Distributed Artificial Intelligence and Intelligent Agents

MAS Architectures

Mihhail Matskin:

Multi-Agent Architectures

1. Low-level architectures
   - Blackboards
   - ACTORs
2. Infrastructures that use Middle Agents: RETSINA
3. Infrastructures that use ”wrappers”: ARCHON
4. Market-based Architectures
   - MAGNET
   - AGORA?

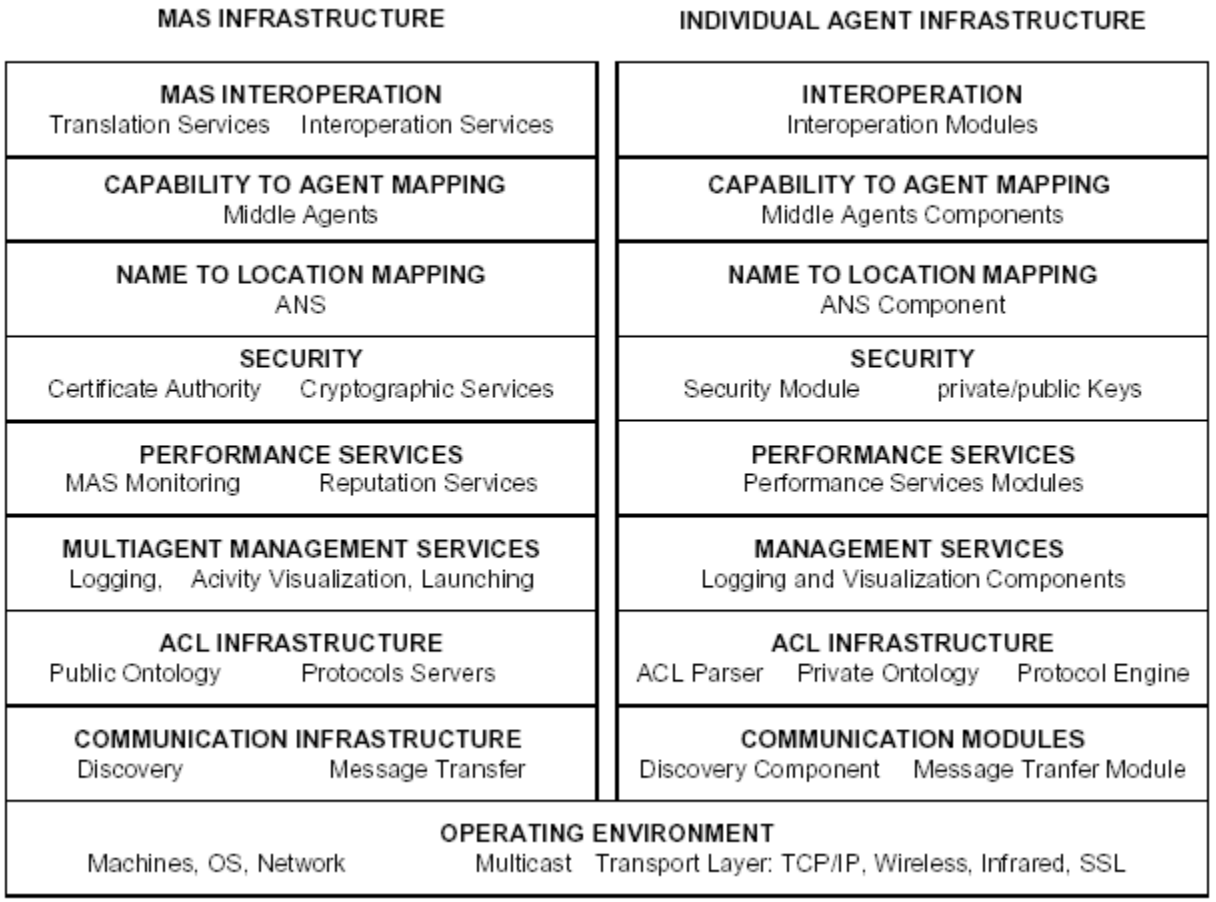
What is Multi-agent Architecture

- The infrastructure for a MAS is a set of services, conventions and knowledge that support complex social interactions (e.g. negotiations, agree on commitments).

K. Sycara

- Agents need services to:
  - Enable them to find each other in open environments
  - Communicate
  - Warrant that proper security constraints are satisfied.
  - …

Agents in MAS



Ref: Sycara et. al., 2001

Blackboard Architecture

Metaphor: A collection of intelligent agents gather around a blackboard, look at pieces of information written on it, think about them, and add their conclusions.

Some basic assumptions:

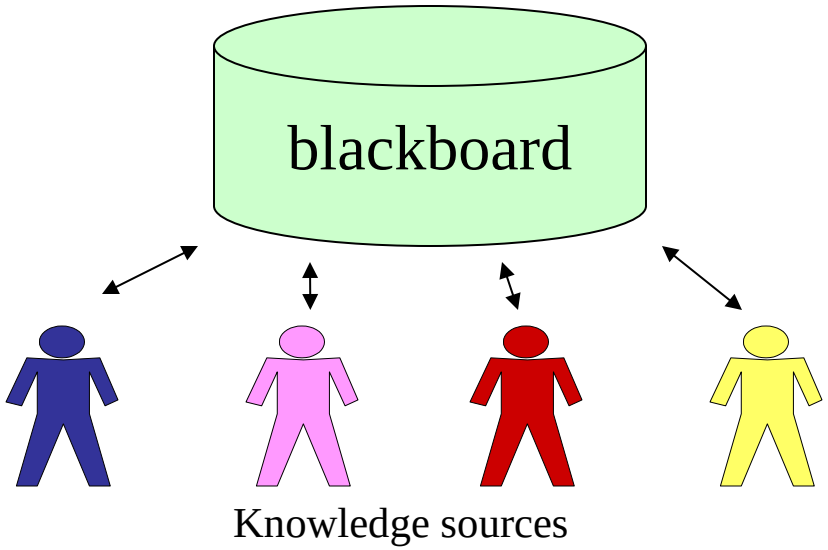
- all of the agents can see all of the blackboards all the time, and what they see represents the current state of solution
- any agent can write his conclusions on the blackboard at anytime without getting in anyone else's way
- the act of an agent writing on the blackboard will not confuse any other agents as they work

Blackboard Architecture

- Key ideas: problem solving should be
  - **Incremental** – complete solutions are constructed piece by piece, first hypothesizing a partial solution based on incomplete data and then attempting to verify additional data to verify hypothesis.
  - **Opportunistic** – the system chooses the actions to take next that it determines will allow it to make the best progress towards meeting its goals in the current situation.

Blackboard Architecture

- Main components:
  - A **blackboard** – a global database containing data and hypotheses (potential partial solutions)
  - A set of **Knowledge Sources** (KS), or agents
  - A **control mechanism**



