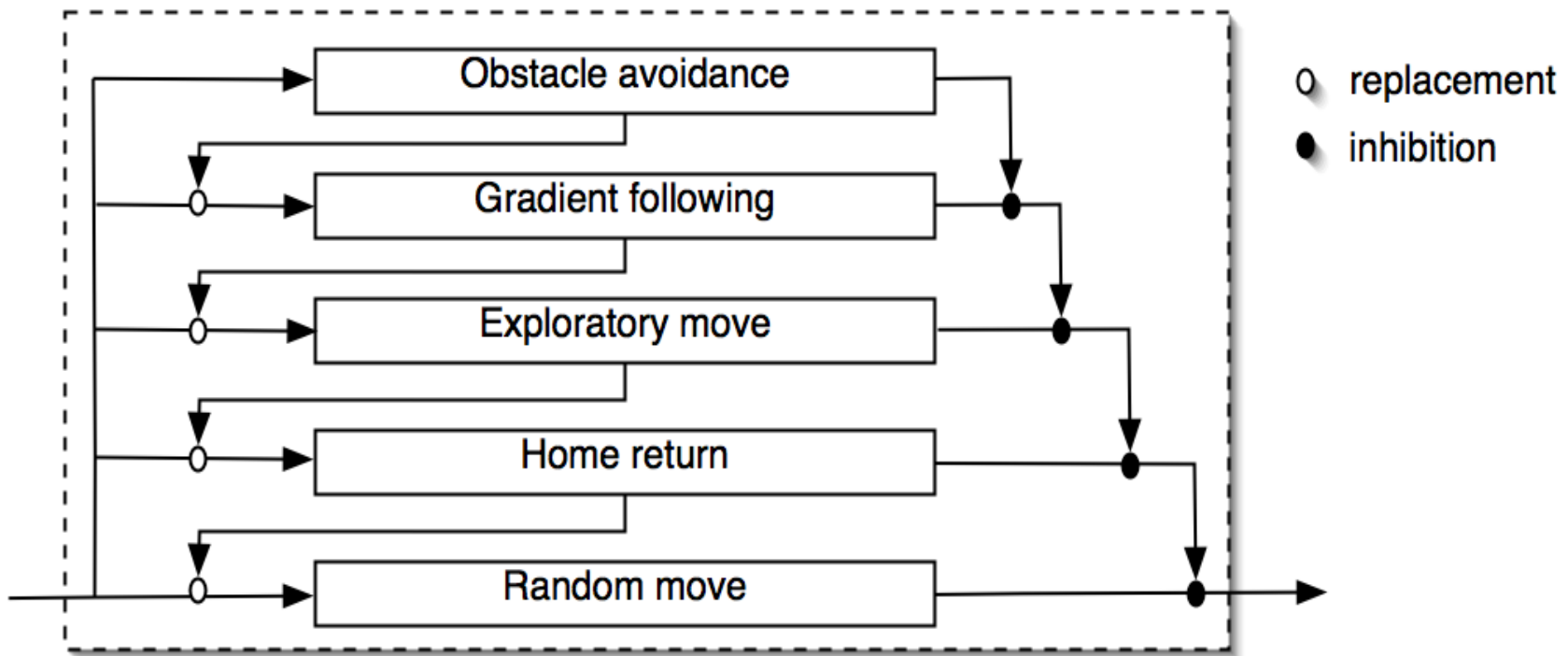
Ex. of Level decomposition: InteRRap [Müller 94]

- 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

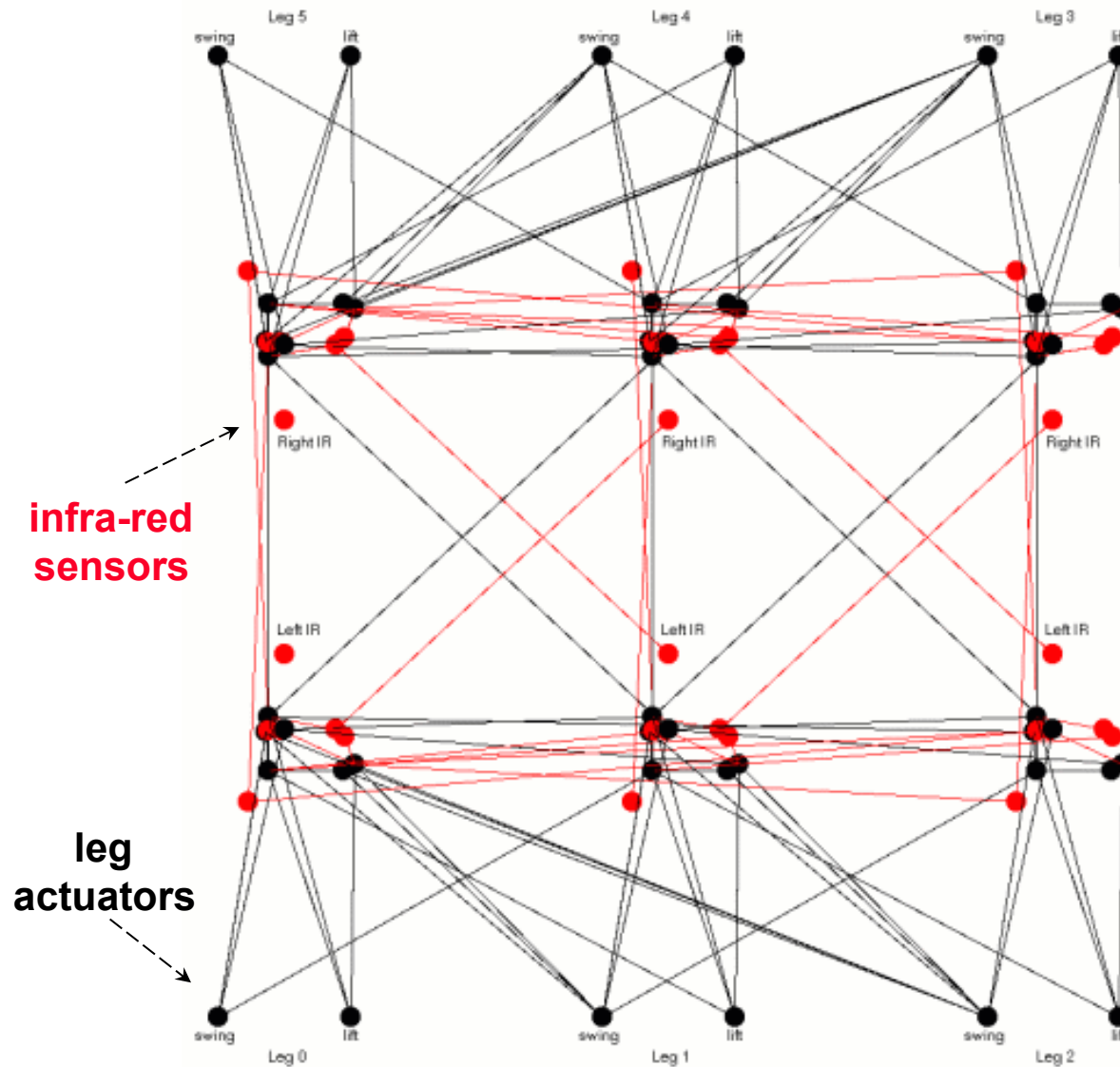


Ex. of Behavior decomposition: Subsumption architecture [Brooks 86]

- components activated in parallel
- competitive *and* hierarchical
- priorities and inhibitions:
 - replacing input of lower component ○
 - inhibiting output of lower component ●
- hard-wired



Other: Evolvable architectures [Meyer et al. 98]



Genetic programming

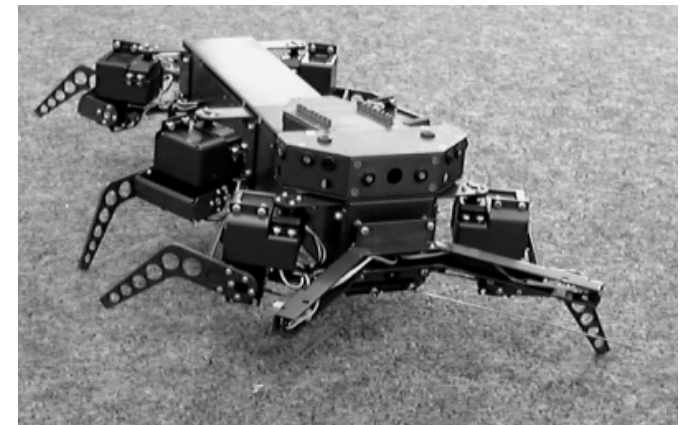
Evolution of the development program

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:

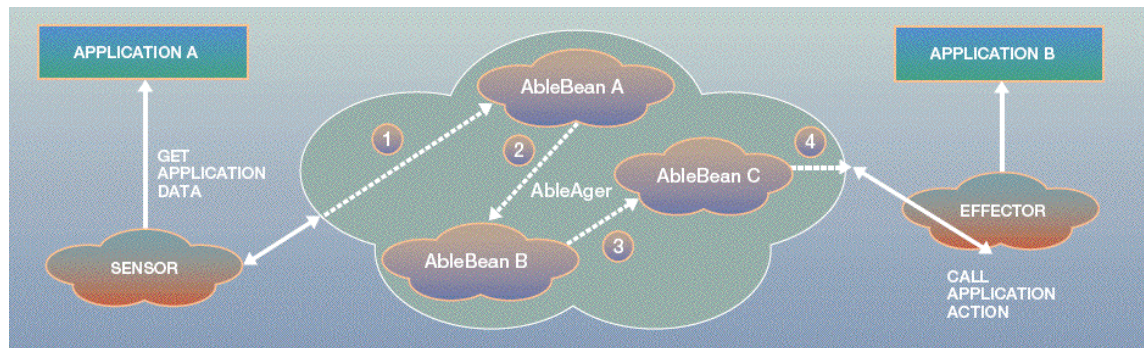
- No more architecture
- Just a component model (like ex: JavaBeans)

