

# Outline

## Introduzione ai sistemi multi-agente basati su logica computazionale

Paolo Torroni  
DEIS, Università di Bologna

1. Introduction to multi-agent systems and their applications
2. Logic (programming)-based agent languages, architectures, and frameworks

CILC 2004 Tutorial

Parma, 16 Giu 2004

2

# Overview

- Agent research: theory and practice
  - difficult even to *define* the overloaded concept of 'agent'
  - many industrial/commercial applications already exist which directly or indirectly refer to the agent paradigm
- Part One: application perspective
  - to motivate the use of agents in concrete scenarios,
  - to single out a set of characteristics that distinguish agents from other computational paradigms
- Discuss about the use and importance of logics in the development of agent applications

CILC 2004 Tutorial

Parma, 16 Giu 2004

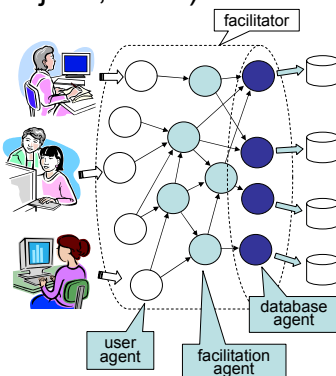
4

## Part One

### Introduction to multi-agent systems and their applications

## Virtually integrated Distributed DBs (Format-X, Fujitsu, 1998)

- Environments where databases are necessarily distributed:
  - security
  - management
  - sharing maintenance
- E.g.: *Steel Plant*: a steel making company can quickly find parts to repair equipment
  - from within the company
  - from equipment manufacturers
  - from plants of other steel-making companies



- Each agent has a set of rules {condition} → {agent}, e.g.: [Bay area] and [motors] → bay.area@tcals.or.jp
- When a user sends a request to the user agent, a database navigation phase is started, based on the *conditions* of requests
- If the conditions are consistent with the conditions of a rule (no contradiction): *fwd*
- **Advantages:**
  - load balance
  - agent-local maintenance
  - distribution of maintenance and management
  - agent-based security
  - scalability
  - robustness

CILC 2004 Tutorial

Parma, 16 Giu 2004

6

## Scenario #1: knowledge integration

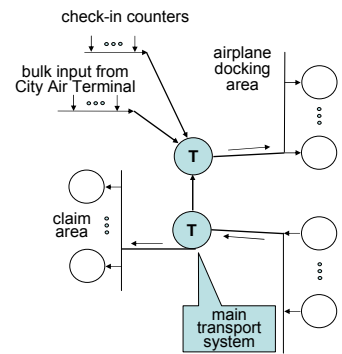
- Fujitsu's Format-X is an example of knowledge integration based on agents
- Some other experiences:
  - Information and Knowledge intergration (SAGE and FIND Future), Fujitsu (2000)
  - Web-linked Integration of Network-based Knowledge (WINK), GFormula (2001)
  - Distributed Information Systems and Intelligent Information Integration, ITC-IRST (Trento)
  - Oracle Intelligent Agents (2000)
  - Business Process Management, Agentis Software
- Advantages of an agent-based architecture

## Baggage Handling System (Flavors Technology Inc.,)



### Denver Int'l Airport

- 4000 telecards that carry bags at 28 kmph, servicing 20 airlines along 32 km of track
- control handled by 64 PCs, hooked to 5,800 electric eyes, 315 radio receivers, 182 switches, and at least 60 bar code scanners
- it didn't work!! 1 year delay, \$500K per day



## Problems with a centralized control

- Dimension of the problem
- Complexity of the solution (controller programs and mainframe)
- Chaos (performance or several components depending on many factors)
- Maintenance
- Difficulty in tracking problems
- Real-time scheduling

## Scenario #2: distributed scheduling

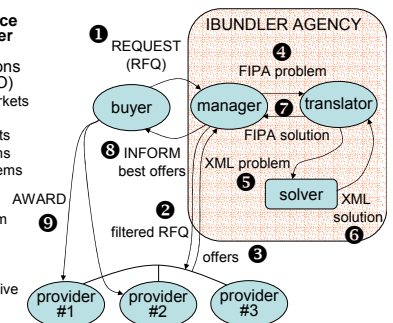
- Motivation:
  - **uncertainty** in the availability of resources
  - **multi-objective optimization** problem
  - **accuracy of data** (better if optimization is solved at the source)
  - **sub-optimal solutions** are often enough
  - **information persistence, self-configurability, interoperability of software** call for the need of distribution of information processing and integration of planning and execution
- Advantages:
  - scalability (time taken to get to a sub-optimal solution)
  - modelling (agents can model resources and their constraints)
- Other experiences:
  - ANTS (Agent Network for Task Scheduling and Execution), Deneb robotics, SRI Int.
  - IntelliDiary, Distributed Schedule Management System, Fujitsu

## Similar problems: supply chain management and manufacturing

- GM Paint System, Flavors Technology Inc.
- Distributed control, Rockwell Automation
- Manufacturing Agility Server, Flavors Technology Inc.
- Holonic Manufacturing Systems (HMS), consortium including: Hitachi, Toshiba, Softing GmbH, Aitec, Anca, Broken Hill, Mandrelli SpA, Nestlé, Yaskawa, Rockwell/Allen Bradley
- Supply chain planning, Lost wax

## Market based resource allocation

- IBundler: **negotiation service** for buying agents and **winner determination service** for reverse combinatorial auctions with side constraints (ISOCO)
  - negotiate over multiple markets
  - offer aggregation
  - business sharing constraints
  - constraints over single items
  - constraints over multiple items
  - specification of providers' capacities
  - multiple bids over each item
  - combinatorial offers
  - multi-unit offering
  - packing constraints
  - complementary and exclusive offers

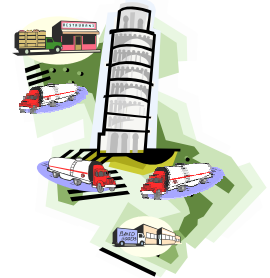


## Scenario #3: e-procurement

- Advantages for the buyer:
  - reduced time and cost of the whole sourcing process
  - simplified decision-making and supplier selection process
  - process automation
  - quality improvement and time to market reduction
- Advantages for the seller:
  - companies can access new markets
  - selling process cost reduced
  - competitive advantage for the buyer
- Which parts can be automated, and how?

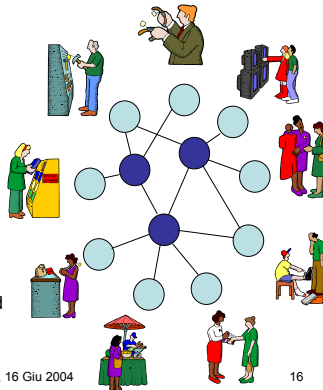
## Scenario #4: Distribution and logistics

- Highly dynamic domain:
  - weather
  - work stoppages due to funds/material shortages
  - congested traffic
  - ...
- What can be done?
  - trucks ↔ agents
  - automatic negotiation (e.g., Contract Net Protocol)
  - find similar orders
  - transportation management and just-in-time vehicle re-routing,



## Scenario #5: customer-oriented services

- High diversity of customers:
  - needs
  - knowledge
  - constraints
  - priority
- E.g. U.S. State Public Housing Authority (Agentis)
  - tear down and rebuild public housing for 160,000 families
  - off-the-shelf solutions are no good: highly customized management required
  - traditional approach 1 year



## More scenarios

- **Agents for armed forces**, Cambridge Consultants W.P.,
- **Health care** (information selection and filtering, proactive monitoring), Cambridge Consultants W.P.
- **Crawling agents**, iSOCO, GruSMA
- **Games and film industry** (The Lord of The Rings)
- **Mission critical** unmanned vehicle piloting, agent-software
- **PDA-oriented services**: virtual secretaries and recommender systems, travel planners, ..., see Siemens MOTIV and AgentCities projects
- **Human-Machine Interfaces**: humanoid cartoon characters, Fujitsu (1999) and Animation systems for interface agents, Fraunhofer IGD
- **Network management**, Fujitsu (2001)
- **Simulation**: Central JR Shinkansen, Flavors Technology Inc., (1998), Air traffic management, Intelligent Automation Inc., Multi-modal transportation, Ketensimulator, TNO