Blackboard Architecture
The Control Problem

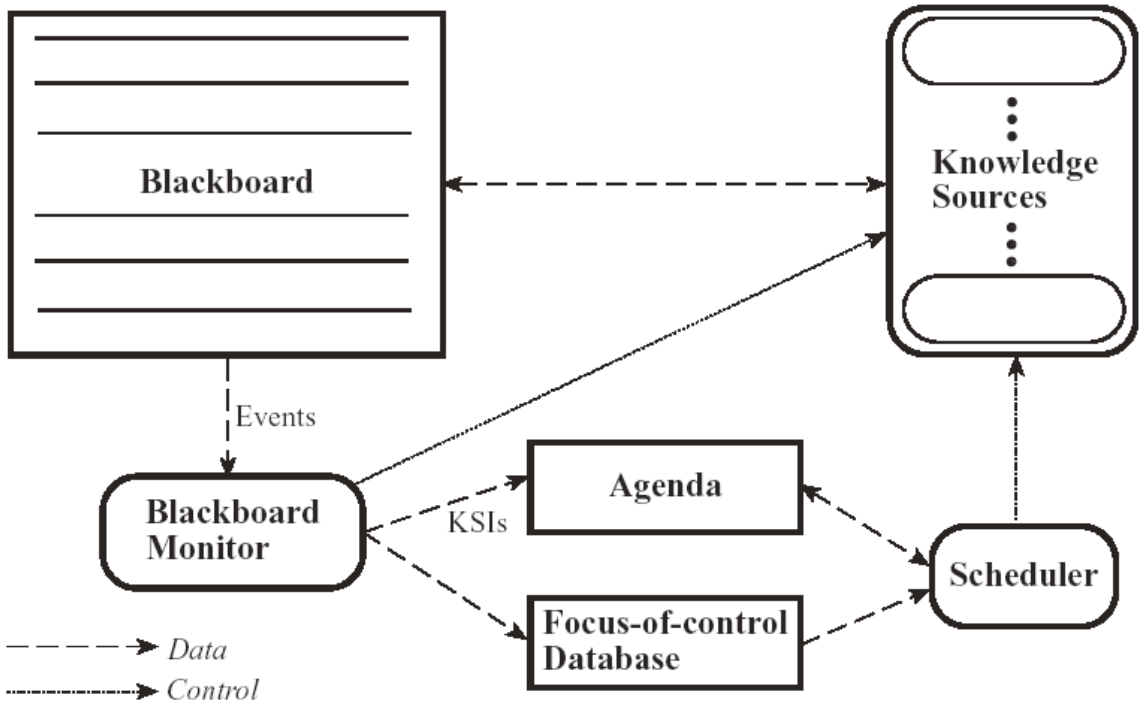
- **Control problem**: the problem of selecting the "best" action to execute next.
- Blackboard systems must have some **control mechanism**:
  - Effective control requires to take into account:
    - Goal-directed factors: based on what an agent wants
    - Data-directed factors: based on what an agent is best able to do
  - Blackboard control is difficult because it can be difficult to determine the expected value of an action by an agent as there may be complex interrelationships among the agents.

Blackboard Architecture
Serialisation Hypothesis

- Agents are schedulable entities and only one can be running at any time.
- The control mechanism selects only the most productive agent at any given moment to work on the problem.
- The blackboard is not globally visible. Agents generally work on a limited area of the blackboard, known as the agent's context.
- Implicit assumption: a knowledge source/agent operates within a valid, or consistent context.

Blackboard Architecture

Agenda-based Control



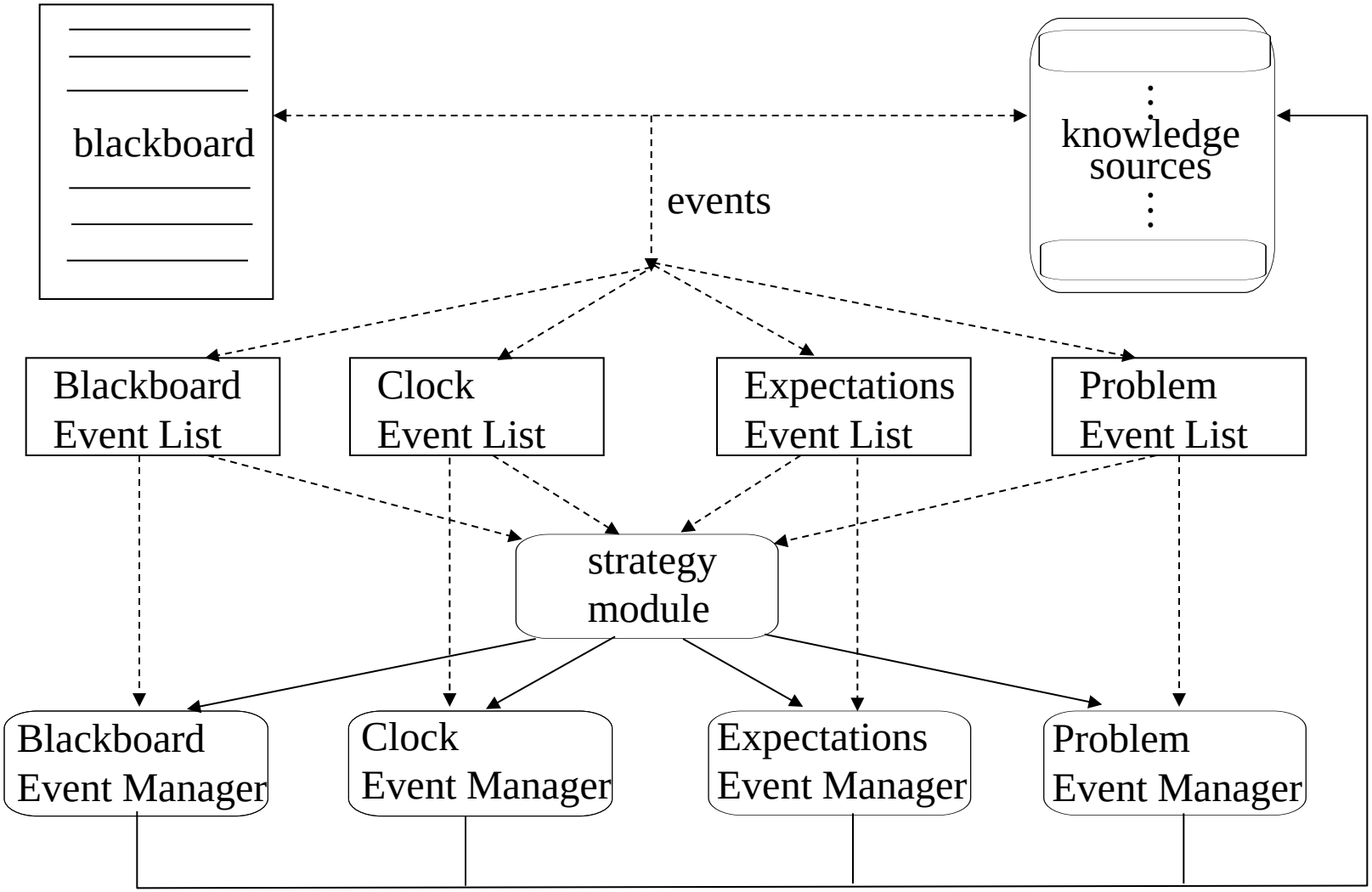
Blackboard Architecture

Agenda-based Control

- In the basic agenda-based blackboard architecture, all the control (strategy) knowledge of the system is represented in a single scheduler rating function.
- This makes it difficult to encode and modify complex control strategies.
- The knowledge and reasoning are not explicit.