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

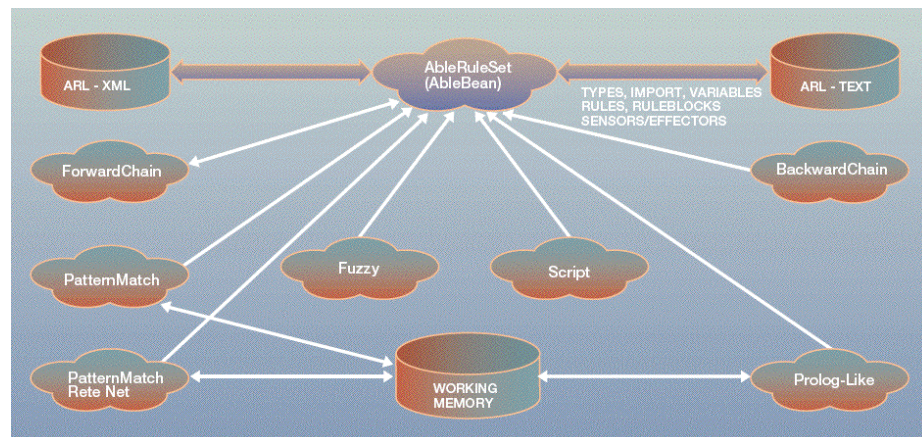
Tools (tool box) decomposition, ex: ABLE architecture [Bigus et al. 02]

IBM Autonomic computing programme

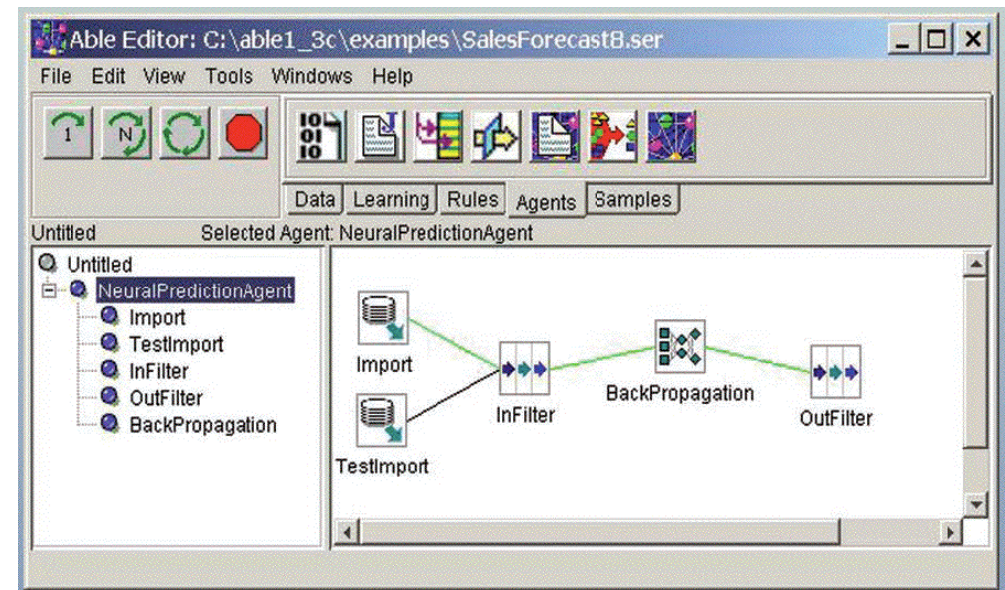


- Data beans
e.g., TimesSeriesFilter
- Learning beans
e.g., BackPropagation
- Rule beans
e.g., FuzzyForwardChaining
- Specific beans
e.g., GenericSearch

Java Beans-based implementation

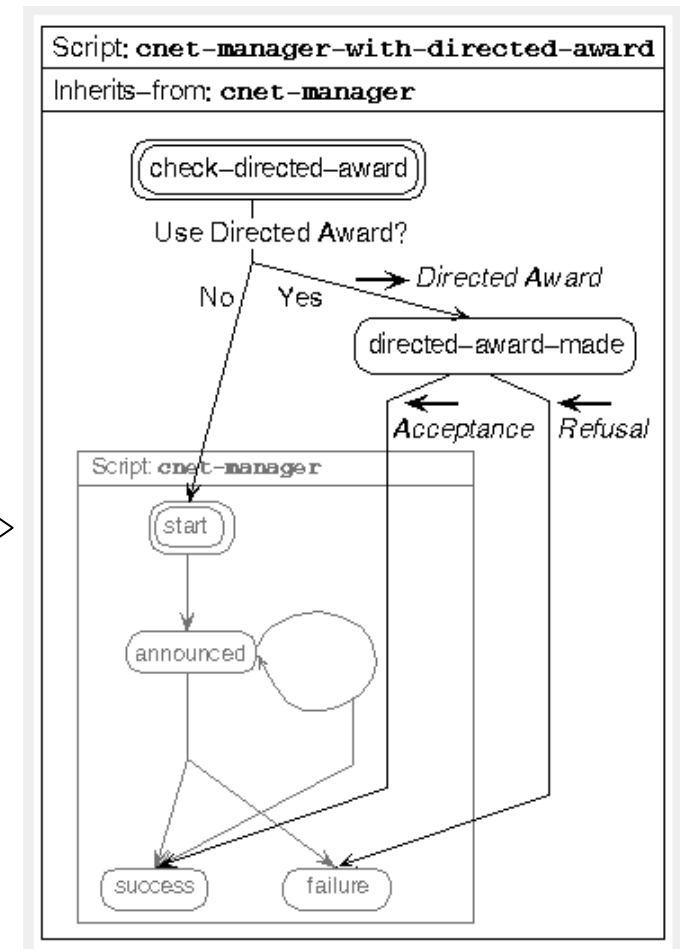
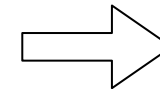
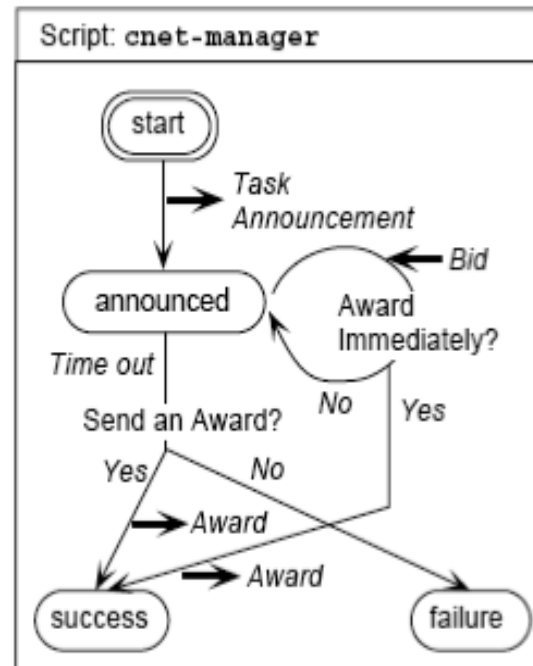


E.g., rule beans



Reusing protocol components

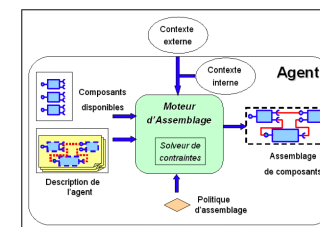
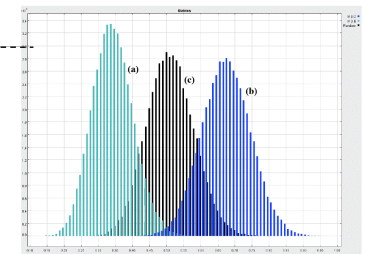
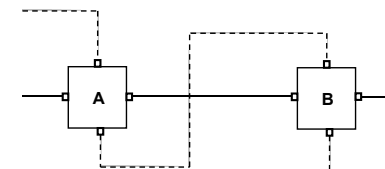
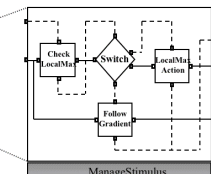
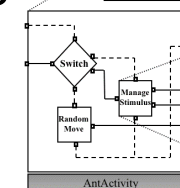
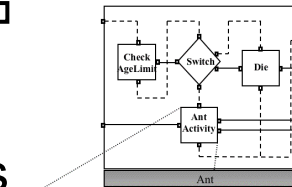
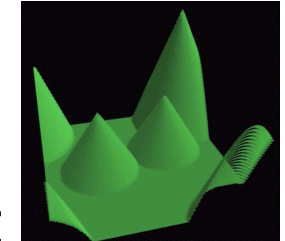
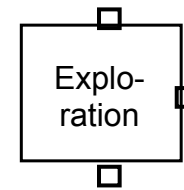
- Extensions of the Contract Net Protocol (CNP)
- Agentalk [Kuwabara et al. 95]
 - inheritance (e.g., directed-award-CNP)
 - customization interface



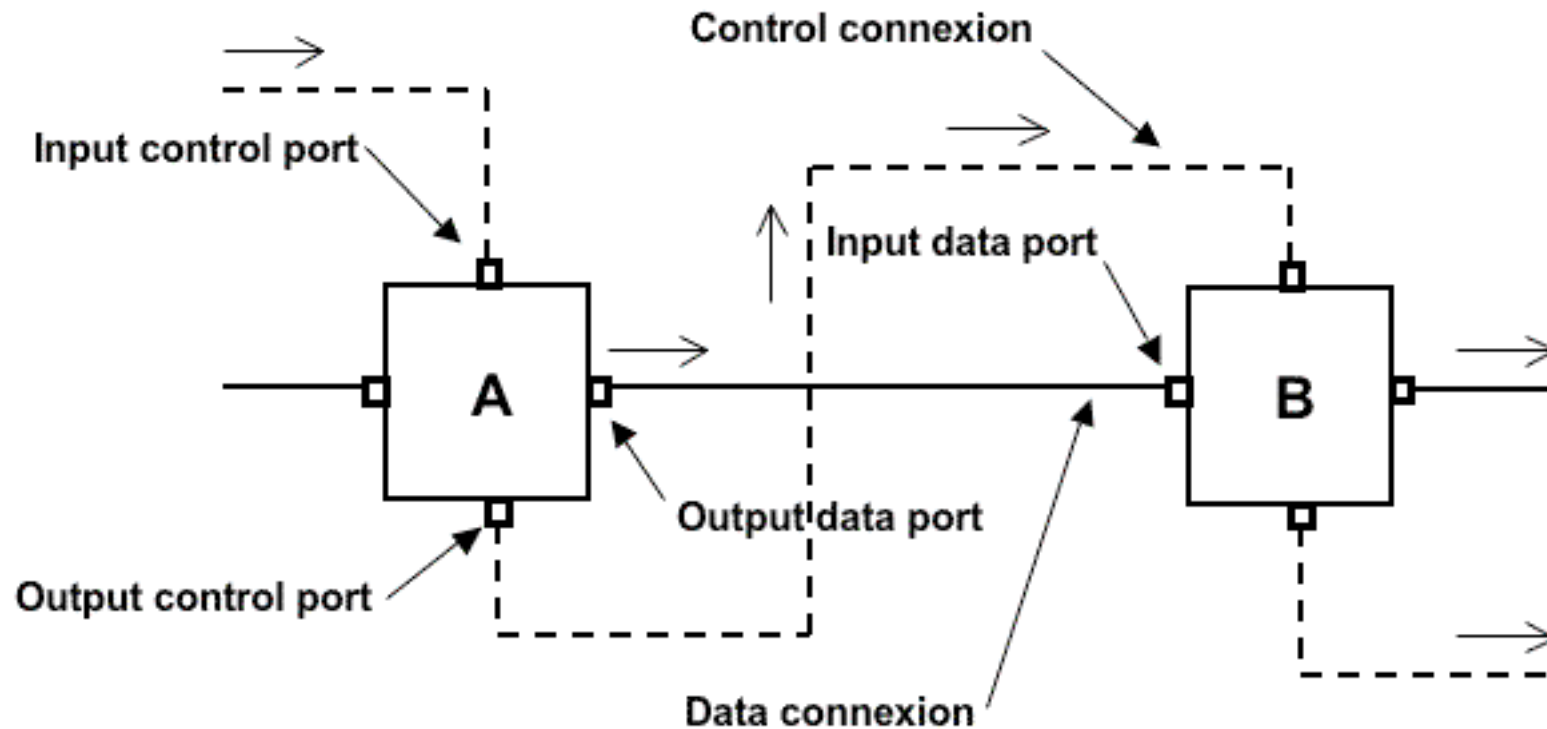
- SCD [Yoo et al. 98]
 - inheritance (e.g., time-out-CNP)
 - composition (e.g., iterated-CNP)