## Why agents?

1. **Agents don't get tired or frustrated with negotiation**
  2. **Agent negotiation can happen more often**
  3. **In complex negotiation settings, automated agents will be more successful at obtaining better deals because they can keep track of more options** (K. Woghiren, LostWax)
- ...this allows for:
    - Market searching, sorting, monitoring, and negotiation
    - Dynamic solutions
    - Customer-oriented solutions
  - Other experiences:
    - Living markets, living systems AG (2001; now Whitestain Technologies)
    - Lost wax
    - Adaptive Deal Flow Optimization, living systems AG (2002)

## What challenges

- buyers/market owners need to **trust** agents
- users need be able to **specify** the behaviour of agents
- infrastructures / **interoperability** !!
  - **CUSTOMERS** do not have the financial or technical clout to force the development of interoperable solutions, or to test products to ensure that they really are interoperable
  - **USERS** and IT staff in large business may want interoperability, but will have a difficult time justifying the possible long-term savings vs. the short-term costs (O. Omidvar, ATP PM)
- agents must be **intelligent** in order to be able to negotiate effectively
- agents need to be able to **adapt** to new scenaria  
→ *role of logic?*

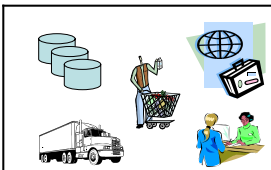
## Summary: you should consider looking at agent architectures...

- When the **metaphor is appropriate** (*customer modelling, recommender systems, interfaces*)
- When there is a decision to take based on **multiple sources**, on **large amounts of data**, and in a **dynamic environment** (*e-markets, logistics*)
- For **complex control tasks**, when it is not possible to use a centralized controller and **decentralized problem solving is needed** (*supply chain management, manufacturing*)
- For **simulation of populations of proactive individuals**, when a mathematical model is not available (*traffic, games, cinema*)
- When it is necessary to **integrate and share knowledge from multiple sources** (*databases, business support*)
- Where **autonomous problem solving is needed** (*electronic trading, space crafts*)
- With **high run-time uncertainty**, or **incomplete or complex information** (*telecom services across multiple providers*)

## ...what is an agent...?

- **Basic components:**
  - Data structures (mental state)
  - Control structure (life cycle)
  - I/O (communication language & protocols)
- **Design choices**
  - Pro-active (goal-oriented) vs. reactive (swarm intelligence)
  - Standard (inter-platform) vs. proprietary (intra-platform)
  - Cooperation vs. competition
  - Adaptive vs. static
  - ...

## Domain-dependent categories

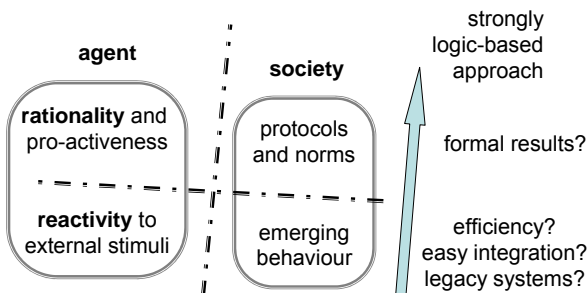


	Pro-activeness	Standards	Robustness	Properties	Cooperation	Adaptability
<i>Knowledge integration</i>	?	✓	?	✓	✓	?
<i>Distributed scheduling</i>	✓	✗	✓	✓	✓	✓
<i>E-procurement</i>	✓	✓	✓	✓	✗	✓
<i>Distribution &amp; logistics</i>	✓	?	✓	✓	✓	✓
<i>Service composition</i>	?	?	✗	✓	✓	✓

## Is agent technology mature enough?

- A number of applications for “closed” domains
- *Open domains*: importance of fundings by State
- Some problems to sort out:
  - trust, security (monitoring, mobile agents)
  - user-interface
  - extensive testing
  - availability of standards and general-purpose development framework
- New challenges
- Logics?
  - For prototyping (AOSE)
  - For intelligence (reasoning, goals, consistency)
  - For verification (individuals, interactions)

## Where do we use logics?



## Part Two

Logic (programming)-based agent languages, architectures, and frameworks

## Developing agent-based applications

### • Basic elements

- Agents:
  - Knowledge representation
  - Control structure
- Agent systems:
  - Agent Communication Languages
  - Ontologies
  - Protocols
  - Institutions and norms

## OOB vs. AOB [Sho93]

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>• <i>Basic unit:</i> <ul style="list-style-type: none"> <li>– object</li> </ul> </li> <li>• <i>Parameters defining state of basic unit:</i> <ul style="list-style-type: none"> <li>– unconstrained</li> </ul> </li> <li>• <i>Process of computation:</i> <ul style="list-style-type: none"> <li>– message passing and response methods</li> </ul> </li> <li>• <i>Types of message:</i> <ul style="list-style-type: none"> <li>– unconstrained</li> </ul> </li> <li>• <i>Constraints on methods:</i> <ul style="list-style-type: none"> <li>– none</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• <i>Basic unit:</i> <ul style="list-style-type: none"> <li>– agent</li> </ul> </li> <li>• <i>Parameters defining state of basic unit:</i> <ul style="list-style-type: none"> <li>– beliefs, commitments, capabilities, choices, ...</li> </ul> </li> <li>• <i>Process of computation:</i> <ul style="list-style-type: none"> <li>– message passing and response methods</li> </ul> </li> <li>• <i>Types of message:</i> <ul style="list-style-type: none"> <li>– inform, request, offer, promise, decline, ...</li> </ul> </li> <li>• <i>Constraints on methods:</i> <ul style="list-style-type: none"> <li>– honesty, consistency, ...</li> </ul> </li> </ul> |
|---|--|

## BDI logics

- Software Agent: a system that enjoy the properties of
  - **Autonomy:** make decisions based on an internal state
  - **Reactivity:** perceive the environment and respond in a timely fashion to changes that occur in it
  - **Pro-activeness:** take the initiative to achieve goals
  - **Social ability:** interact with other agents via an ACL
- **Beliefs:** information about the state of the environment (*informative state*)
- **Desires:** objectives to be accomplished (*motivational state*). Adopted desires are often called Goals
- **Intentions:** currently chosen course of action (*deliberative component*)

## BDI architecture

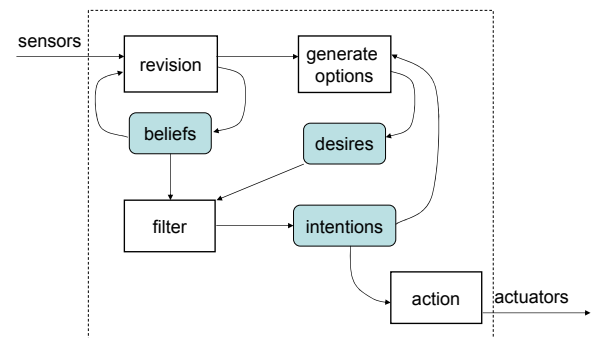
- BDI formalization has 2 main objectives:
  - To build practical systems
  - To build formally verifiable systems
- Building blocks:
  - Interpreter and cycle theory
  - Logics
  - Semantics

## BDI interpreter (cycle)

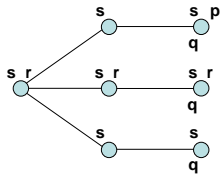
```

Initialize-state();
repeat
  options := option-generator (event-queue);
  selected-options := deliberate (options);
  update-intentions (selected-options);
  execute();
  get-new-external-events();
  drop-successful-attitudes();
  drop-impossible-attitudes();
end repeat
  
```

## BDI architecture



# Temporal reasoning: Time trees



- Inevitably always  $s$
- Inevitably eventually  $q$
- Optionally always  $r$
- Optionally eventually  $p$

- Time tree: temporal structure with
  - Branching time future
  - Single past
- A particular time point is called *situation*
- Standard temporal operators operate over state and path formulas

# Example [RG92]

- John acquires a goal to quench its thirst. He believes that he can satisfy it in one of two ways:
  - (a1) open the tap, (a2) fill the glass, (a3) drink water from the glass
  - (b1) get to a state where he has soda, (b2) fill the glass, (b3) drink soda from the glass
- (b1) is a **sub-goal**, the remaining parts of its plans are **atomic actions**
- To get to the state where he has soda, John has to (c1) open the fridge, and (c2) remove the soda bottle
- John's beliefs:
  - about the effects of atomic actions
  - about the possibility of carrying them out successfully
  - about the possible failure of his plan to obtain soda