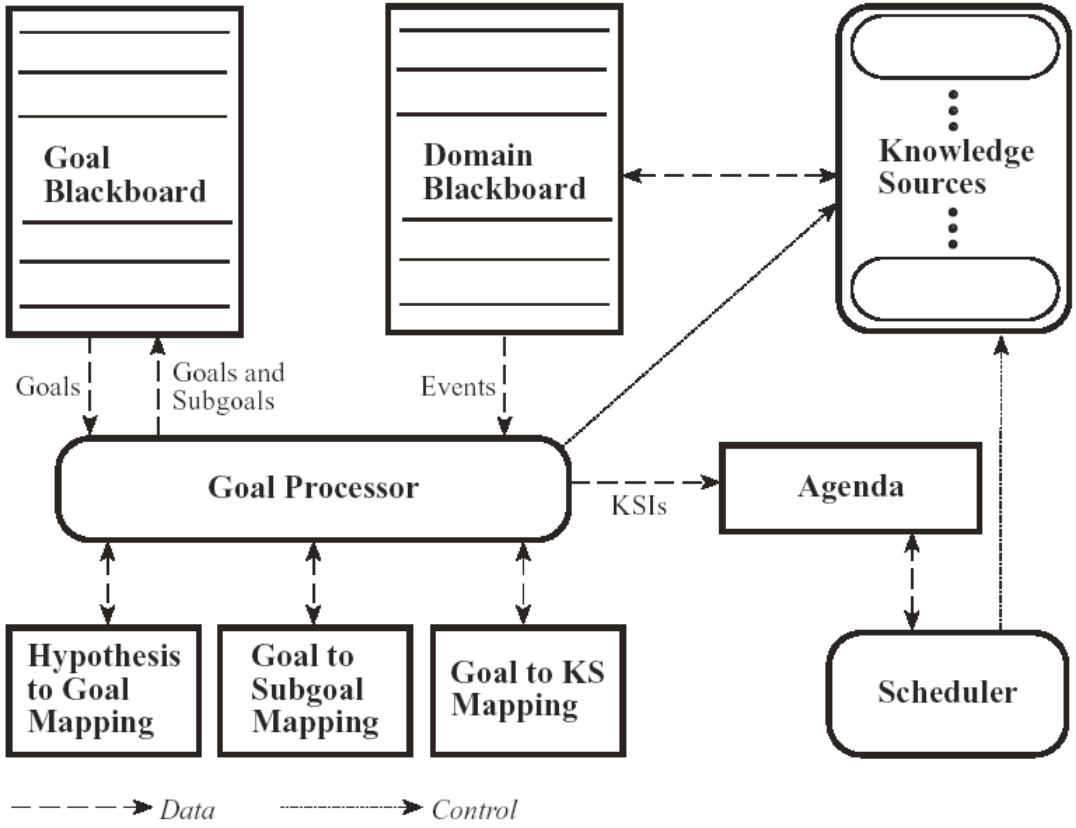
Blackboard Architecture

Event-based Control



Blackboard Architecture
Goal-directed Control

- Control that considers the role and the ultimate value of actions in satisfying the system's goals.



Blackboard Architecture

Goal-directed Control contd.

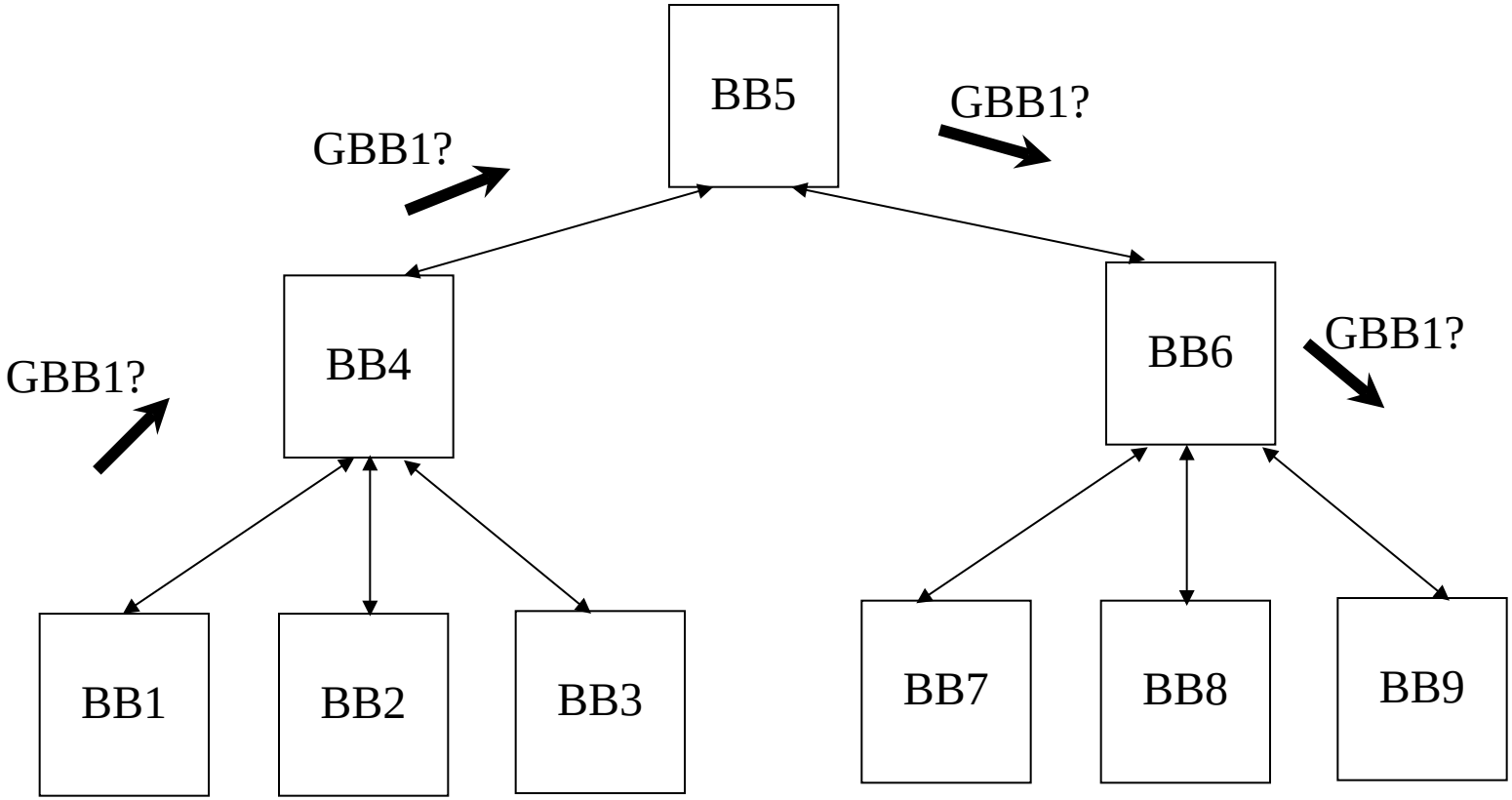
- Goal processor instantiates goals on the goal blackboard.
- Goal processor is driven by the 3 mapping functions.
- Integrates goal-directed and data-directed factors:
  - Goal-directed goals are created in response to the creation of other goals based on the goal-to-subgoal map.
  - Data-directed goals are created in response to the creation or modification of hypotheses on the blackboard based on the hypothesis-to-goal map.

Blackboard Architecture

Goal-directed Control contd.

- When a goal is inserted onto the goal blackboard, it may trigger KSs that can achieve the goal – identified by the goal-to-KS map.
- If KS is likely to generate a hypothesis to achieve the goal, KS is added to the agenda.
- Possible problem: subgoaling needs to be carefully controlled.

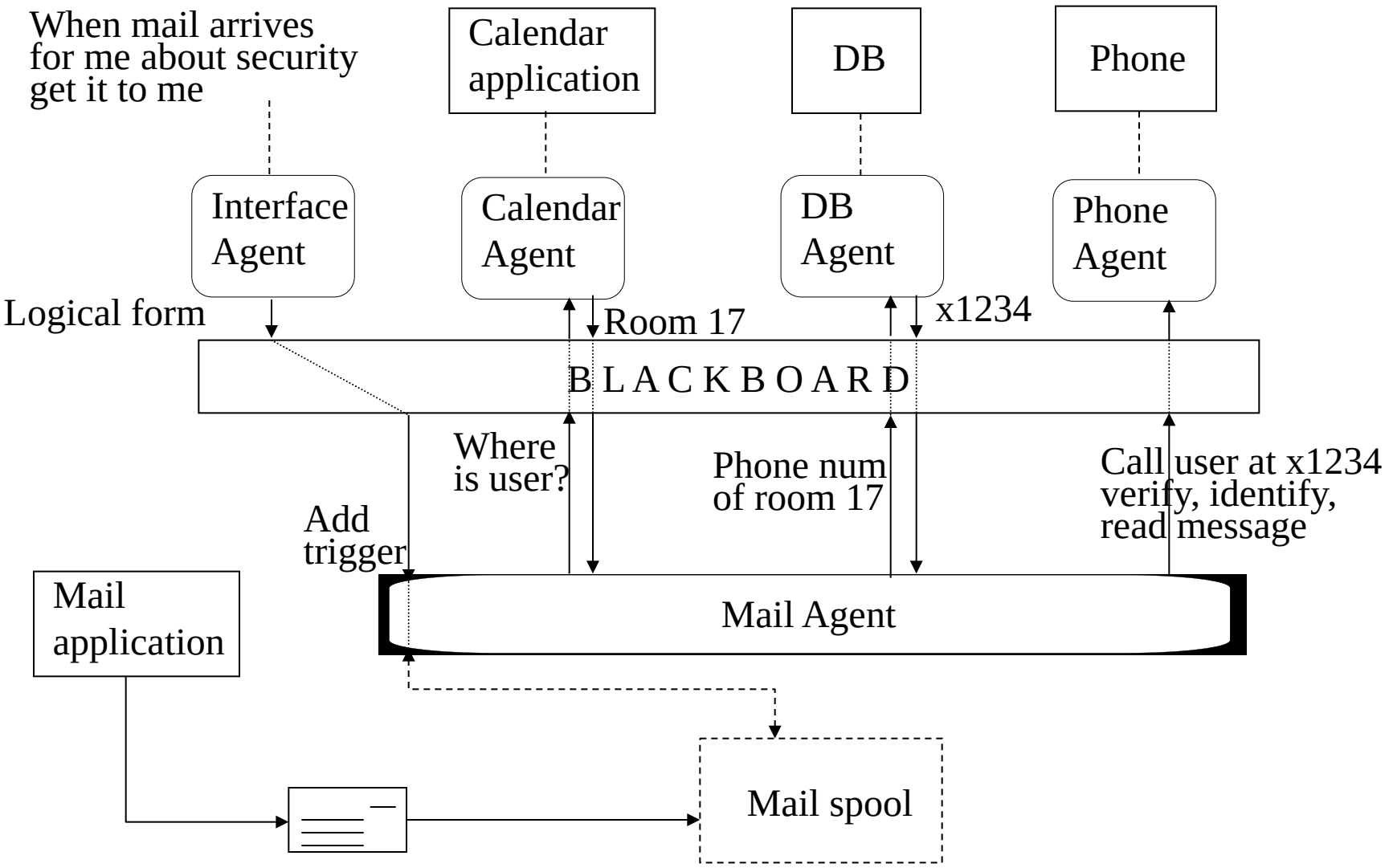
Hierarchy of Blackboard Servers
(Cohen, Cheyer, Wang, Baeg)