The MALEVA component model

- Domain: multi-agent-based simulation (*but in fact more general model*)
 - Ex: traffic simulation, eco-systems, population micro-simulation...
- Unit of decomposition: agent behavior
 - Ex: random move, gradient follow, reproduce...
- Assembling behaviors into more complex behaviors
 - Concept of composite component (behavior)
- "Componentification" of control flow:
 - Control ports, control connexions, control components
 - Distinction between control flow and data flow
- Helps at a fine-grained control of intra-agent scheduling
- Support of behavior dynamic change
 - Ex: from an egg, to a larva, to a worker ant

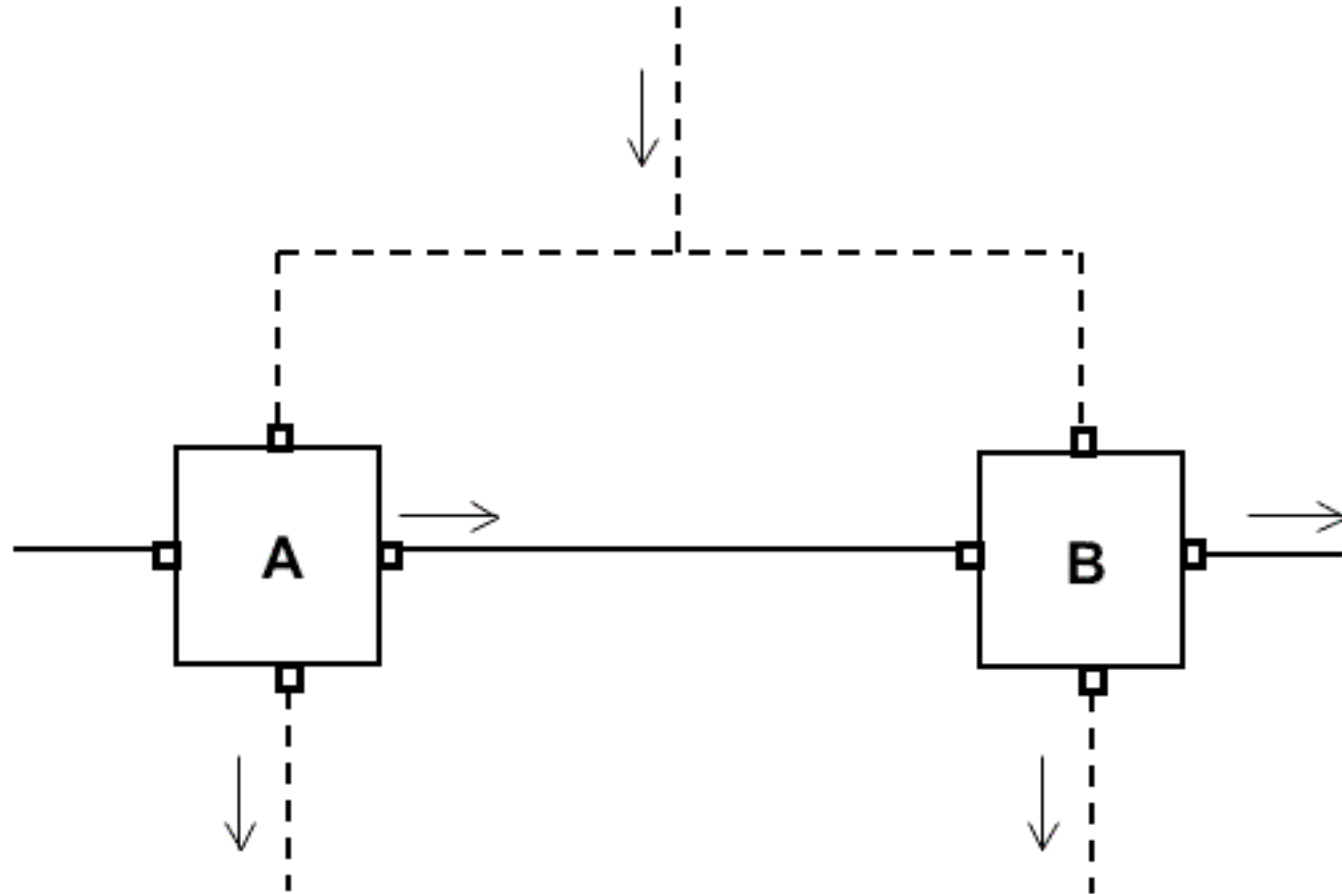