- BEL (inevitable  $\Box$ (have-soda; fill-glass; drink)  $\supset$ quenched-thirst)
- BEL (inevitable  $\Box$ (open-tap; fill-glass; drink)  $\supset$ quenched-thirst)
- BEL (inevitable  $\Box$ (open-fridge; remove-soda)  $\supset$ have-soda)
- BEL (optional  $\Diamond$ (have-soda; fill-glass; drink))
- BEL (optional  $\Diamond$ (open-tap; fill-glass; drink))
- BEL (optional  $\Diamond$ (open-fridge; remove-soda))
- BEL (inevitable  $\Box$ ( $\neg$ (soda-in-fridge)  $\supset$ inevitable  $\neg$   $\Diamond$ (remove-soda))
- GOAL (inevitable  $\Diamond$ (quenched-thirst))

BEL	GOAL	INTEND	done	succeeded
$B$	$\Diamond$ (quenched-thirst)	-	-	-
$\neg \Diamond$ (remove-soda)	<i>idem</i>	-	open-fridge	open-fridge
<i>idem</i>	<i>idem</i>	fill-glass; drink	open-tap	open-tap
<i>idem</i>	<i>idem</i>	drink	fill-glass	fill-glass
quenched-thirst	-	-	drink	drink

John is **not** blindly committed  $\Rightarrow$  he can choose an alternative plan

# Is BDI logic implemented in practical systems?

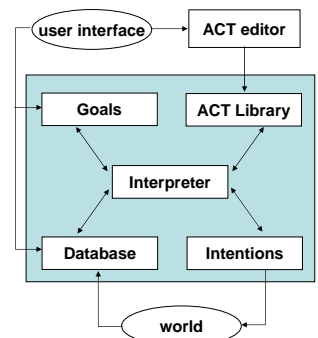
- Many implemented systems are *inspired* to BDI concepts...
- Problem: the time taken by agents to reason is potentially unbounded !!
- The abstract architecture is an *idealization* that faithfully captures the theory, **not a practical** system for rational reasoning
- Solution: some important 'choices of representation' (simplifications) must be made...

# BDI operationalized (PRS, dMARS)

- Only beliefs about the *current state* of the world are explicitly represented
- Only *ground* sets of literals with *no disjunctions or implications*
- The information about the means of achieving certain future world state is coded in a *plan library* (special beliefs)
- *Intentions* are represented *implicitly* using a *conventional run-time stack* of hierarchically related plans
- Each *plan* consists of:
  - a trigger (invocation condition)
  - a context (precondition)
  - a maintenance condition (to hold true during the execution)
  - a body (course of *goals* / primitive actions)

# PRS components [PRS01]

1. A database containing current *Beliefs*
2. A set of current *Goals* to be realized
3. A set of plans (*Acts*)
  - goal achievement
  - reaction to situations
4. *Intentions* containing chosen plans
5. An *Interpreter*



## An Agent Oriented Programming computational framework: **AGENT-0**

- More or less contemporary to BDI
- Builds on work by Cohen and Levesque
- A different set of modalities
  - Beliefs
  - Capabilities
  - Choices
  - Commitments
- Stress on the *social aspect* (Commitments)

## Components of an AOP system

1. A restricted **formal language** with clear syntax and semantics for describing **mental state**;
2. An **interpreted programming language** in which to **define and program agents**, with primitive commands such as **REQUEST** and **INFORM**;
  - the semantics of the programming language will be required to be **faithful** to the semantics of the mental state;
3. An **“agentifier”**, converting neutral devices into programmable languages

## Agent programs

- A program is constituted by:
  - a definition of *capabilities* and *initial beliefs* + fixing of *time grain*, and
  - a sequence of **conditions under which the agent will enter into new commitments** (*commitment rules*)
- Example (commitment rule):
 

```
(COMMIT (?a REQUEST ?action)
  (B (now (myfriend ?a)))
  (?a ?action)).
```
- Syntax:
 

```
(COMMIT msgcond mntlcond (agent action)*)
```

## Communicative acts

- 3 types of communicative actions:
  - **INFORM**
  - **REQUEST**
  - **UNREQUEST** (to cancel a request)
- Example:
 

```
(REQUEST 1 a
  (REQUEST 5 b
    (INFORM 10 c fact ))).
```

## Agent Communication Languages

- Two major proposals
  - **KQML** (1993 - ~1998)
  - **FIPA ACL** (1996 - now)
- Define a number of *communicative actions / performatives*
- Semantics based on *mental states* (KQML):
  1. An *intuition* given in natural language
  2. An expression describing the illocutionary act
  3. Pre-conditions for sender and receiver
  4. Post-conditions in case of successful receipt
  5. Completion condition (final state of a conversation)
  6. Any comments

## KQML: semantics for *tell* [LF98]

1. A states to B that A believes the content to be true.
2.  $BEL(A,X)$
3. **Pre(A)**:  $BEL(A,X) \wedge KNOW(A,WANT(B,KNOW(B,S)))$   
**Pre(B)**:  $INT(B,KNOW(B,S))$ ,  
 where S may be any of  $BEL(B,X)$ , or  $\neg(BEL(B,X))$ .
4. **Post(A)**:  $KNOW(A,KNOW(B,BEL(A,X)))$   
**Post(B)**:  $KNOW(B,BEL(A,X))$
5. **Completion**:  $KNOW(B,BEL(A,X))$
6. The completion condition holds, unless a *sorry* or *error* suggests B's inability to acknowledge the *tell* properly, as is the case with any other performative.

## FIPA ACL semantics for *inform* [FIP01]

- Semantic Language
  - Feasibility Preconditions (FP) for a CA:
    - Ability preconditions
    - Context-relevant preconditions
  - Rational Effect (RE)
- $\langle i, \text{inform}(j, \phi) \rangle$
- FP :  $B_i \phi \wedge \neg B_i (B_i f_j \phi \vee U i f_j \phi)$
- RE :  $B_j \phi$
- Where  $B_i f_j \equiv B_j \phi \vee B_j \neg \phi$ ; U means uncertainty

## Social semantics of ACL

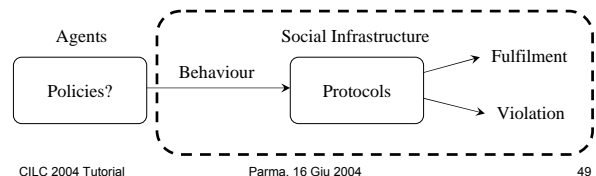
- Some questions...
  - Why constrain agents' social acts?
  - Why refer to a particular agent architecture?
  - How to *verify* communication?
  - How to approach *openness* and heterogeneity?
- Other approaches!
- Semantics based on social *commitments*
  - Singh & Yolum [Sin98,YS02]
  - Colombetti, Fornara & Verdicchio [CFV02,FC02,...]
- Semantics based on *expectations* [SOCS]

## The SOCS social model [AGM+\*]

- Perspective: openness = no information about internals of agents
- Focus: different aspects of interaction (ACL, interaction protocols, properties of interaction)
- Aim: Declarative representation + operational model
  - Possibility to verify interactions and prove properties
  - ACL semantics, interaction protocols, and *properties* specified using the same formalism!

## Protocols

- Agents behave according to their own policies
- Social expectations can be used:
  - to check the correct functioning of the society
  - to suggest to the agents a course of actions
- Protocols are defined through Social Integrity Constraints:
- The society generates expectations out of protocols & events



## Social Integrity Constraints (SICs)

- SICs ::=  $[\chi \rightarrow \varphi]^*$
- $\chi ::= (\neg)H(Event [, Time])$
- $\varphi ::= \vee \{ \wedge (\neg)E/NE(Event [, Time]) / constraints \}$
- Examples
  1.  $\neg H(tell(A,B,propose),T), T < T1 \rightarrow NE(tell(B,A,accept),T1)$
  2.  $H(tell(X,Y,ask,D),T) \rightarrow E(tell(Y,X,yes,D),T'), T' > T \vee E(tell(Y,X,no,D),T')$
  3.  $H(tell(X,Y,yes,D),T) \rightarrow NE(tell(Y,X,no,D),T')$
  4.  $H(tell(X,Y,no,D),T) \rightarrow NE(tell(Y,X,yes,D),T')$
  5.  $H(tell(X,Y,S,D),T) \rightarrow NE(tell(Y,X,S,D),T'), T > T'$

## SIC-based ACL semantics and Interaction Protocol specification

- The semantics of communicative acts can be recovered into the SIC formalism:
  - E.g.: semantics of **promise**
- $H(\text{promise}(Sender, Receiver, P, Context), T_p)$   
 $\rightarrow E(\text{do}(Sender, Receiver, P, Context), T_d):$   
 $T_d \leq T_p + \tau$
- Similar to the idea of commitment
  - Verifiable!