Operational agents (example)

- A User-interface agent
- A Database agent
- A Calendar agent
- A Mail agent
- A News agent
- A Telephone agent

Example of agent interaction via blackboard



The ACTOR Model (Hewitt)

What is an ACTOR

- An ACTOR is a computational system with the following properties:
  - Social – can send messages to other ACTORs.
  - Reactive – carry out computation in response to a message received from another actor.
    - Message driven!
- In the ACTOR model, computation itself is viewed as **message passing**.
- It can be considered as consisting of:
  - An address
  - A behavior which specifies what the ACTOR will do upon receipt of a message.

The ACTOR Model

What is an ACTOR

The Actor approach is formulated around three main design objectives:

- Shared, mutable data
- Reconfigurability
- Inherent concurrency

The ACTOR Model

What is an ACTOR

An Actor is an object that carries out its actions in response to communication it receives

Actor may perform three basic actions:

- Sending messages to itself or other ACTORs.
- Creating more ACTORs.
- Specify a replacement behavior, which is essentially another actor that takes the place of the actor that creates it, for the purpose of responding to certain communications

The ACTOR Model
Behaviour

- How an ACTOR works:
  - Upon receipt of a message, the message is matched against the ACTOR's behavior (script)
  - Upon a match, the corresponding action is executed, which may involve:
    - Sending more messages
    - Creating more ACTORs
    - Replacing the ACTOR by another

The ACTOR Model

Behavior

- Actor computation is reactive
- An Actor is dormant until it receives communication
- in any computation, each actor receives a linearly ordered sequence of computations
- messages are not guaranteed to arrive in the order in which they are sent
- there is no assignment to local variables in basic actor model
- every communication must be sent to a mail address - mail system is important part of the model

The ACTOR Model
Communication

A communication event is called a task and it has 3 parts:

- A unique tag, distinguishing it from tasks in the system
- A target, the mail address of intended receiver
- A communication which is the data passed by this communication event

The ACTOR Model
Communication

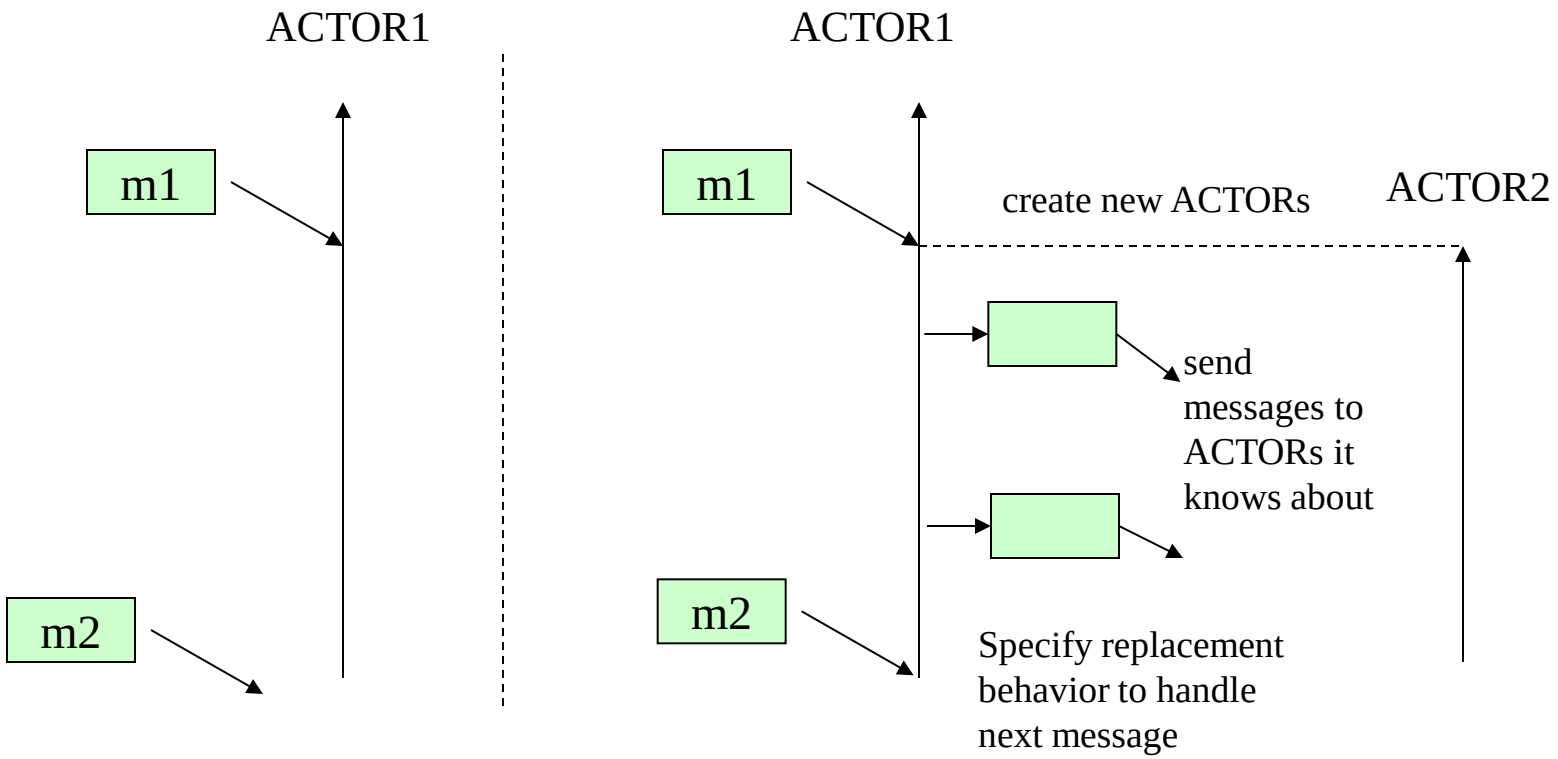
- Actor model is asynchronous
- communication is
  - explicit, through mail addresses, without shared variables
  - buffered and asynchronous
- weak fairness is assumed:
  - every message which is sent is eventually delivered
  - every computation eventually progress (no starvation)

The ACTOR Model

Behavior contd.