Data flow and control flow: ports and connexions



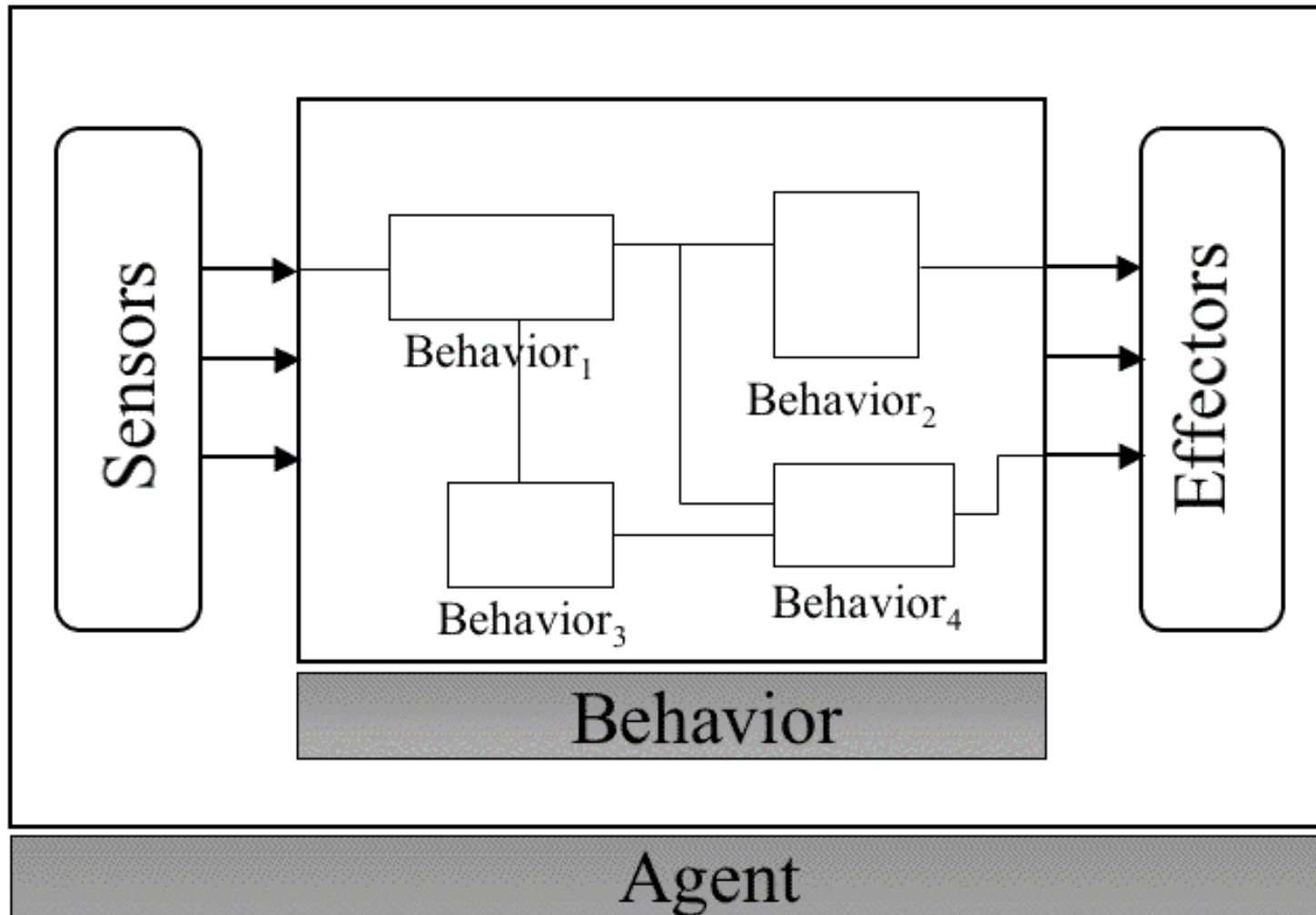
Sequential activation

Data flow and control flow: ports and connexions



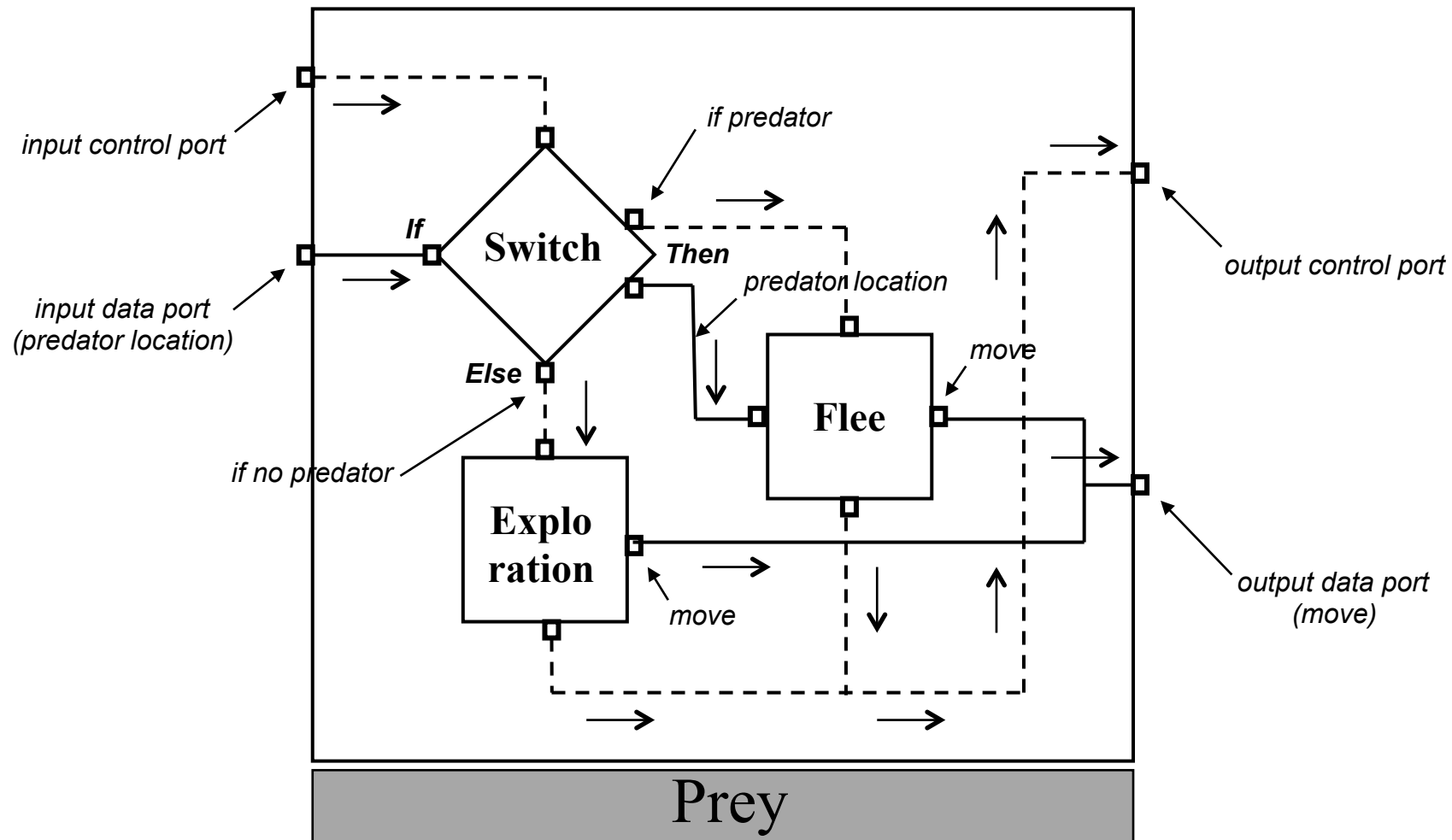
Concurrent activation

Abstract architecture of a situated agent



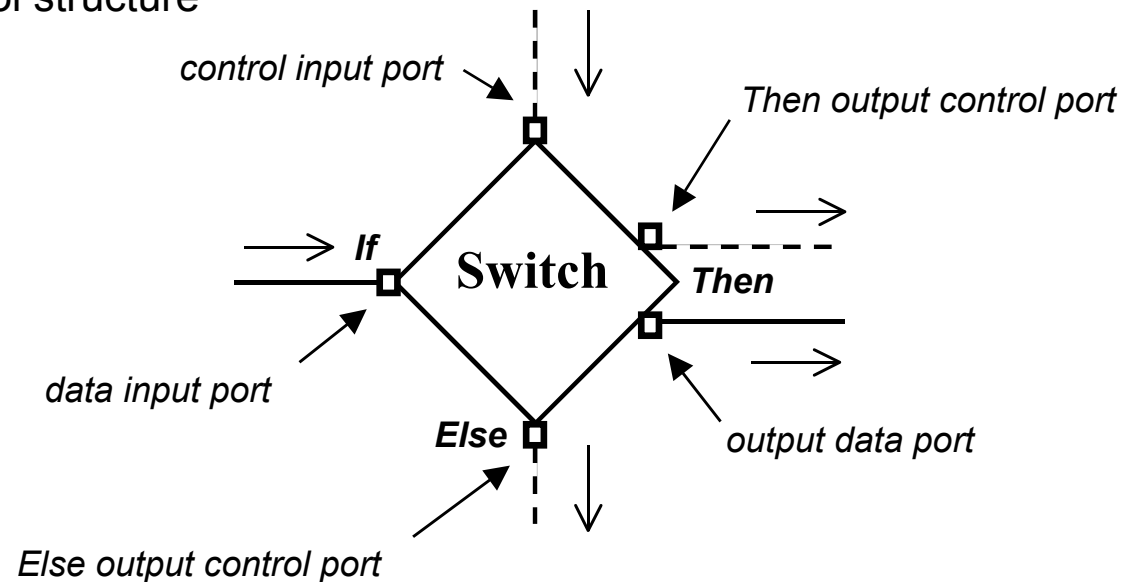
A first example: behavior of a Prey

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



Control components

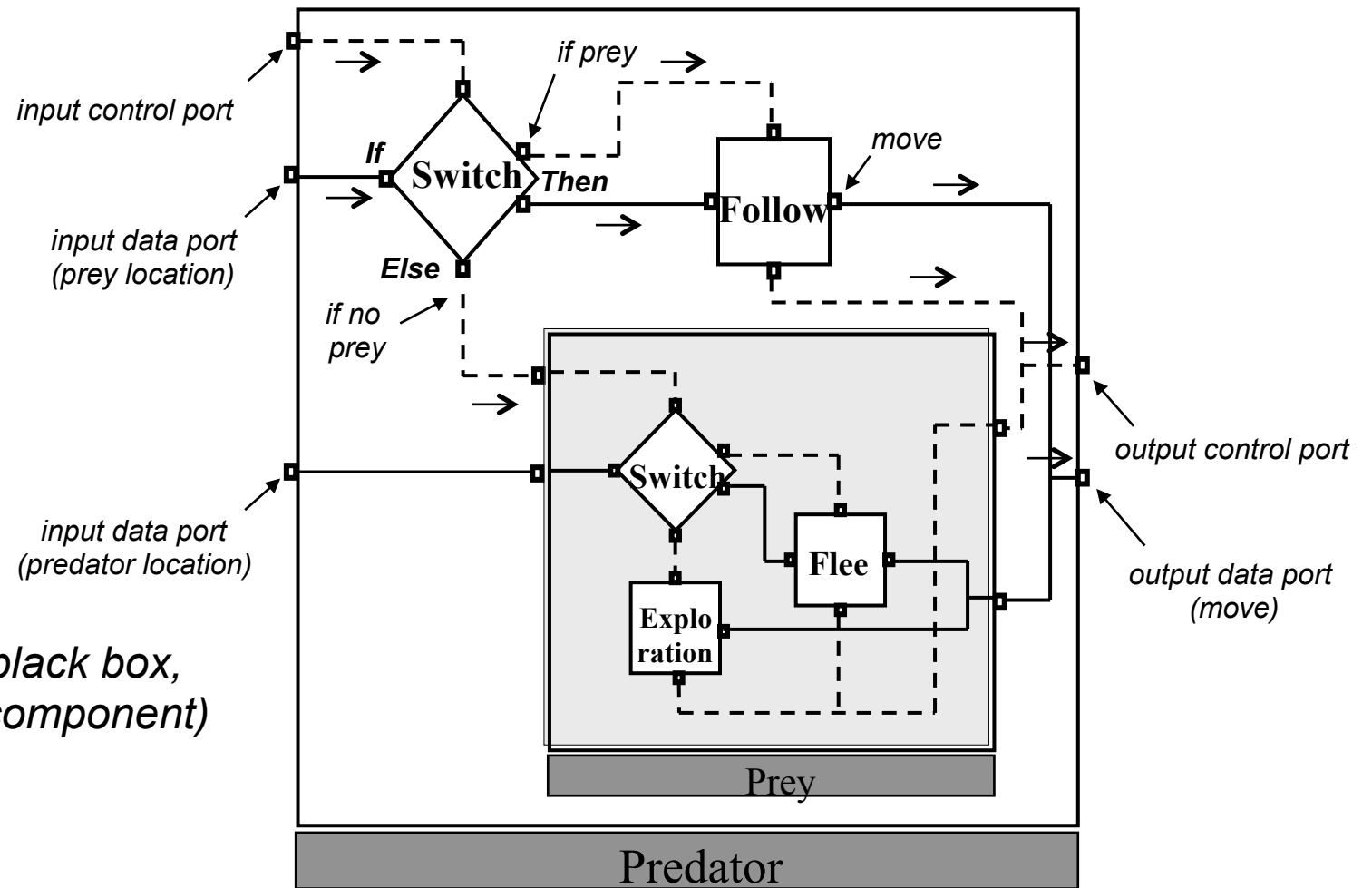
- Switch
 - reifies in a component
 - The traditional conditional control structure
 - *(if then else)*



- *other control components:*
 - control structures
 - » e.g., Repeat
 - synchronization
 - » e.g., Sync (*synchronization barrier*)