## Declarative Semantics

Given a society and a set **HAP** of events...

1. a set of expectations **EXP** is *admissible* iif  
 $SOKB \cup HAP \cup EXP \models SICs$
2. a set of expectations **EXP** is *coherent* iif  
 $\{E(p), NE(p)\} \notin EXP$
3. a set of expectations **EXP** is *consistent* iif  
 $\{E(p), \neg E(p)\} \notin EXP$   
 $\{NE(p), \neg NE(p)\} \notin EXP$
4. a coherent, consistent, and admissible **EXP** is *fulfilled* iif  
 $HAP \cup EXP \models \{E(p) \rightarrow H(p)\} \cup \{NE(p) \rightarrow \neg H(p)\}$
5. if each coherent and consistent admissible set of expectations is not fulfilled, HAP produces a *violation* in the society



## Agents in logic

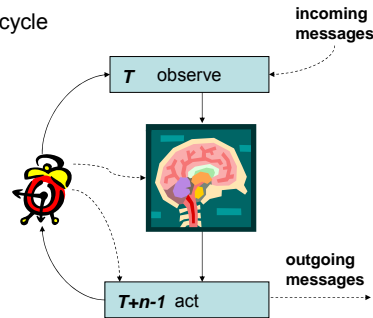
- Agent architectures
  - ✓ BDI
  - ✓ Agent-0
  - KS-agents, STT, ...
  - ALIAS
  - Speculative Computation
  - MINERVA
  - Kakas et al., KGP, ...
  - 3APL
  - IMPACT
  - Linear Logics
  - MetateM
  - ConGolog
  - Dylog
  - ...
- Agent interaction
  - ✓ KQML, FIPA
  - ✓ Yolum & Singh / Fornara, Colombetti & Verdicchio
  - ✓ SOCS social infrastructure
  - norms / institutions (deontic logic-based approaches)



## KS-agents [KS99]

The *observe-think-act* cycle

- To *cycle* at time  $T$
- *observe* any inputs at time  $T$
- *think*
- *select* one or more actions to perform
- *act*
- *cycle* at time  $T+n$



## Thinking component

- *Backward reasoning* (ALP) combined with *forward reasoning* (ICs): **IFF proof-procedure** [FK97]
- The set of predicates is partitioned into:
  - closed predicates (iff definitions)
  - open representing actions of the agent
  - open representing events in the environment
  - “constraint” predicates
- Internal state:
  - Beliefs which are iff-definitions
  - Beliefs which are observations
  - Goals in if-then form

## Goals

- Queries to database:
  - for-all  $E$  [if *employee* ( $E$ ) then exists  $M$  *manager* ( $M, E$ )]
- Obligations:
  - if *true* then co-operate
- Prohibitions:
  - for-all Time [if *do* (steal, Time) then *false*]
- Reaction:
  - for-all Agent, Act,  $T1, T2$
  - if *happens* (*ask* (Agent, *do* (Act,  $T2$ )),  $T1$ )
  - exists  $T, T'$
  - then [*confirm* (*can-do* (Act,  $T2$ ), [ $T, T'$ ])] & *do* (Act,  $T2$ ) &  $T1 < T < T' < T2$

## Observations

- Positive observations:
  - simple, variable-free atomic predicates
  - e.g.: *employee* (mary)
- Negative observations:
  - variable-free implications with conclusion *false*
  - e.g.: if *employee* (bob) then *false*

## The unified agent cycle

To **cycle** at time  $T$ ,

- **record** any observations at time  $T$ ,
- **resume** the proof procedure, **giving priority to forward reasoning** with the new observations,
- **evaluate** to *false* any disjuncts containing subgoals that are **not marked as observations** but are atomic actions to be performed at an earlier time,
- **select** subgoals, that are **not marked as observations**, from among those that are atomic actions to be performed at times consistent with the current time,
- **attempt to perform** the selected actions,
- **record the success or failure** of the performed actions and **mark them as observations**,
- **cycle** at time  $T + n$ .

## Example

*happens* (become-thirsty,  $T$ )

→ *holds* (quench-thirst,  $[T1, T2]$ ) &  $T \leq T1 \leq T2 \leq T+10$

*holds* (quench-thirst,  $[T1, T2]$ ) ↔ *holds* (drink-soda,  $[T1, T2]$ ) or

*holds* (drink-water,  $[T1, T2]$ )

*holds* (drink-soda,  $[T1, T2]$ ) ↔ *holds* (have-glass,  $[T1, T1]$ ) &

*holds* (have-soda,  $[T1, T2]$ ) &

*do* (drink,  $T2$ ) &

$T1 < T1' < T2 \leq T1'$

*holds* (have-soda,  $[T1, T2]$ ) ↔ *do* (open-fridge,  $T1$ ) &

*do* (get-soda,  $T2$ ) &

$T1 \leq T2$

*holds* (drink-water,  $[T1, T2]$ ) ↔ *holds* (have-glass,  $[T1, T1]$ ) &

*do* (open-tap,  $T1$ ) &

*do* (drink,  $T2$ ) &

$T1 < T1' < T2 \leq T1'$

- **Similarities:**
  - cycle
  - both distinguish goals and beliefs as separate components of an agent's internal state
  - both have two kinds of beliefs: facts and plans
- **Differences:**
  - **BDI:** distinguishes **intentions as a separate component**, **KS: intentions are treated as goals** that represent the actions to be performed in the future
  - **BDI: goals are conjunctions of literals; KS: goals are more general implications**
  - **BDI:** uses **two languages** (modal logic specifications / procedural implementation); **KS:** uses the **same language** for specification and implementation
  - **BDI: implicit representation of time** in the implementation (modal operators in the specifications); **KS: explicit representation of time** (which allows for historical record of past observations)

## KS-agents vs. BDI

## Extensions

- **Communication & Updates:**
  - Dell'Acqua, Sadri & Toni [DST98 & 99]
  - Dell'Acqua, Nilsson & Pereira [DNP02]
- **KS agents for resource sharing (Sadri, Toni and Torroni) [STT\*]**
  - social interactions; protocols & policies
  - results on termination, "completeness"

## Argumentation-based negotiation (Sadri, Toni and Torroni)

- A model of agent which puts together
  - declarative specification
  - an operational counterpart
- Tools:
  - abductive logic programming
  - agent cycle inspired by KS-agent
  - social interaction by way of dialogues
- Scenario: negotiation for resource sharing
- Results:
  - general properties and
  - application-specific properties

## Negotiation Policies

- Policies are part of the agent *Beliefs*. They are *dialogue constraints*
- Policies can be used to decide how to reply to requests
- Example of dialogue constraint:

*tell*(  $Y, x, request(give( R, (Ts, Te) )), D, T$  )

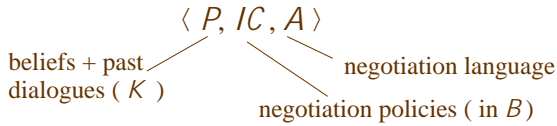
∧ *have*(  $R, (Ts, Te), T$  ) ∧ *not need*(  $R, (Ts, Te), T$  )

⇒ ∃  $T'$  | ( *tell*(  $x, Y,$

*accept*( *request*( *give*(  $R, (Ts, Te) )), D, T'$  ),  $T' > T$  )