When an ACTOR receives a message:

In response, it can concurrently:



Message sending based architecture

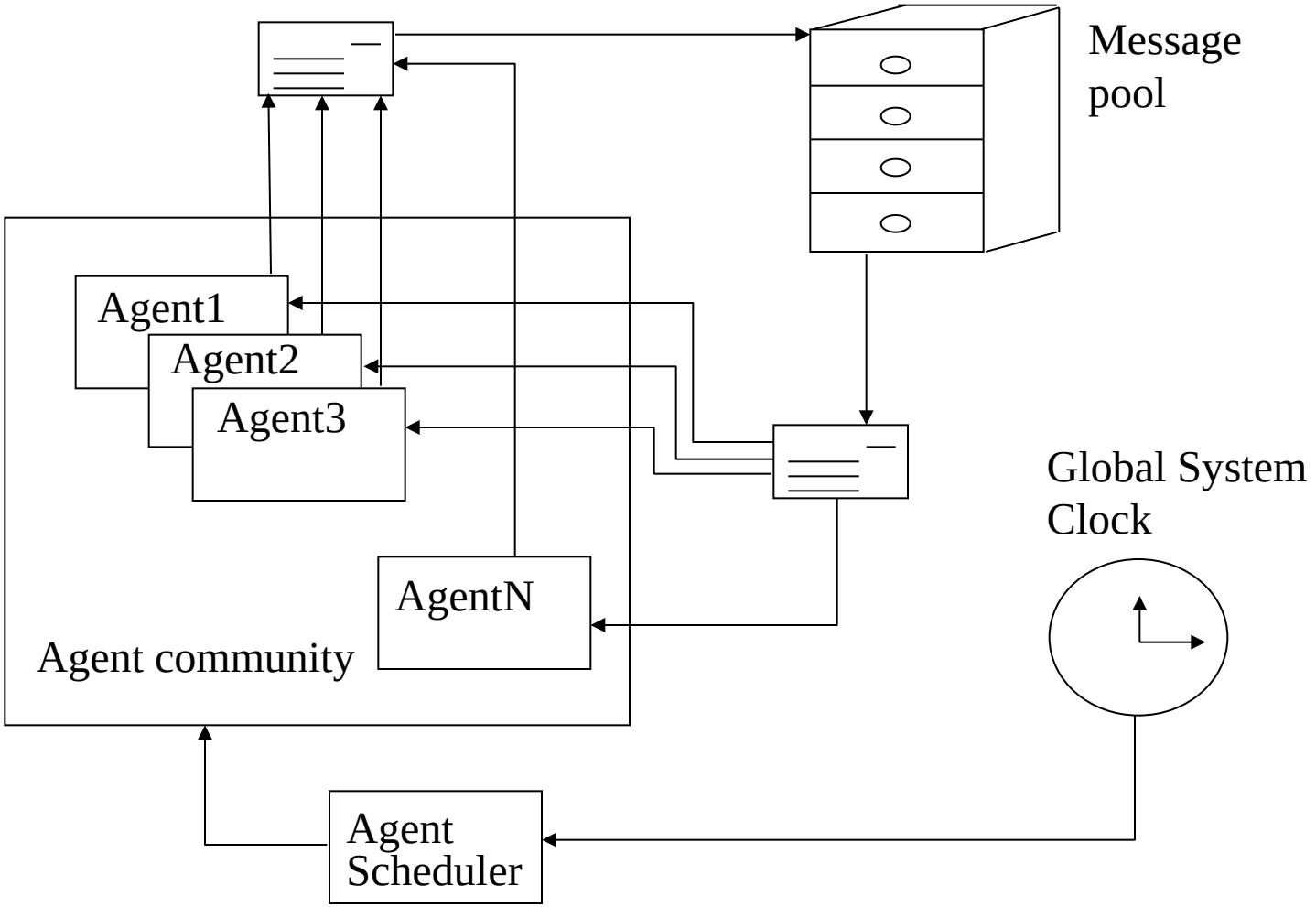
Agent Factory (O'Hare)

A few words about basic principles of Agent Factory:

- Linear discrete model of time which can be visualized as a sequence of numbers
- A guaranteed delivery assumption. Messages are delivered correctly – content of message doesn't change in transaction, message is sent to destinations and only to them
- For any message there is only one sender
- In spite of process synchronization – communication is asynchronous via message pool

Message sending based architecture

Agent Factory (O'Hare)



Architectures that provide assistance for interaction between sub-components

- In many industrial applications, a substantial amount of time, effort and money was spent on developing complex and sophisticated software systems, e.g. expert systems, databases.
- Can these (sub-)systems be integrated into a coherent community in which they work together to better meet the needs of the entire application?

ARCHON

ARchitecture for Cooperative Heterogeneous ON-line systems

(Cockburn & Jennings)

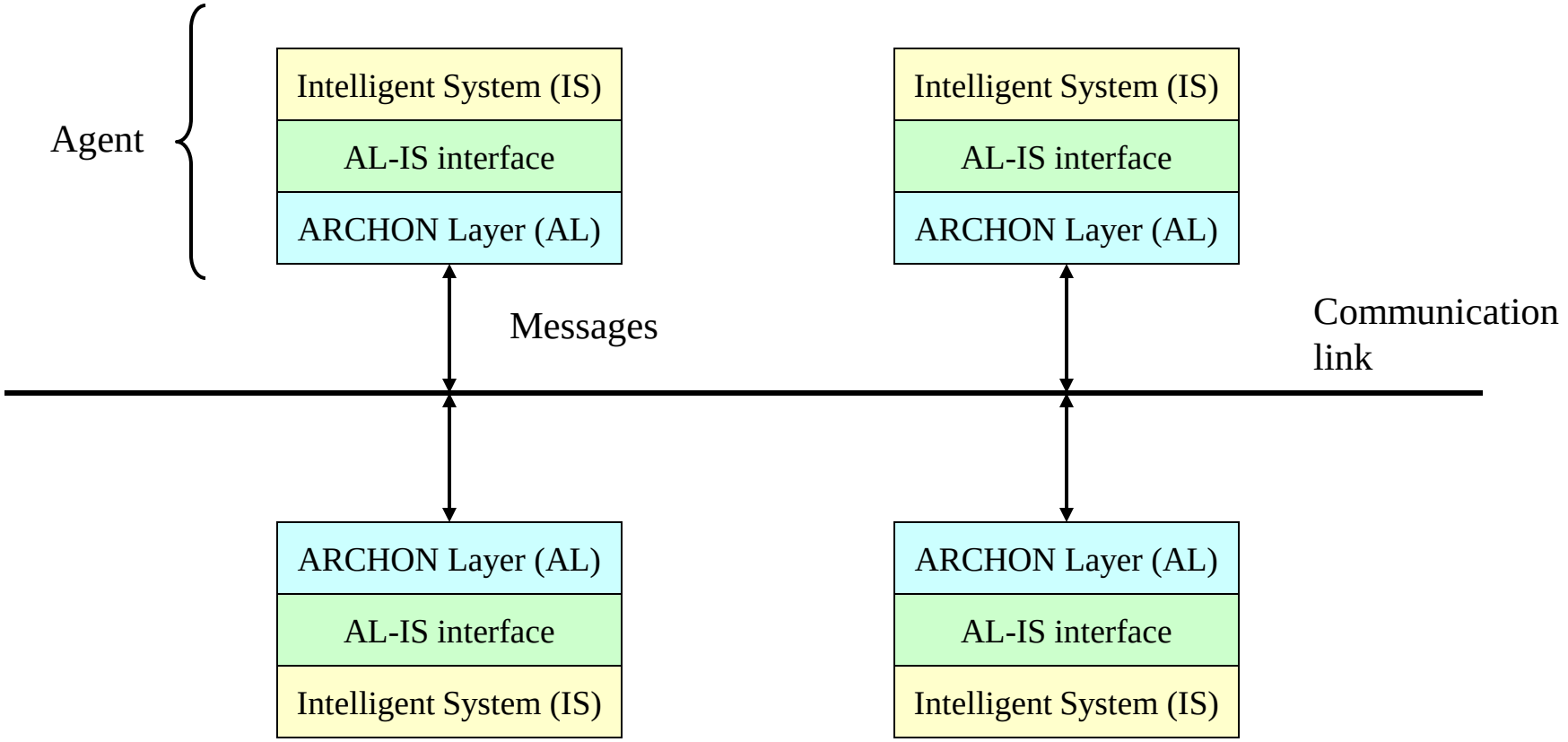
- **ARCHON** was (in 1996) Europe's largest project in the area of DAI.
- It was originally focused on getting a number of expert systems to pool their expertise in solving problems and diagnosing faults in several industrial domains.
- Supports a cooperating community that has decentralized control and individual problem solving agents.
- Consists of a:
  - **Framework** which provides assistance for interaction between constituent subcomponents.
  - **Methodology** which provides a means for structuring these interactions.