Reuse of a Prey: Predator

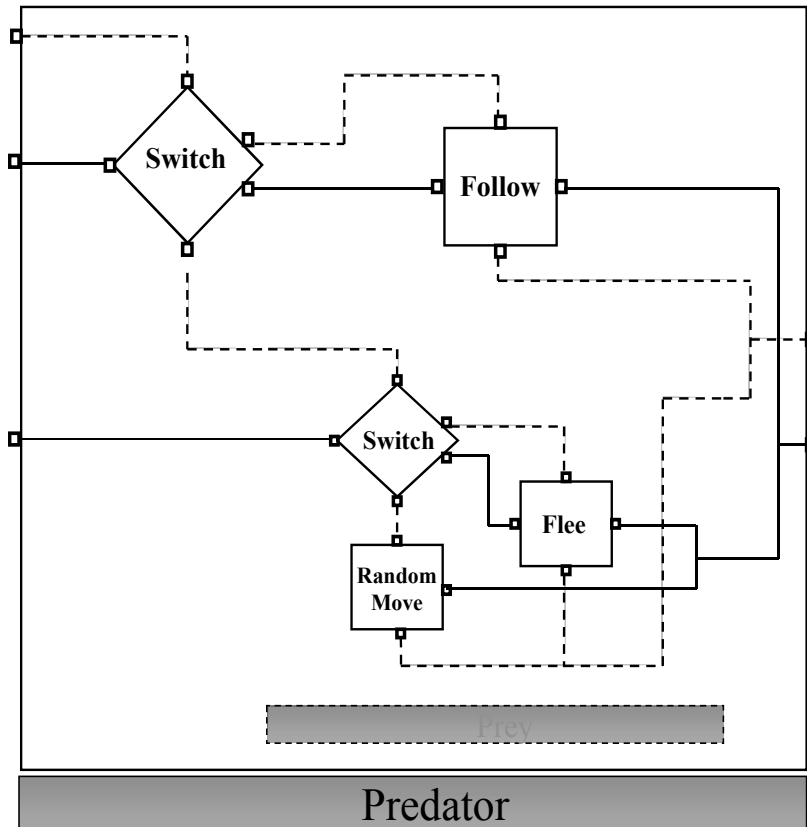
- if the Predator detects a prey, it follows the prey, otherwise, it acts as a Prey (*cannibalism among Predators*)



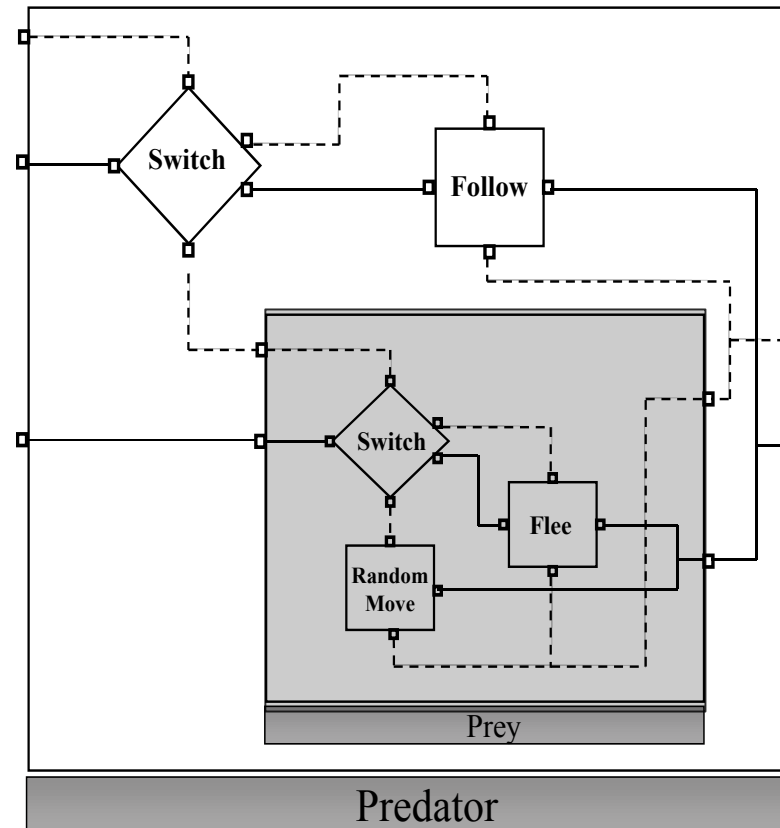
*Prey is reused as a black box,
(notion of composite component)*

Importance of composite component

- Two kinds of composition:



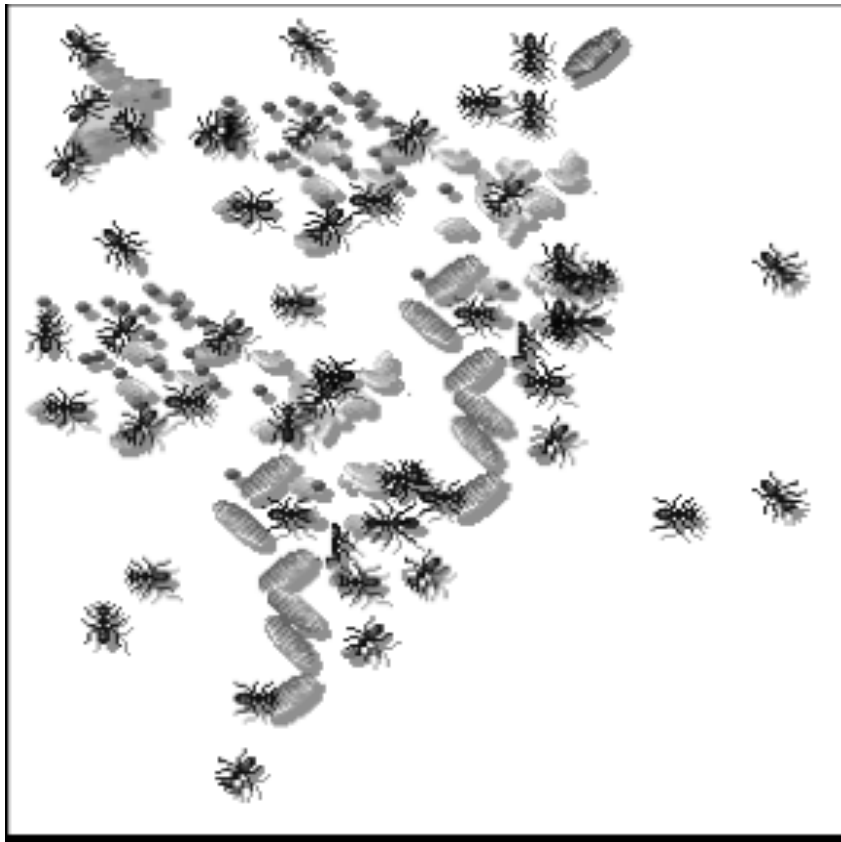
functional composition
(assemblage)



structural composition
(composite component,
information hiding, black box)

2nd Example: Ant Nests Simulation

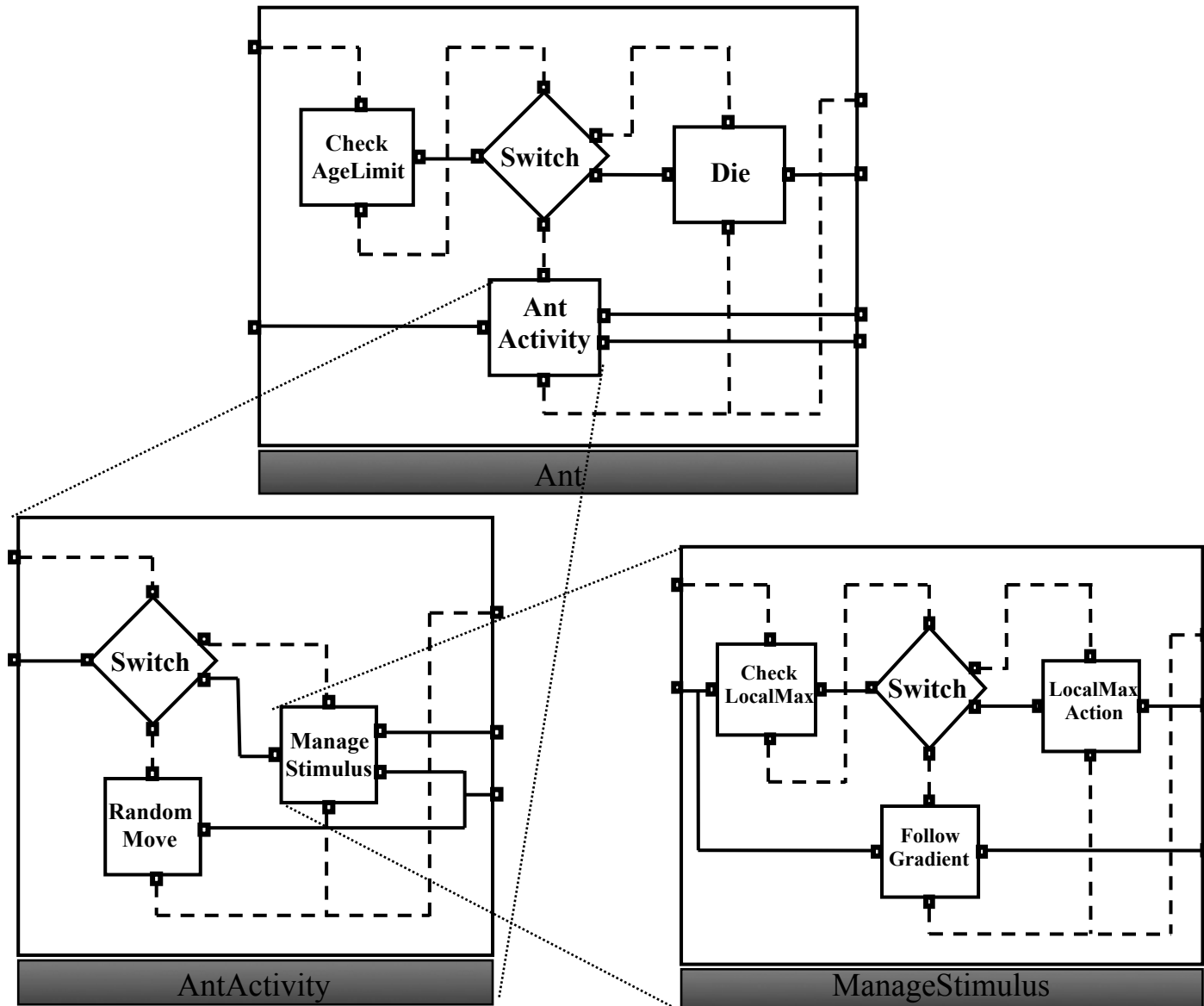
- Reengineering of MANTA simulation testbed [Drogoul 93]