# Mapping onto ALP

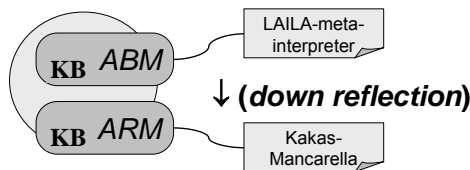
Abductive Logic Program =  
 Logic Program (  $P$  )  
 + Integrity Constraints (  $IC$  )  
 + Set of abducible predicates (  $A$  )



# Coordination of agent reasoning: the ALIAS architecture [CLMT\*]

- Problem solving in *open worlds*:
  - Incomplete knowledge
  - Multiple knowledge
- Definition and implementation of an architecture based on *multiple intelligent agents*:
  - Reasoning capabilities
  - **Coordination of reasoning** among agents
- Agent social behaviour
  - Collaborative vs. competitive

# The Architecture of an agent



- Agent Behaviour Module + KB (LAILA, Language for Abductive Logic Agents)
- Abductive Reasoning Module + KB (abductive program)

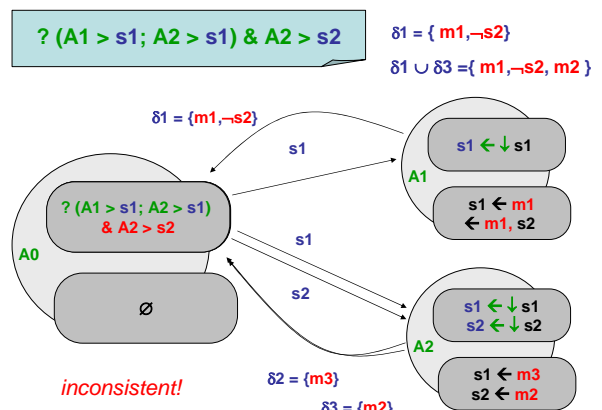
# An Example: Distributed Diagnosis

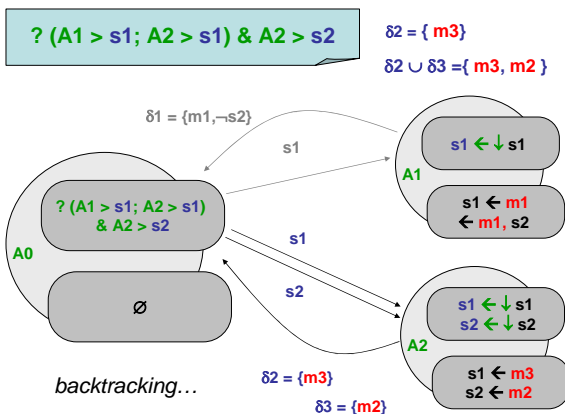
- distributed diagnosis is a popular application domain of agents
- in ALIAS, we can have diverse specialised agents with local domain knowledge (e.g. different kinds of vehicles, or parts of vehicles)
  - problem: observed symptoms  $s_1, s_2, \dots$  ( $\rightarrow$  query)
  - agents: diagnosis (abducibles  $d_1, d_2, \dots$ )
- **collaboration**: when there is a problem, agents of different areas produce explanations of the symptoms, they must be coherent with each other
- **competition**: several agents specialized in different areas (with different KBs) produce different alternative explanations of the same symptoms
- *Advantage*: maintenance of local knowledge, specialization of tasks

# An Example

•  $A_0$  :  
 $? (A_1 > s_1; A_2 > s_1) \& A_2 > s_2$

- $A_1$  and  $A_2$  represent two diagnostic agents
- $A_1$  and  $A_2$  are asked to solve  $s_1$  competitively; a  $\delta_1$  is selected
- $A_2$  is asked to solve  $s_2$ ; a  $\delta_2$  is obtained;
- $\delta_1$  and  $\delta_2$  must be consistent (possible backtracking)





## Logics for Kiga-kiku computing: Speculative Computation [SIIS00]

- Another approach to coordination of reasoning
  - Idea of "Kiga-kiku": computers *understand a situation and take an appropriate action* for the situation *without being told explicitly what to do*
  - What do we need:
    - situation-awareness
    - understanding user intention without much interaction
    - learning user's preference
    - handling incompleteness and a mechanism of back-up in the "kiga-kiku" action is not appropriate
- **Speculative computation by abduction**

## Meeting room reservation

- *A*, *B*, and *C* to attend the meeting.
- If a person is available, then he will attend the meeting.
- We ask a person whether he is free or not.
- If all the persons are available, we reserve a big room.
- If only two persons are available, we reserve a small room.

Suppose we have answers from *A* and *B* that they are free but we do not have an answer from *C*.

Then, non-"kiga-kiku" computer (or person) cannot decide a room reservation since the answers from *C* are not obtained.

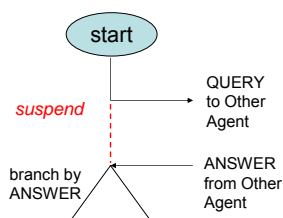
## Solution

- We can decide a room reservation based on a *plausible answer* whether *C* is usually busy or not. ("kiga-kiku" reservation)
- If the answers *C* is an *exception*, then we cancel the room and make a new reservation. (backing-up for failure of "kiga-kiku" action)

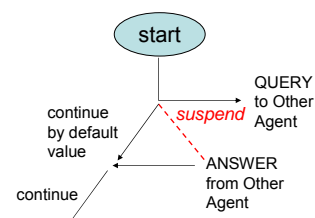
This process is called **speculative computation**.

- need to have a set of *default answers*
- *operational semantics* given in terms of process activation/suspension
- a set of *assumptions of the form (not) Q@S* is maintained

## Ordinary computation

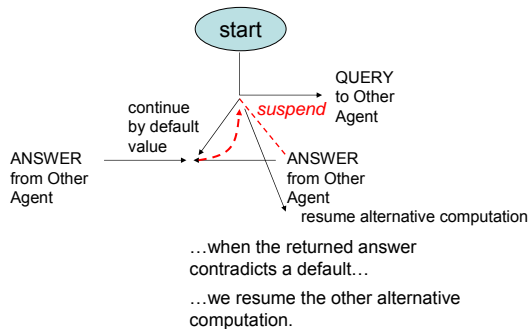


## Speculative computation



...when the returned answer is consistent with a default, we just continue the computation...

## Speculative computation



## MINERVA - A Dynamic Logic Programming based Agent Architecture [Lei03, LAP02a]

- An architecture to represent the epistemic states of agents and its evolution.
- It employs:
  - Multi-dimensional Dynamic Logic Programming (MDLP) [ALP\*\*, LAP01a, Lei03];
  - Knowledge And Behaviour Update Language (KABUL) [Lei03]

## Multi-Dimensional Dynamic Logic Programming (MDLP)

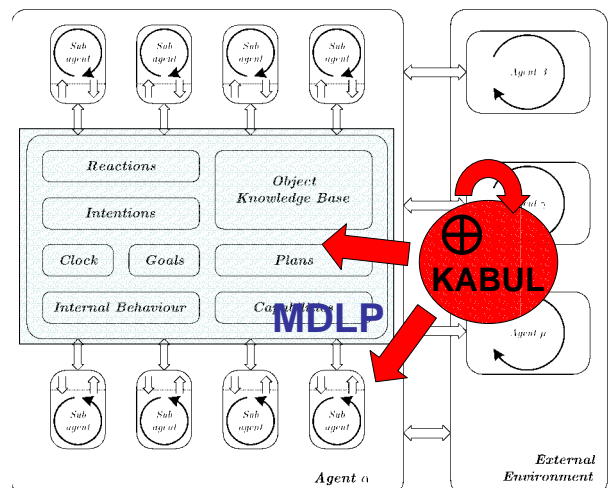
- Knowledge is given by a set of Generalized Logic Programs related according to a Directed Acyclic Graph (DAG)
- The DAG can encode several aspects e.g., temporal relations, hierarchy relations, etc. [LAP01b]
- MDLP assigns semantics to such knowledge representations
- The semantics of MDLP is a generalization of the answer-sets semantics

## KABUL

- MDLP: the declarative representation of knowledge states,
- KABUL: declarative representation of state transitions i.e. behaviours.
- A program in KABUL is a set of statements
- Statements allow the specification of updates (e.g., assertions, retractions, ...), both
  - to the MDLP (knowledge)
  - and to the KABUL program itself (behaviour), thus allowing for its own evolution.