ARCHON

- ARCHON's individual **problem solving entities are agents**; they have the ability to control their own problem solving and to interact with other community members.
- Agents are large grain, loosely coupled and semi-autonomous.
- Each agent consists of an **ARCHON Layer (AL)** and an application program (known as **Intelligent System (IS)**).
- The ISs can be heterogeneous as their differences are masked by a standard AL-IS interface.

ARCHON

Structure of an ARCHON Community



ARCHON

ARCHON Community

- The system's overall objective is expressed in the separate local goals of each agent.
- Agents' goals are usually interrelated. Therefore, social interactions are required to meet global constraints.
- Such interactions are controlled by the ARCHON Layer.

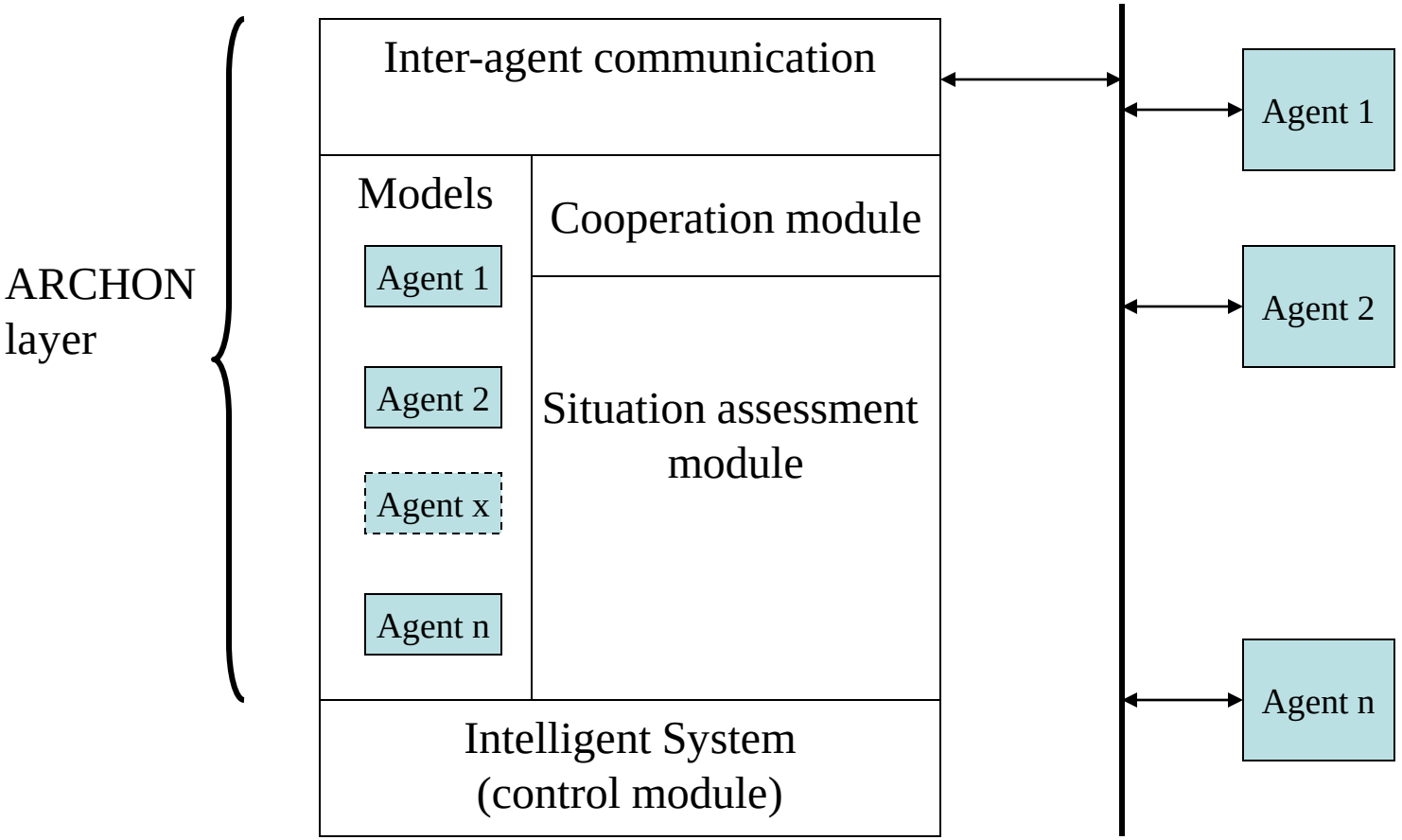
ARCHON

ARCHON Layer

- Functions of the ARCHON Layer:
  - Control tasks within the local IS
  - Decide when to interact with other agents (it needs to model the capabilities of its own IS as well the ISs of the other agents).
  - Communicate with other agents.

ARCHON

Functional View of an ARCHON Agent



MAS Architectures - Testbeds

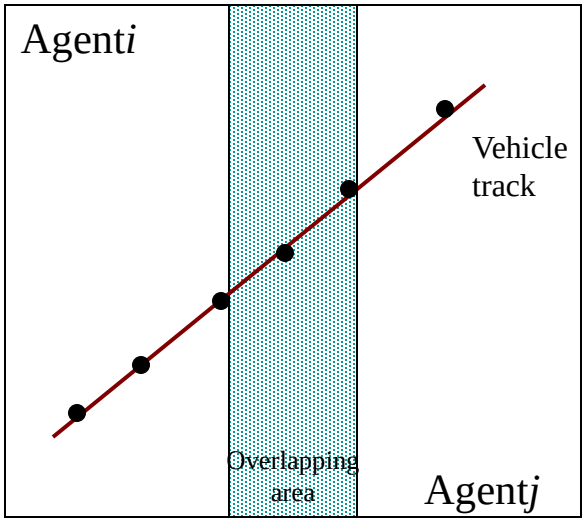
The main purpose of a DAI testbeds is to support the implementation of ideas so that they can be evaluated in a useful context.

Most DAI testbeds provide three classes of facilities:

- Domain facilities - representation and simulation of the problem being solved
- Development facilities - an environment or tools for building the agents that will solve the problem
- Evaluation facilities - tools for display, data collection, and analysis to understand how well the agents perform

DVMT V. R. Lesser - University of Massachusetts

- A DAI testbed – Distributed Vehicle Monitoring Testbed (DVMT) – to successfully track a number of vehicles that pass within the range of a set of distributed sensors (agents).