ants (workers)
larvae
eggs
queen

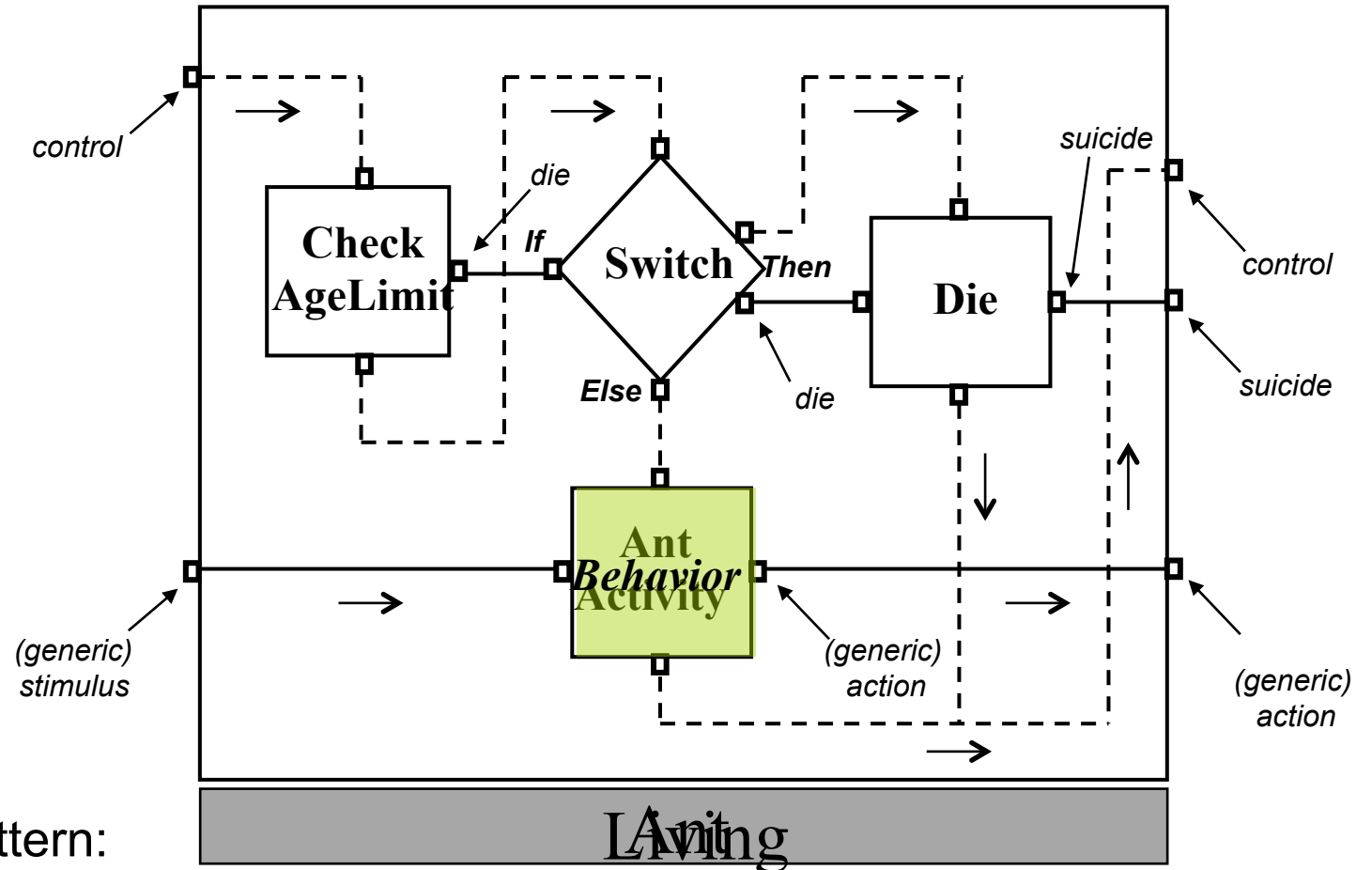
- Redesign of ant behaviors using MALEVA [Guillemet et al. 98]

Decomposition of the behavior of an ant



Reuse: Design Patterns [Guillemet et al. 99]

- E.g., “Living Agent” (ageing agent) pattern
 - used for egg,
 - larva,
 - ant worker,
 - queen...
-
- The diagram illustrates the Living Agent pattern. It shows a state transition from a 'control' state to a 'die' state. A solid line with an arrow points from the 'control' label to a small square on a vertical line. A dashed line with an arrow points from this square to another small square on a horizontal line. A dashed line with an arrow points from this second square to a third small square on a horizontal line, which is labeled 'die'.



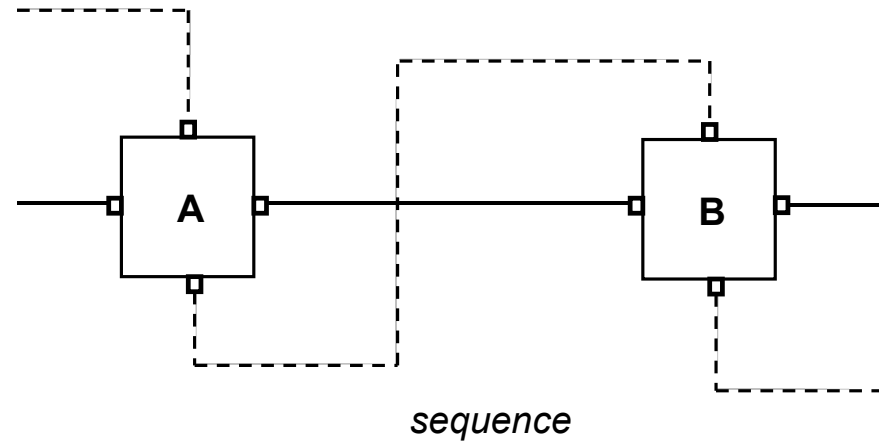
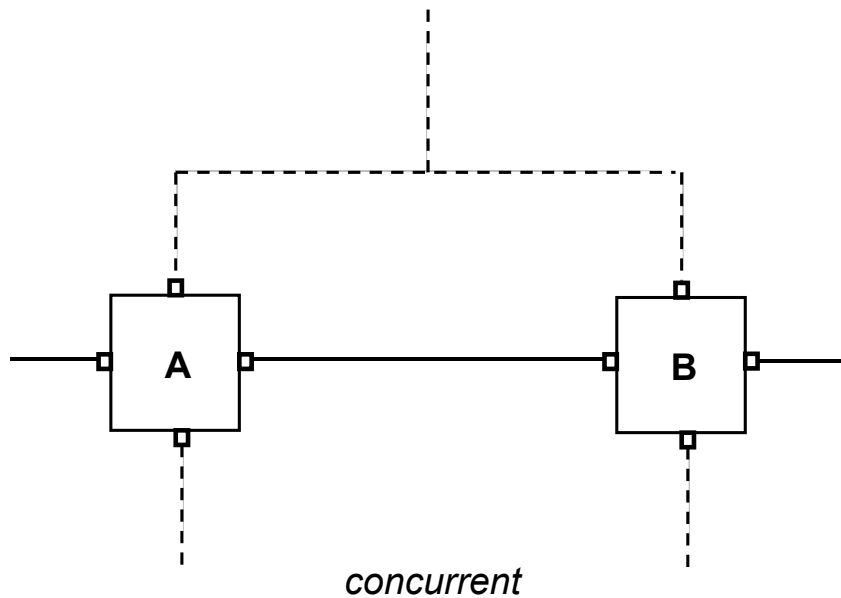
Actually, more
than just a design pattern:

a **black box micro-framework** - *parameterized component*

with (in this case) one hot-spot (***Behavior***)

Advantages of explicit control flow

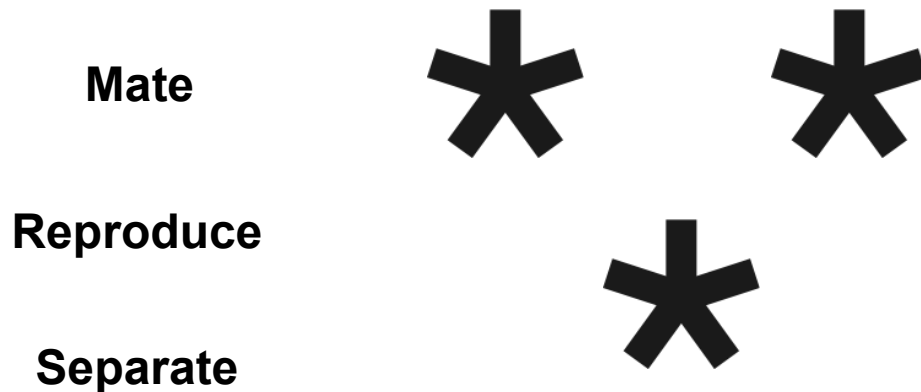
- decoupling activation logic from functionality
- more genericity



- fine grain control of intra-agent scheduling
(specification of temporal dependencies)
see next example/slide

3rd example: microsimulation of population evolution

Simplification of demography
micro-simulation model Destinie [INSEE 1999]



3 behaviors/components
(probabilistic state change):