## MINERVA – modular agent architecture

- Every agent is composed of specialized sub-agents that execute special tasks, e.g., reactivity, planning, scheduling, belief revision, action execution
- A common internal KB (one or more MDLP), concurrently manipulated by its specialized sub-agents
- The MDLPs may encode
  - object level knowledge,
  - knowledge about goals, plans, intentions, etc...
- KABUL used to encode specification and evolution of the epistemic state of each sub-agent



# Argumentation & decision making [KM02, KM03a]

- Uniform method of Decision Making via argumentation-based decision policies for:
  - “Professional” policies, related to the different agent’s capabilities (i.e. problem solving, cooperation, communication, etc.)
  - “Personality” policies, on needs and motivations
- Deliberation to be sensitive to Roles & Context

# Argumentation with Roles and Context

□ **Default Context** ↔ definition of roles  
 ■ Market: normal, regular customer

□ **Specific Context**  
 ■ High season, sales season

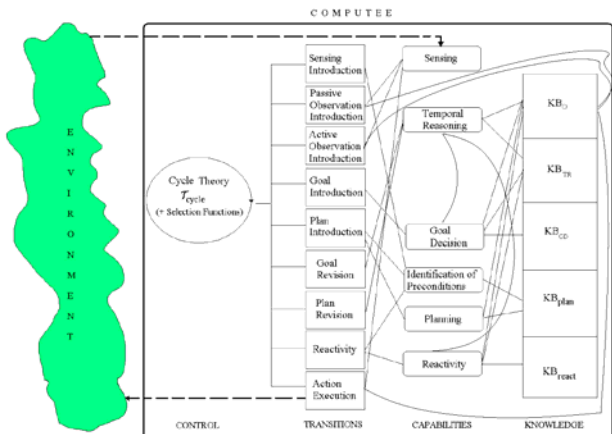
- **Example Agent theory:  $T=(T, P_R, P_C)$**
- R1:** h-p(r1(Prd, Ag), r3(Prd, Ag))
  - R2:** h-p(r3(Prd, Ag), r1(Prd, Ag)) ← regular (Ag), buy-2 (Ag, Prd)
  - R3:** h-p(r3(Prd, Ag), r1(Prd, Ag)) ← regular (Ag), late-del (Ag, Prd)
  - C1:** h-p(R1(Prd, Ag), R2(Prd, Ag)) ← high-season
  - C2:** h-p(R1(Prd, Ag), R3(Prd, Ag)) ← high-season
  - C3:** h-p(R2(Prd, Ag), R3(Prd, Ag))

# Capabilities and Personality

- The **Personality** can influence the decision making of the agent associated to his different capabilities
- **Example:** Decide within the problem solving module which requested task to perform according to his “professional” policy and his personality
  - Professional Policy
    - r1**(A, T1, A1): perform (A, T1, A1) ← ask (A1, T1, A)
    - r2**(A, T1, T2, A1): ¬perform(A, T1, A1) ← perform (A, T2, self)
    - R1:** h-p (r1(A, T1, A1), r2(A, T1, T2, A1)) ← higher-rank (A1, A)
    - R2:** h-p (r2(A, T1, T2, A1), r1(A, T1, A1)) ← competitor (A1, A)
    - C1:** h-p (R1(A, T1, T2, A1), R2(A, T1, T2, A1)) ← common-project (A, T1, A1)
    - C2:** h-p (R2(A, T1, T2, A1), R1(A, T1, T2, A1)) ← urgent (A, T2)
  - Personality Policy: The case of a selfish agent (3 = social needs; 4 = ego)
    - $R_{43}^2$ : h-p (G<sub>4</sub>, G<sub>3</sub>) ← ¬S<sub>4</sub>, ¬N<sub>3</sub>
    - $R_{34}^2$ : h-p (G<sub>3</sub>, G<sub>4</sub>) ← ¬S<sub>3</sub>, ¬N<sub>4</sub>
    - $H_{43}^2$ : h-p (R<sub>43}^2, R<sub>34}^2) ← true (basic hierarchy)</sub></sub>
    - $E_{234}^2$ : h-p (R<sub>34}^2, R<sub>43}^2) ← dangerous-for-company (G<sub>4</sub>) (exception policy)</sub></sub>
    - $C_{34}^2$ : h-p (E<sub>34}^2, H<sub>234}^2) ← true</sub></sub>

# The KGP model of agency [KMS<sup>++</sup>]

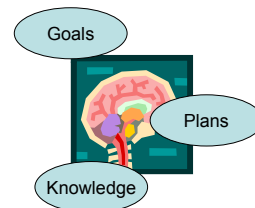
- An *internal (mental) state*;
- A set of *reasoning capabilities* for performing
  - planning,
  - temporal reasoning,
  - identification of preconditions of actions,
  - reactivity, and
  - goal decision;
- A *sensing capability*;
- A set of formal *state transition rules*;
- A set of *selection functions*;
- A *cycle theory*.



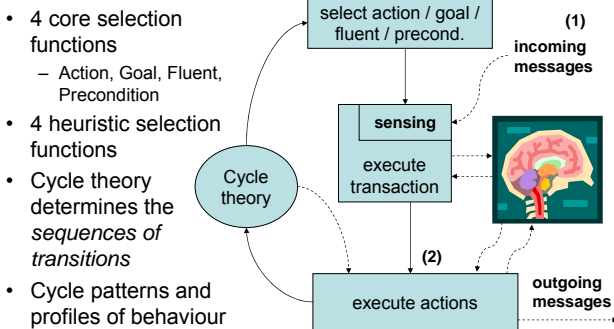
# Mental state of a computee

< KB, Goals, Plans >

- **KB** consists of different modules (KB<sub>plan</sub>, KB<sub>react</sub>, ...), supporting different capabilities
- KB<sub>0</sub> contains the *state* of the environment
- **Goals** can be mental / sensing
- **Plan** is a concrete set of (physical / communicative / sensing) actions



## Selection functions & cycle



## Goal Decision based on LPwNF

- Taken from work by Kakas and Moraitis
- $KB_{GD}$  is composed of two main parts:
  - lower-level: rules that generate goals
 
$$L \leftarrow L_1, \dots, L_n \quad (0 \leq n)$$
 where  $L_1, \dots, L_n$  are either time-dependent conditions of the form *holds-at* (l,t), or time-independent conditions or temporal constraints
  - higher-level: rules that specify priorities between other rules of the theory
 
$$h-p(\text{rule1}, \text{rule2}) \leftarrow L_1, \dots, L_n, Tc$$
- rule1*, *rule2* are names of other rules in  $KB_{GD}$
- $KB_{GD} \cup KB_{TR}$  is used to evaluate the conditions

## Cycle theories

- A cycle theory is a logic program  $T_{cycle}$  with priorities reasoning on the whole state of the computee (meta-program)
- 4 components:
  - $T_{initial}$  (initialization), containing rules of the form:
 
$$r_{0|k}(S_0, X) : T_k(S_0, X) \leftarrow T_k(S_0, \tau, X)$$
  - a *basic* part  $T_{basic}$  (basic steps of iteration), of rules:
 
$$r_{i|k}(S', X') : T_k(S', X') \leftarrow T_i(S, X, S'), C_{i|k}(S', \tau, X')$$
  - an *interrupt* part  $T_{interrupt}$  (cycle steps that can follow a POI):
 
$$r_{POI|k}(S', X) : T_k(S', X) \leftarrow T_{POI}(S, S'), C_{POI|k}(S', \tau, X)$$
  - a *behaviour* part  $T_{behaviour}$  (computee's characteristics):
 
$$R_{k|l} : h-p(r_{i|k}(S, X_k), r_{i|k}(S, X_l)) \leftarrow BC_{k|l}(S, X_k, X_l, \tau)$$

## Cycle theories

- Example:

$$R_{POI|AE}^{POI} : h-p(r_{POI|AE}(S, As'), r_{POI|*}(S)) \leftarrow h^u_{AS}(S, \tau) = As', As' \neq \emptyset, \text{very-urgent}(As', \tau)$$

meaning that : we do not want to carry out any of the interrupt cycle-steps at the expense of delaying the execution of very urgent actions