- problem-dependent testbed
- agents (problem solvers) have a blackboard architecture

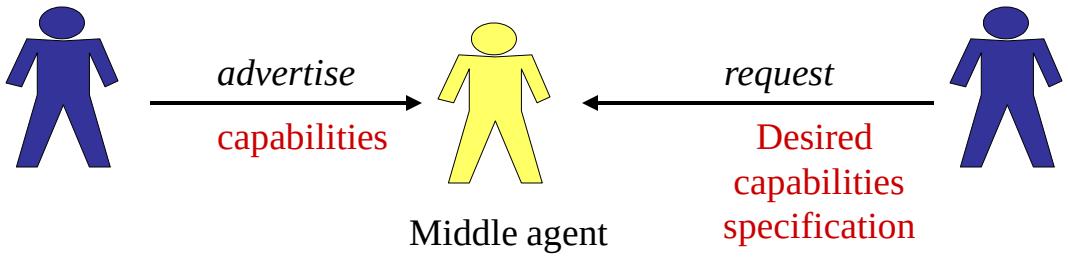
MACE testbed

L. Gasser - University of Southern California

- Provides tools for constructing DAI systems at different levels of abstraction:
  - each rule can be an agent
  - coarse-grained systems with large-scale agents
- MACE had no fixed domain for problem solving activity

Architectures that use Middle Agents

- In an open system, the set of agents is not known apriori.
- The infrastructure should provide ways for its agents to locate each other based on **name, functionality** or **capability**.
- Agents that provide this service: **Middle Agents**.
  - Facilitators
  - Matchmakers

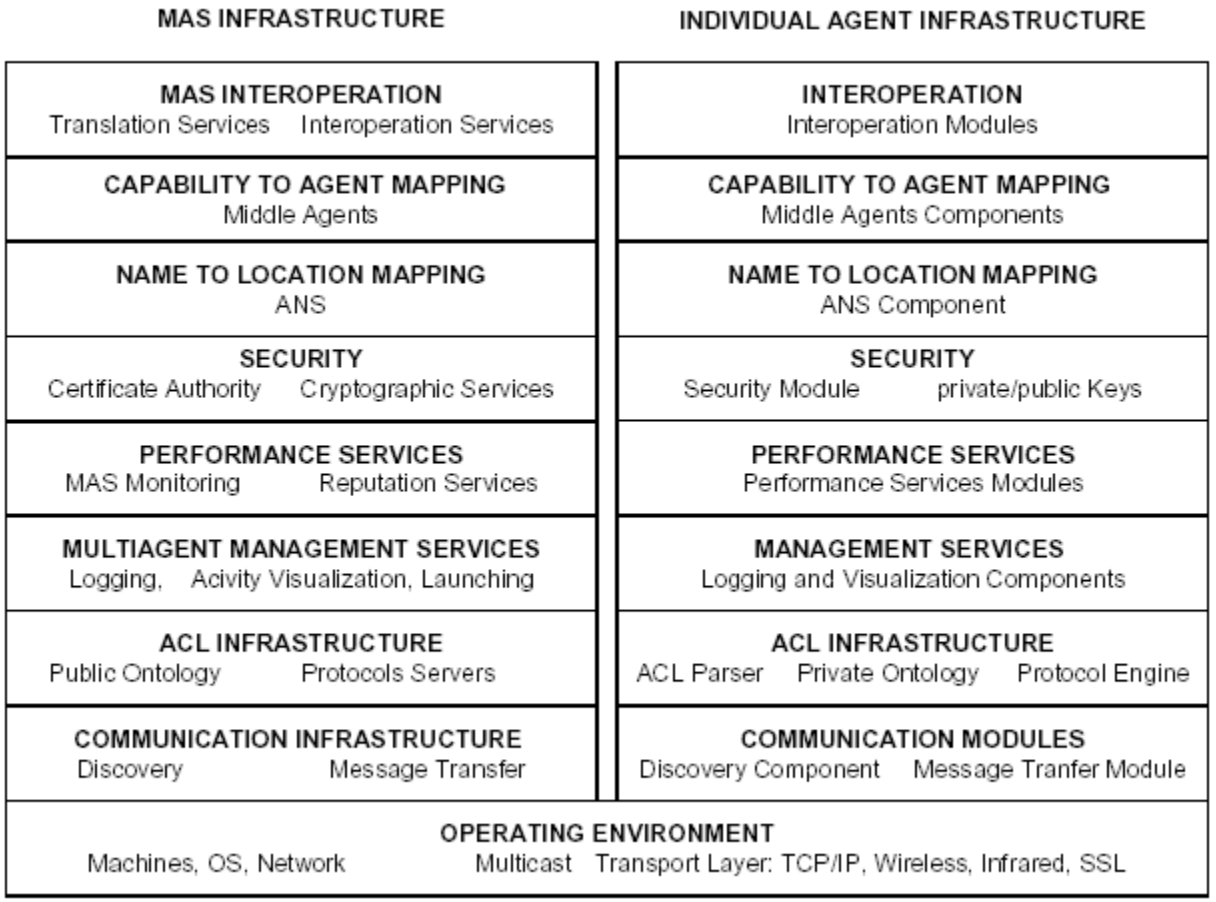


RETSINA

REusable Task Structure-based Intelligent Network Agents (Sycara et. al.)

- **RETSINA** is an open MAS infrastructure that supports communities of heterogeneous agents.
- **Main idea**: agents should form a community of peers that engage in peer-to-peer relations.
- No central control.
- Implements distributed infrastructural services that facilitate the relation between agents.

Agents in MAS



Ref: Sycara et. al., 2001

RETSINA

MAS Infrastructure

Infrastructure component	RETSINA
Operating Environment	Platform independent
Communication Infrastructure	1. Peer-to-peer between agents 2. Multicast, used for Discovery process to find infrastructure components
ACL Infrastructure	KQML, provides an ontology based on diverse domain-specific taxonomies
MAS Management Services	•Logger – records the activity of the agents. •Activity visualiser – displays activity within the system •Launcher – configures and starts both infrastructure components and agents on different machines.
Security	1. Agent authentication, via a certificate authority 2. Communication security (guarantees no eavesdropping) 3. Integrity of the components
Agent Naming Service (ANS)	Maps an agent's ID to its address in the system. ANS is queried by agents.
Middle Agents	Matchmakers distributed across the MAS

RETSINA

Middle Agents: Matchmakers

- RETSINA middle agents are called **Matchmakers**: information agents that look for other agents rather than information.
  - Records a mapping between agents in the system and the services they provide.
  - Two types of data:
    1. Advertisements
    2. Requests
  - Task of matchmaker: match advertisements to requests
  - Two types of protocols:
    1. Single shot
    2. monitor

Market-based Architectures

- Online market places offer an opportunity to buyers and sellers to meet electronically and conduct trade.

✚ Offers **benefits** for both buyers and seller:

  – Buyers: ease the process of searching for and comparing sellers.

  – Sellers: provide access to much broader customer bases.

- Major **challenges**:

  – to go beyond simple buying and selling

  – to incorporate time constraints, enforce deadlines, interact with a highly distributed web of suppliers with different capabilities and resources, interact over long periods of time and deal with failures in contract execution.

Market-based Architectures Requirements

- Provide support for a variety of transaction types (buying & selling, complex multi-agent contract negotiation, auctions).
- Control fraud and misrepresentation.
- Discourage counterspeculation.
- Provide for secure and private credit and payment mechanisms.