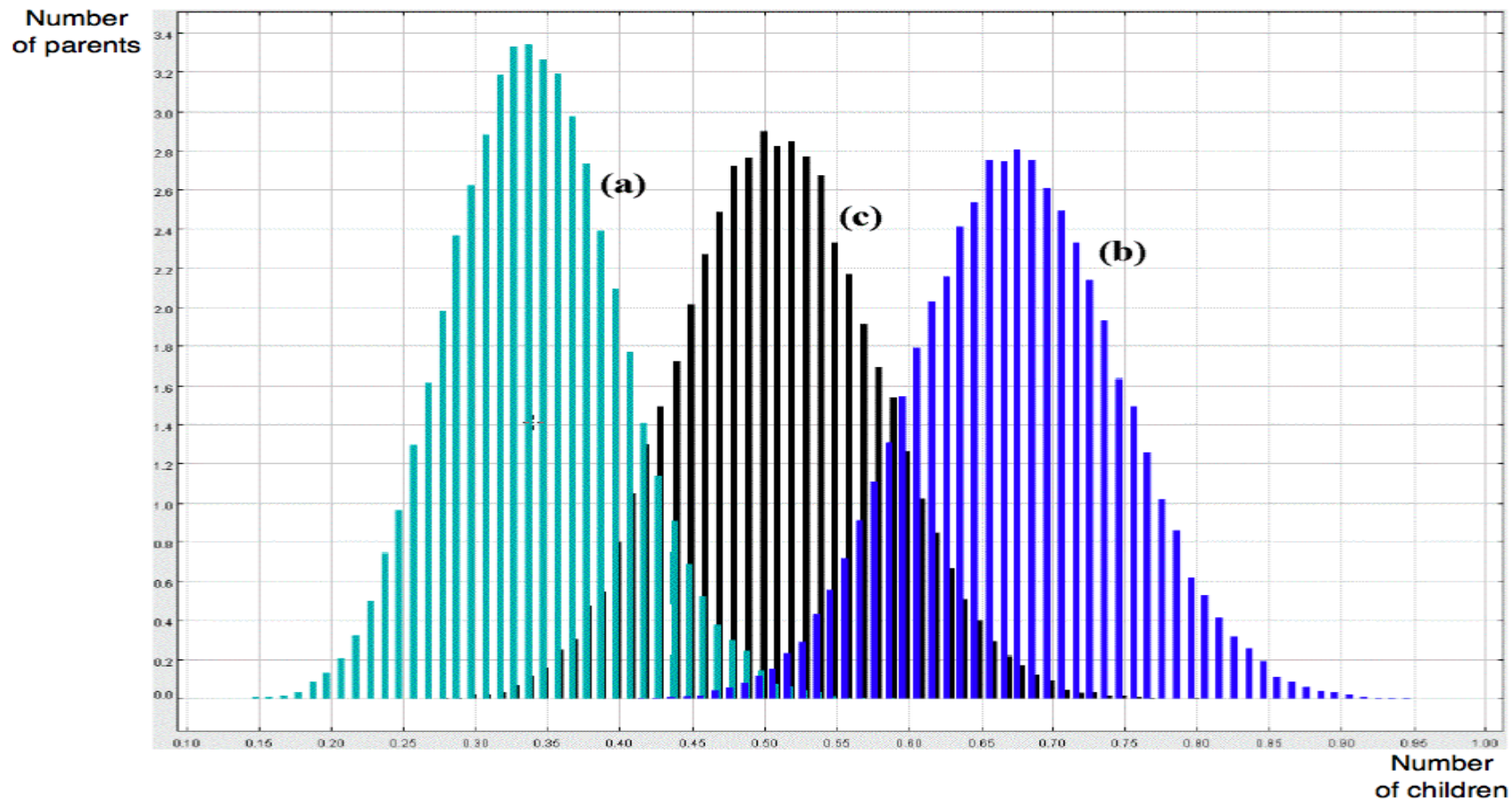


Issue for the designer of the model/simulation:
(*Note: often not an expert programmer*)

in what order should we activate these behaviors ?

Possible bias

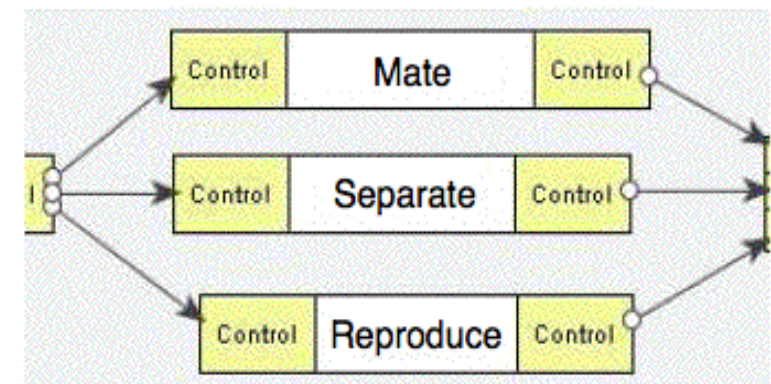
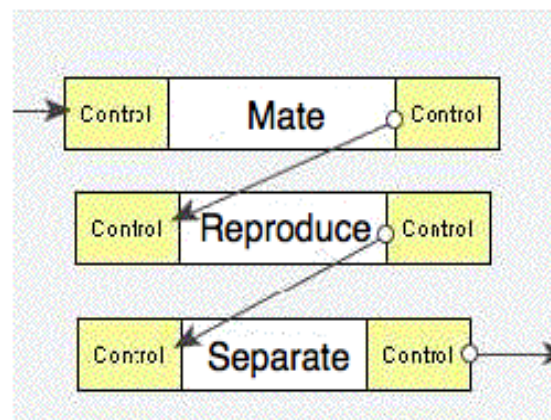
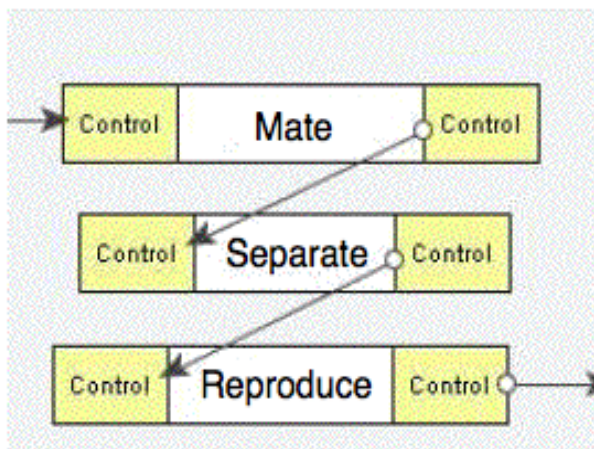
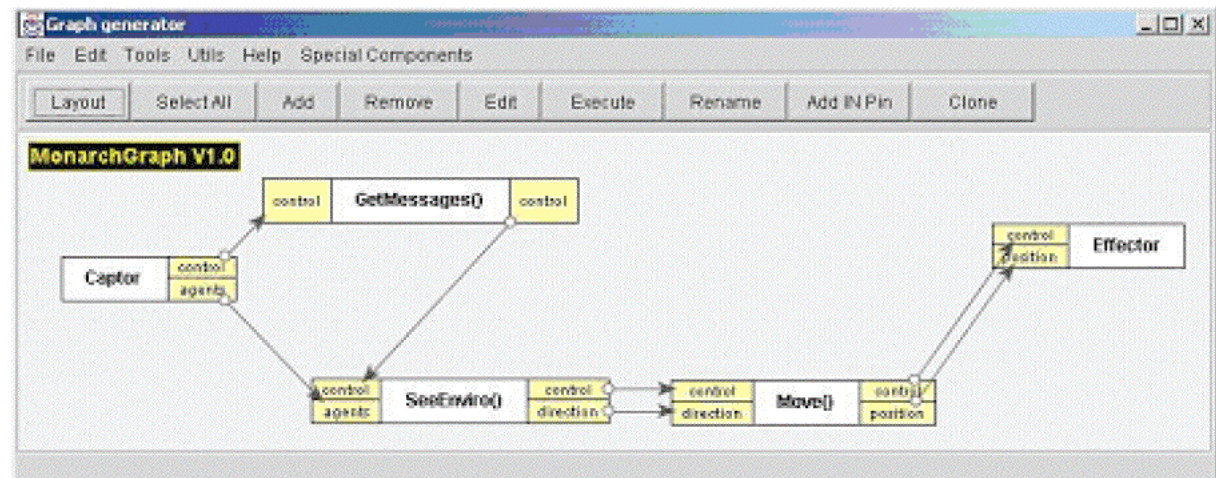
- (a) Mate ; Separate ; Reproduce
- (b) Mate ; Reproduce ; Separate
- (c) Mate || Reproduce || Separate



Specification of intra-agent temporal dependencies [Meurisse 04]

- Via control flow connexions
- Without any change to the components/behaviors
- GUI tool (CGraphGen)

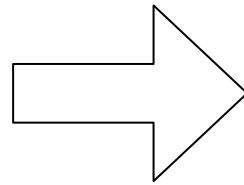
- (a) Mate ; Separate ; Reproduce
(b) Mate ; Reproduce ; Separate
(c) Mate || Reproduce || Separate



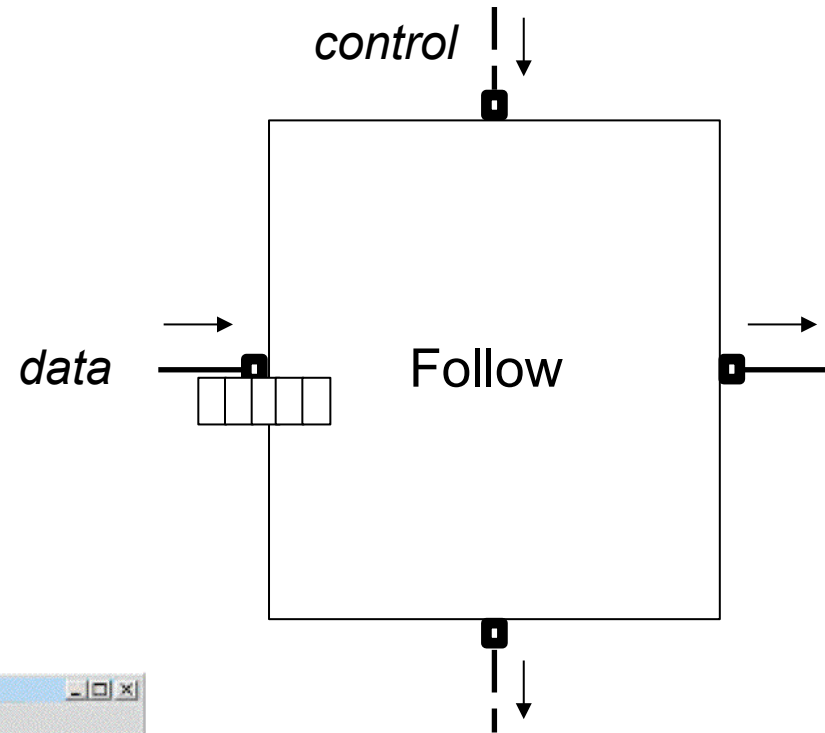
Reengineering of existing behavioral code [Meurisse 2004]

- a Java class (name)
- a method (name)
- method signature

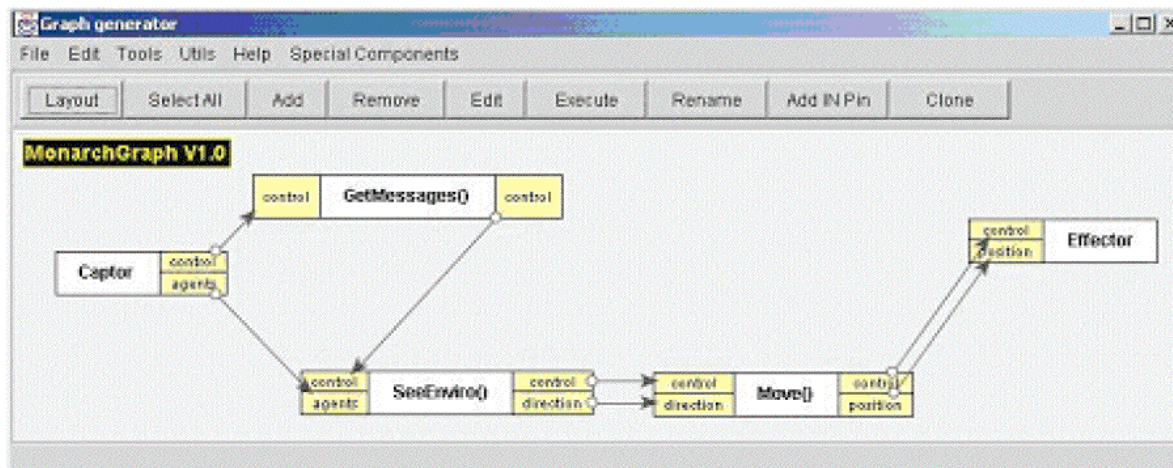
e.g., position Follow(position p)