## Cycle theory as control

- The cycle theory of an agent provides a form of *declarative and flexible control*
- Cycle theories with special features (namely inducing a total ordering on the transitions) give a more conventional *fixed control*, namely whose operational trace is given by
 
$$T_1, \dots, T_n, T_1, \dots, T_n, T_1, \dots, T_n, \dots$$
 (e.g. Plan, Execute, Observe, React, Plan...)
- Control via cycle theories can in principle be adopted via any agent architecture

## Agents in logic

- Agent architectures
  - ✓ BDI
  - ✓ Agent-0
  - ✓ KS-agents, STT, ...
  - ✓ ALIAS
  - ✓ Speculative Computation
  - ✓ MINERVA
  - ✓ Kakas et al., KGP, ...
    - 3APL
    - IMPACT
    - Linear Logics
    - MetateM
    - ConGolog
    - Dylog
    - ...
- Agent interaction
  - ✓ KQML, FIPA
  - ✓ Yolum & Singh / Fornara, Colombetti & Verdicchio
  - ✓ SOCS social infrastructure
    - norms / institutions (deontic logic-based approaches)



## 3APL: a combination of declarative and imperative programming

- Agent programming language, **primarily concerned with the dynamics of an agent's mental life** [HBHM99a]
  - representation of beliefs
  - belief updating
  - goal updating, to facilitate practical reasoning
  - no communication/social aspects involved
- Beliefs: entailment relation for 1<sup>st</sup> ord. logic,  $\models$ , and CWA
- Goals: *procedural* notion (goals-to-do). Basic actions affect the *mental state* of the agent
- Basic goals: basic actions, achievement goals, test goals
- (static) **practical reasoning rules**:
  - to *build* a plan library
  - to revise and monitor goals of the agent

## Practical reasoning rules

- The set Rule of p.r.r. is defined by:
    - $\pi_h \leftarrow \varphi \mid \pi_b \in \text{Rule}$  s.t. any goal variable X occurring in  $\pi_b$  also occurs in  $\pi_h$  ( $\varphi = \text{guard}$ )
    - $\leftarrow \varphi \mid \pi_b \in \text{Rule}$  s.t. no goal variables occur in  $\pi_b$ , i.e.,  $\pi_b \in \text{Goal}$
    - $\pi_h \leftarrow \varphi \in \text{Rule}$
- $\pi_h \leftarrow \varphi \mid \pi_b$  states that if the agent has adopted some goal or plan  $\pi_h$  and believes that  $\varphi$  is the case, then it may consider adopting  $\pi_b$  as a new goal.
- (sub-)goals in the head are replaced by those body, when the guard is believed true.

## Programming agents in 3APL

- Agent defined as a tuple**:  $\langle \Pi, \sigma, \Gamma \rangle$  (goals, beliefs, p.r.r.)
- Classification of rules (*ordering*)
  - Failure rules (highest priority)
  - Reactive rules
  - Plan rules
  - Optimization rules (lowest priority)
- Selection mechanisms (to reduce the non-determinism of the language): **a meta-language for programming control structures**.
- Basic actions:
  - rule selection, **selap** (r,g,R,G)
  - application of a number of rules, **apply** (R,G,G')
  - goal selection, **selex** (g,G)
  - execution of a set of goals, **ex** (G,G')

## Update-Act cycle

- Select a rule R to fire
- Update goal base by firing R
- Select a goal G
- Execute (part of) G
- Goto 1

```

while  $\Pi \neq \emptyset$  do
  begin
    selap ( $R \cup P \cup O, \Pi, R, G$ )
    apply (R, G, _)
  repeat
    selex ( $\Pi, G$ )
    selap ( $F, G, R, \_$ )
    apply (R, G, _)
  until R =  $\emptyset$ 
  ex (G, _)
end
    
```

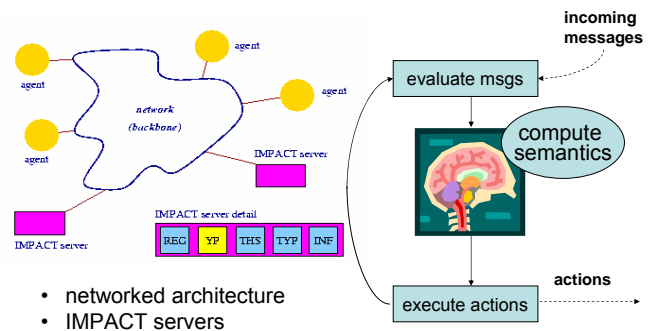
*F, R, P, and O are the 4 sets of rules*

Labels: **planning** (lines 2-3), **filtering** (lines 4-5), **execution** (lines 6-7)

## IMPACT: the Interactive Maryland Platform for Agents Collaborating Together [SBD<sup>+</sup>00]

- Motivations:
  - Agentize arbitrary Legacy Code (à la Shoham)
  - Code-calls to access distributed and heterogeneous knowledge
  - Clear semantics to agent activity
- Data access
- Architecture
- Programs
- Semantics based on *deontic operators*

## IMPACT Architecture & Cycle



## Agent Program

- A program  $P$  is a Set of rules of the form  

$$\text{Op } a(\text{arg1}, \dots, \text{argn}) \leftarrow \text{<code call condition>} \ \& \ \text{Op}_1 a_1(\text{<args>}) \ \& \ \dots \ \& \ \text{Op}_n a_n(\text{<args>})$$
- $\text{Op}$  is a “deontic modality” and is either
  - $P$  - permitted
  - $F$  - forbidden
  - $O$  - obligatory
  - $W$  - waived
  - $Do$  - execute
- If **code call condition** is true and the deontic modalities in the rule body are true, then  $\text{Op } a(\text{arg1}, \dots, \text{argn})$  is true.

## Example: select driving lane

```

O(go-rightmost) ←
O(drive(r-lane)) ← Do(go-rightmost),
                    in(r-lane, status.free-lanes())

F(drive(L)) ← not-in(L, status.free-lanes())

Do(drive(l-lane)) ← F(drive(r-lane)),
                    in(l-lane, status.free-lanes())
  
```

## Semantics: *Status Set*

- A *status set* is a collection of ground action status atoms  $\text{Op } \alpha$ . Status set  $S$  is **feasible** on an agent state, if
  1.  $S$  is closed under rules of  $P$ ,
  2.  $S$  satisfies deontic and action consistency,
  3.  $S$  is **deontically and action closed**,
  4. executing  $\text{Do}(S) = \{ \alpha \mid \text{Do}(\alpha) \in S \}$  leads to a consistent new state.
- Example (both lanes free):  

$$S = \{ O(\text{go-rightmost}), Do(\text{go-rightmost}), P(\text{go-rightmost}), Do(\text{drive}(r\text{-lane})), P(\text{drive}(r\text{-lane})) \}$$

## Deontic Logic for Agents and for Societies

- IMPACT uses deontic concepts to define feasible status sets: stress on the behaviour of *the individual agent*
- Deontic logic used to represent and reason about rules and norms in a society
- **ALFEBIITE** project
  - Legal Aspects of Inter-Agent Communication
  - Open societies
  - Formal approach to trust

## Linear Logics

- Logic of *occurrences*: two copies of a formula are not equivalent to one copy of it !!
- Suitable for agents because resources are usually *bounded*
- Two forms of conjunction:
  - cumulative:  $p \otimes p$  *not equivalent to*  $p$
  - not cumulative:  $p \& p \equiv p$
- Potentially infinite amount of a resource:  $!p$ 
  - *classical reasoning: formulae beginning by !*
- Classical disjunction:  $p \oplus q$
- Negation:  $F^\perp$
- Derivation (and consumption):  $p \multimap q$ 
  - once the formula is used,  $p$  is no longer available, but  $q$  is

## Agents in Linear Logic

- Harland & Winikoff [HW02,THH03]
- Küngas & Matskin [KM03b,KM04]
- Mascardi *et al.* [MMZI], using Delzanno’s  $E_{hhf}$ 
  - linear logic language that can be used to specify
    - concrete agent architectures,
    - agent program and
    - state.
  - Specifications in  $E_{hhf}$  [Del97] are executable, so it is possible to directly interpret the given specification.
  - $E_{hhf}$  can be used to characterize an agent architecture under a semantic point of view.

## Designing Agent Programming Systems using Linear Logics (Harland & Winikoff)

- An agent can be represented by the sequent:  

$$E, A, B, !P \vdash G$$
 where
- $B$  are the *beliefs* of the agent (which are linear since they change)
- $P$  are the *program clauses* (i.e. goal-plan decompositions)
- $G$  are the agent's *goals* (intentions)
- $E$  are *events*
- $A$  are *actions*

## Agent programs in LL

$P$  can include *action descriptions*:

“to achieve (have-lemonade)  
 try do (open-fridge) then do (get-lemonade)”

!( do (get-lemonade)  $\otimes$  fridge (open)  $\multimap$   
fridge (open)  $\otimes$  have-lemonade )  
 !( do (open-fridge)  $\otimes$  fridge (closed)  $\multimap$   
fridge (open) )

...and in case of *failure*:

!( do (get-lemonade)  $\otimes$  fridge (closed)  $\multimap$   
fridge (closed)  $\otimes$  fails (get-lemonade) )

## Linear Logic for rapid prototyping (Mascardi et al.)

- **CaseLP** (Complex Application Specification Environment based on Logic Programming) is:
  - a set of tools for the *specification* of MAS,
  - a set of tools for describing the *behaviour* of agents,
  - a set of tools for the *integration* of legacy systems/data
  - a set of simulation tools for *animation* of MAS
- $E_{hhf}$  is the language used for high-level specification of MAS

## Other approaches using other logics

- Many other approaches in literature!!
  - Temporal Logic – Concurrent MetateM (Fisher)
  - Situation Calculus – ConGolog (De Giacomo, Lespérance, Levesque)
  - Dynamic Logic – DyLOG (Patti)
  - BOLD (Dastani et al.)
  - ...

## Wrap-up

- Given insights on (pointers to)
  - application domains of multi-agent systems
  - basic categories of agent programming
  - logic based construct to model
    - internals (state + thinking)
    - interactions
- Identified links and similarities in the literature
- Logic useful for
  - modelling & specification
  - operational model  $\Rightarrow$  implementation/prototyping
  - identification and verification of properties

## What properties

- Properties are important!
- “Classical” properties of computational systems (e.g., termination, absence of inconsistency, ...)
- “Classical” properties of distributed systems (e.g., robustness, modularity, scalability, openness, ...)
- Properties related to interaction
  - conformance to protocols / social norms
  - competent use of protocols
  - properties of interaction mechanisms
- Properties of “agency”
  - social attitudes (altruistic, selfish, malicious, rational..)
  - profiles of individual behaviour (impatient, focussed, risk-averse, ...)

## Final remarks

- Computational logic used to tackle several different aspects of agent-based programming
- Important link from specification to implementation (including verification): Theory and practice can work together
- Need to make tools understood and accessible by industry (connection with standards, mapping onto existing formalisms)
  - Watch the SOCS website
  - Submit your work to CLIMA-V by June 22<sup>nd</sup>
    - attend DALT2004

## Pointers

- Web sites:
  - AgentLink II: <http://www.agentlink.org>
  - UMBC Agent WEB: <http://agents.umbc.edu/>
  - Agent Based Systems: <http://www.agentbase.com/survey.html>
  - Agent Construction Tools: <http://www.agentbuilder.com/AgentTools/>
- Journals
  - Journal of Autonomous Agents and Multi-Agent Systems
- Conferences and Workshops
  - International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) – next in New York, deadline: 16 January 2004
  - Past events: ATAL, ICMAS, AA and related WS (LNAI, IEEE, and ACM Press)

CILC 2004 Tutorial

Parma, 16 Giu 2004

127

## Pointers

- Journals
  - Artificial Intelligence
  - Journal of Logic and Computation
  - Annals of Mathematics and Artificial Intelligence
  - The Knowledge Engineering Review
  - Journal of Group Decision and Negotiation
  - Theory and Practice of Logic Programming
  - Journal of Cooperative Information Systems
- Conferences and Workshops
  - Workshop on **Computational Logics in Multi-Agent Systems (CLIMA)** – next in Lisbon, Sep. 29-30, 2004  
**deadline: next week!!**
  - **Declarative Agent Languages and Technologies (DALT)** – New York, July 19<sup>th</sup>, 2004, together with AAMAS'04

CILC 2004 Tutorial

Parma, 16 Giu 2004

128

## Research groups & projects

- SOCS, EU Project, <http://lia.deis.unibo.it/research/socs>
- MASSIVE, MIUR Project, <http://www.di.unito.it/massive>
- DyLOG, DI, Università di Torino, <http://www.di.unito.it/~alice/>
- CaseLP, DISI, Università di Genova, <http://www.disi.unige.it/index.php?research/ai-mas>
- ALIAS, DEIS, Università di Bologna, <http://lia.deis.unibo.it/research/ALIAS/>
- ALFEBIITE, EU Project, <http://www.iis.ee.ic.ac.uk/~alfebiite/>
- 3APL: Intelligent Systems Group, University of Utrecht, <http://www.cs.uu.nl/groups/IS/agents/agents.html>
- xGOLoG: Cognitive Robotics Group, University of Toronto, <http://www.cs.toronto.edu/cogrobot/>
- IMPACT: University of Maryland, <http://www.cs.umd.edu/projects/impact/>
- MetateM: Logic and Computation Group, University of Liverpool, <http://www.csc.liv.ac.uk/~michael/>
- DESIRE: <http://www.cs.vu.nl/vakgroepen/ai/projects/desire/>
- JACK: The Agent Oriented Software Group, <http://www.agent-software.com/>
- BOID: <http://boid.info/>
- RMIT: <http://www.cs.rmit.edu.au/agents/>
- Dagstuhl seminar 02481 on logic based MAS: <http://www.cs.man.ac.uk/~zhangy/dagstuhl/>

CILC 2004 Tutorial

Parma, 16 Giu 2004

129

## Pointers

- Surveys on multi-agent systems
  - [JSW98] N. Jennings, K. Sycara, and M. Wooldridge, *A Roadmap of Agent Research and Development*. AAMASJ 1998.
  - [WC00] M. Wooldridge and P. Ciancarini, *Agent-Oriented Software Engineering: The State of the Art*. In Proc. First Int. Workshop on Agent-Oriented Software Engineering, LNCS, 2000
  - [LMP03] M. Luck, P. McBurney, C. Preist, *Agent Technology Roadmap. 2003*. Available electronically <http://www.agentlink.org/roadmap/>
- Books
  - [Wei99] G. Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999
  - [Woo02] M. Wooldridge, *Introduction to Multi-Agent Systems*. John Wiley & Sons, 2002.

CILC 2004 Tutorial

Parma, 16 Giu 2004

130

## Pointers

- Some surveys on logic-based multi-agent systems
  - [ST99] F. Sadri and F. Toni, *Computational Logic and Multi-Agent Systems: a roadmap*. COMPULOG (1999), electronic version available at [http://www2.ags.uni-sb.de/net/Forum/CL\\_and\\_MAS.ps](http://www2.ags.uni-sb.de/net/Forum/CL_and_MAS.ps)
  - [Hoe01] W. van der Hoek, *Logical Foundations of Agent-Based Computing*. In Multi-Agent Systems and Applications, LNAI 2086, pp. 50-73 (2001)
  - [MMS] M. Martelli, V. Mascardi, and L. Sterling, *Logic-Based Specification Languages for Intelligent Software Agents*. To appear in TPLP. Electronic version available via [ftp://ftp.disi.unige.it/pub/person/MascardiV/Papers/\(TPLP03.ps.gz\)](ftp://ftp.disi.unige.it/pub/person/MascardiV/Papers/(TPLP03.ps.gz))

CILC 2004 Tutorial

Parma, 16 Giu 2004

131

# Pointers

- SOCS home page:  
[SOC] <http://ia.deis.unibo.it/research/socs/>
- Publications:
  - **SOCS deliverables** (contact me)
  - Conferences: ECAI'03, AAMAS'04, IJCAI'03, AI\*IA'03, CEEMAS'03, AAMAS'03, JELIA'02, UKMAS'02, ...
  - Workshops: DALT'04, TAPOCS'04, AT2AI-4, ACM SAC2004, CLIMA-IV, DALT'03, CLIMA'02, ESAW'03, LCMAS'03, FAMAS'03, MFI'03, PSE'03, ...