CGraphGen tool



- typed ports
- one FIFO for each data input port

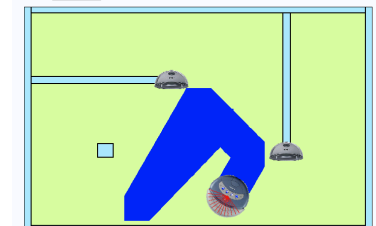
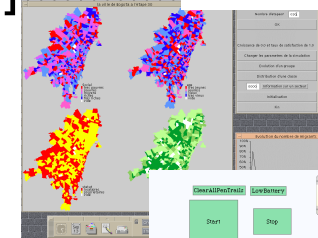
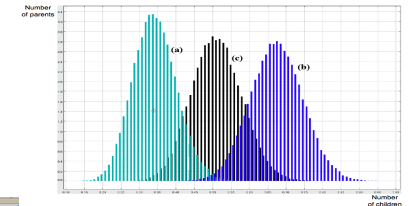
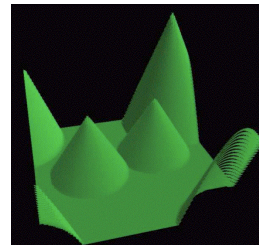
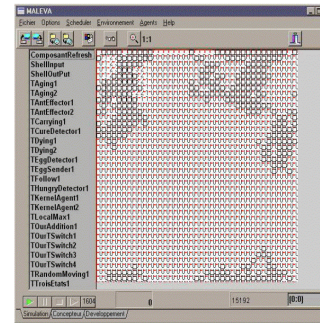


Implementations

- Delphi [Lhuillier 1998]
- Java/JavaBeans [Meurisse 2001]
- C++ [Meurisse 2004]
- Smalltalk (Squeak) [Bouraqui et al. 2006]
 - *(independent implementation)*

Experiments

- Eco-systems [Lhuillier 1998]
- Traffic simulation [LeCerf and Pintado 1997]
- Ant colony (MANTA) [Guillemet et al. 1999]
- Sugarscape [Meurisse 2004]
- Population microsimulation (DESTINIE) [Meurisse 2004]
- Urban migration (SimBogota) [Vanbergue 2002]
- Vacuum cleaner robot (Romba) [Bouraquad et al. 2006]



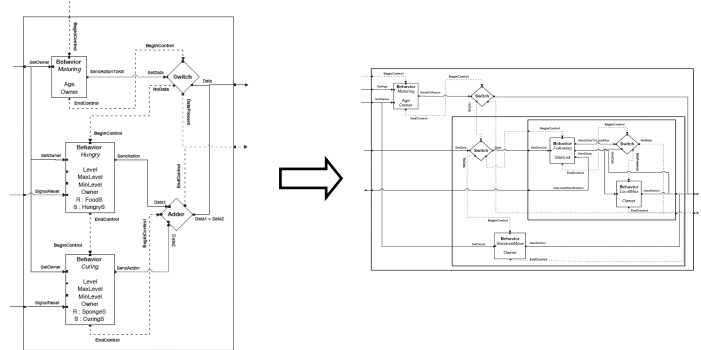
Directions

- Behavior components library
 - concrete components
 - » for multi-agent simulation
 - » *for other technical (ex: interaction) and application (ex: e-commerce) domains*
 - abstract components (design patterns)
 - control components
- Behavior dynamicity
 - (see next slide)
- Control flow vs coordination language term
 - (see next next slide)

Dynamicity (dynamic change of behavior)

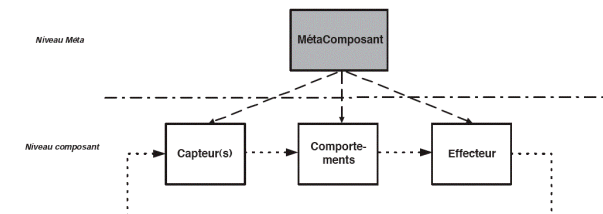
- Motivations

- Internal change - Model change of behavior -
 - » Ex: from an egg, to a larva, to a worker ant or to a queen
- External change - Context-awareness
 - » Ex: adapt to change of resources



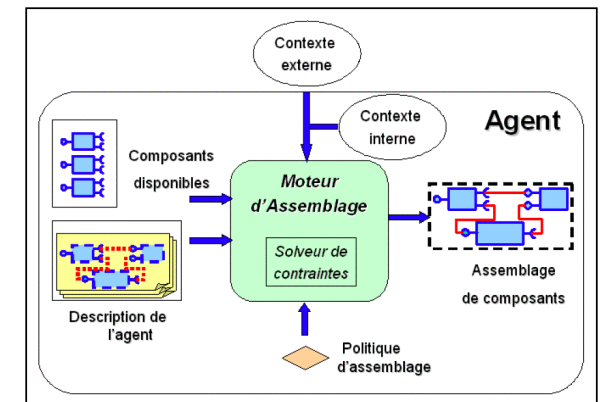
- Currently implemented through a specific behavior/component [Guillemet and Haïk 1998]

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

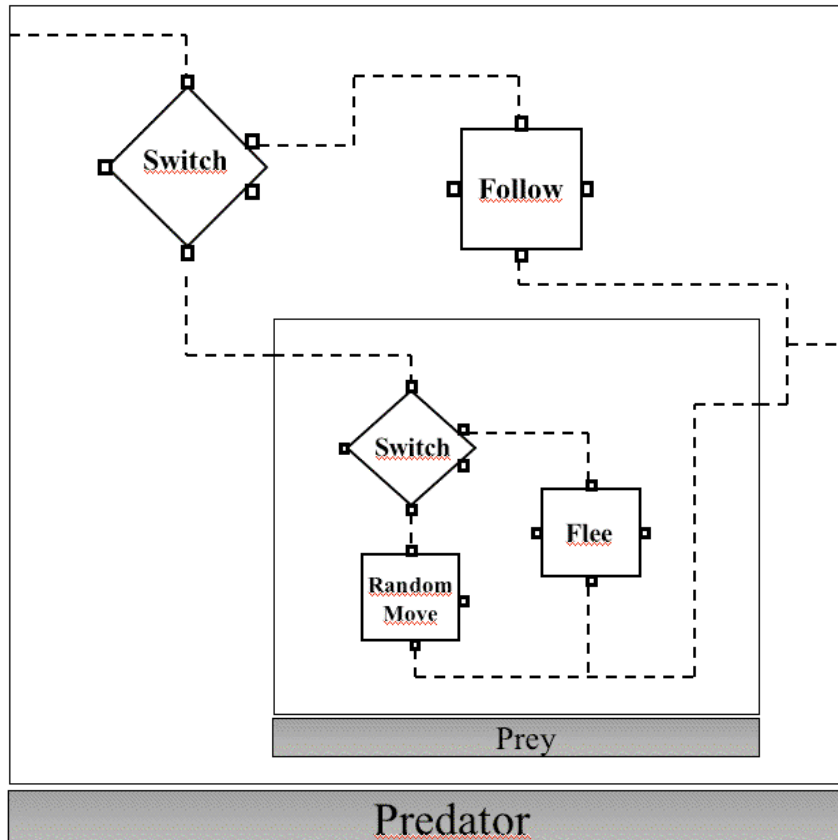


- Ongoing work:

- Use a higher-level general reassemblage model/mechanism
- MadCar [Grondin et al. 2004]
- Based on concepts of configuration, roles and policies



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

$\text{isPrey.Follow} \parallel \text{isPredator.Flee} \parallel (\text{isNoPrey.RandomMove} + \text{isNoPredator.RandomMove})$

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 on MALEVA

- Components can be useful to help at decomposing/recomposing agent architectures
- Componentification (reification) of control
 - Control ports
 - Control connexions
 - Control components
- Separation between data flow and control flow
 - For decoupling activation from functionality (more genericity)
 - For a fine-grained control of intra-agents scheduling,
 - Thus helps at controlling possible bias
- MALEVA model is fine grained (behaviors)
 - But optimizations are possible
- Notion of composite component (encapsulation of hierarchy)
- Dynamic change of behaviors

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]

References

- J.-P. Briot, Composants et agents : évolution de la programmation et analyse comparative, *Technique et Science Informatiques (TSI)*, 33(1-2):85–115, 2014.
- J.-P. Briot, T. Meurisse, F. Peschanski, Architectural design of component-based agents: A behavior-based approach, *Programming Multi-Agent Systems – ProMAS 2006*, LNCS, N° 4411, Springer, 2007, pp. 73–92.
- L. Gasser, J.-P. Briot, Agents and Concurrent Objects, *IEEE Concurrency*, 6(4):74–81, 1998.
- G. Grondin, N. Bouraqadi, L. Verouter, MaDcAr: An Abstract Model for Dynamic and Automatic (Re-)Assembling of Component-Based Applications, *9th Component-Based Software Engineering International Symposium (CBSE'2006)*, LNCS, N° 4063, Springer, 2006, pp. 360–367.
- Z. Guessoum, J.-P. Briot, From active objects to autonomous agents, *IEEE Concurrency*, 7(3):68–76, 1999.
- K. Kuwabara, T. Ishida, N. Osato, AgenTalk: Coordination Protocol Description for Multiagent Systems, *1st International Conference on Multiagent Systems (ICMAS'95)*, San Francisco, CA, USA, June 1995, pp. 445.
- J. Odell, Objects and Agents Compared, *Journal of Object Technology*, 1(1), 2002.
- K. Sycara K, S. Widoff, M. Klusch, J. Lu, Larks: Matchmaking among heterogeneous agents in cyberspace, *Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*, 5(2):173–203, 2002.
- M.-J. Yoo, J.-P. Briot, J. Ferber, Using components for modelling intelligent and collaborative mobile agents, *IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WetIce'98)*, Palo Alto, CA, USA, June 1998, pp. 276–281.

Thank you !
Questions ?