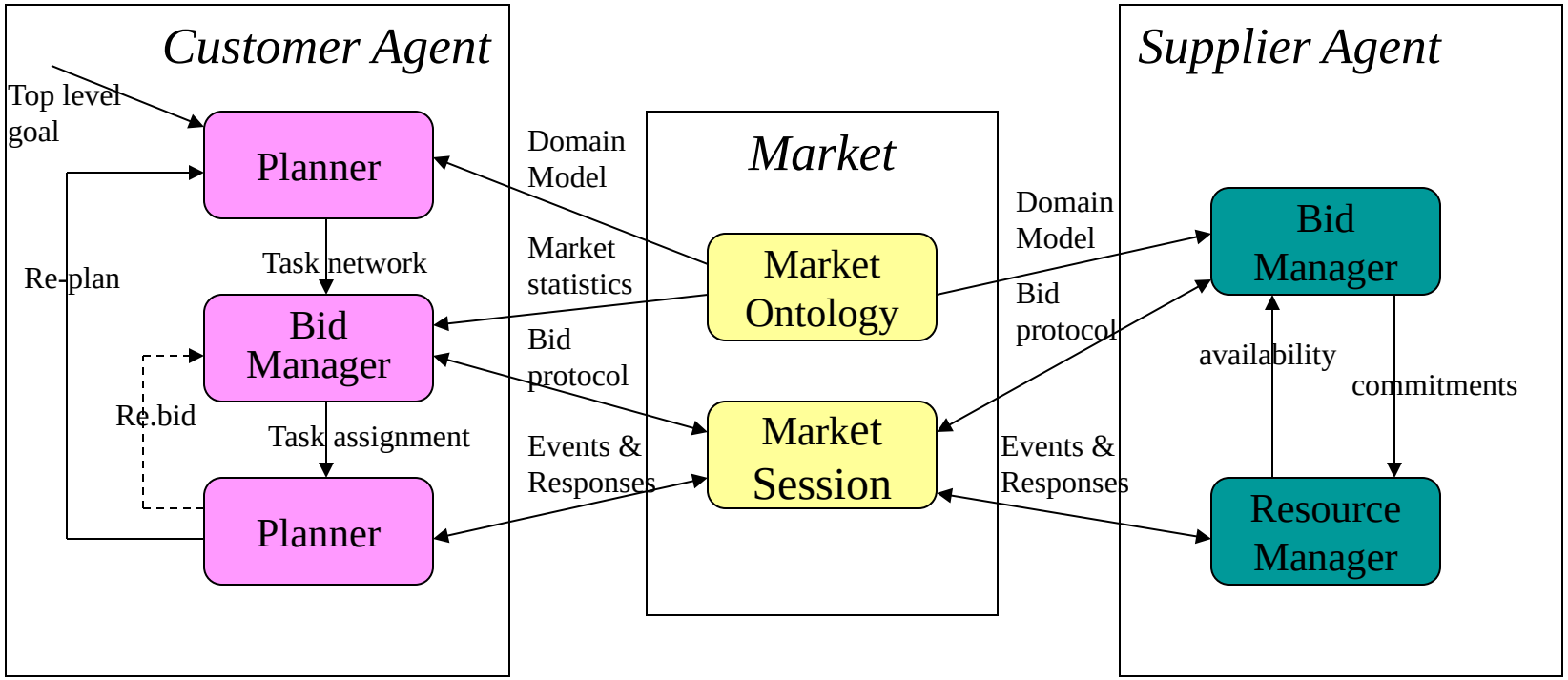
Market-based Architectures Requirements, contd.

- Provide a language in which the rich array of semantic content about commerce can be expressed.
- Provide for robust exception handling.
- Scale smoothly from local to global.
- Be extensible, by third parties.
- Interoperate with other new and existing electronic commerce services.

MAGNET:
A Multi-AGent NEgotiation Testbed
(Collins et. al.)

- A testbed to support multiple agents in **negotiating contracts**.
- Supports negotiation of contracts for tasks that have temporal and precedence constraints.
- Distinguishes between the agents:
  - **Customer**: needs resources outside its direct control in order to carry out its plans.
  - **Supplier**: provides the resources and services required by customers, for a specified price, over specified time periods.

MAGNET: Architecture

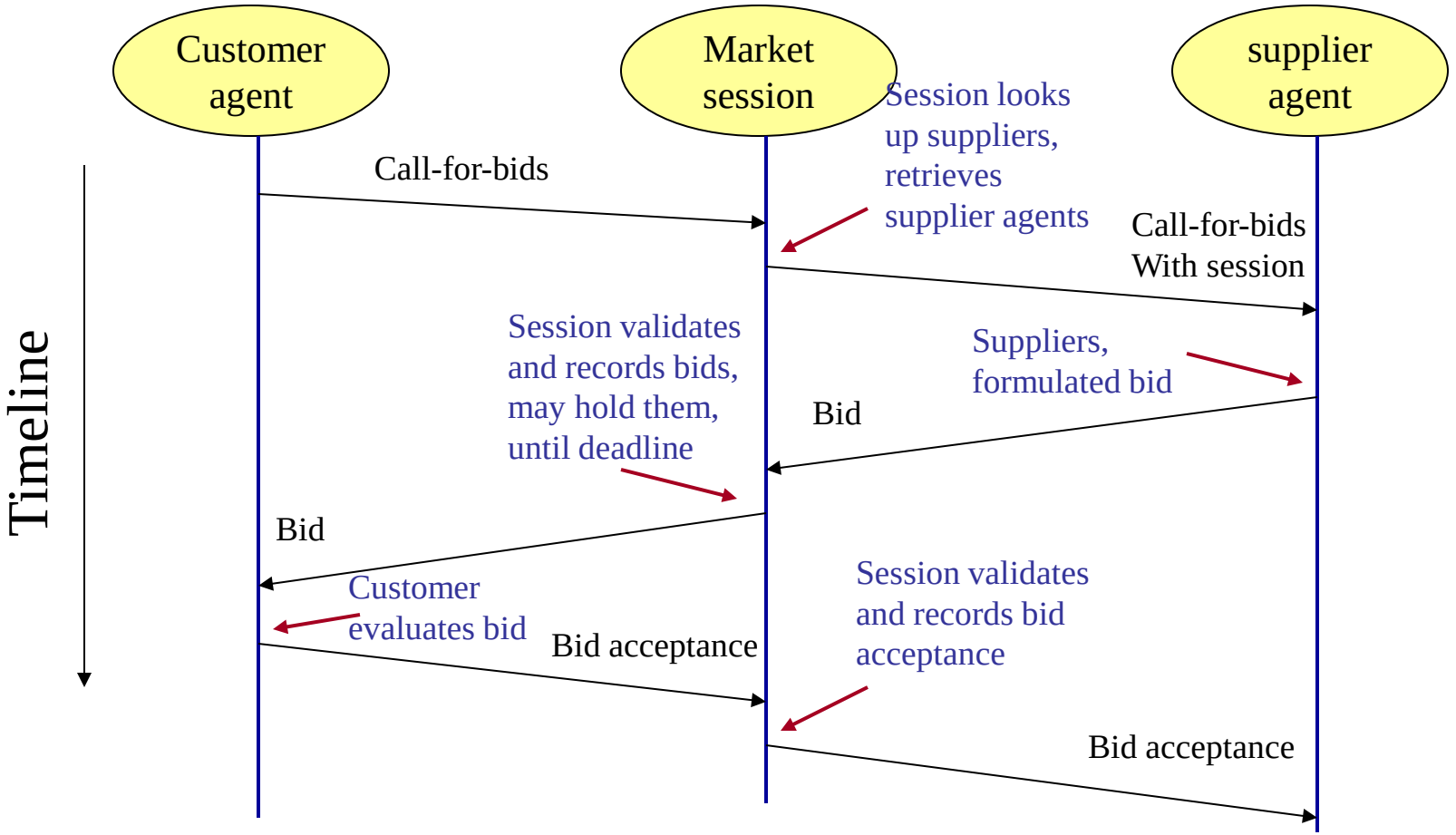


Ref: Botman et. al., AOIS, AAMAS 2002

MAGNET:
Interaction Among Agents

- A customer issues a Request for Quotes (RFQ), which specifies tasks, the precedence relations and a timeline for the bidding process.
- Supplier agents submit a bid, which includes a set of tasks, a price, a portion of the price to be paid as a non-refundable deposit and estimated timeline.
- The customer decides which bid to accept, based on cost, risk and time constraints.
- The customer awards bid, notifies the suppliers of their commitments and specifies the work schedule.

MAGNET: Session-mediated Negotiation



Ref: Collins et. al., 1997

MAGNET:
The Role of the Market

- The interactions between the customer and supplier agents are encapsulated in a **market session**. The market session:
  - finds suppliers interested in bidding for the customer.
  - provides a catalog of services (market ontology).
  - time-stamps all the interactions in order to avoid dispute among customers and suppliers.

MAGNET:
The Role of the Market, contd.

- The session:
  - Truthfully informs the suppliers of the conditions under which the bidding is being done and enforces these conditions.
  - Can limit the number of bids sent by each supplier.
  - May provide information about suppliers to customers.
  - Enforces the “rules of the market”, e.g. in a sealed-bid auction, holds back the bid from the customer until bid deadline has passed.
  - Can act as a “trusted auctioneer”.

Agora: A Multi-Agent Platform

- M. Matskin, O. J. Kirkeluten, S. B. Krossnes and Ø. Sæle. Agora: An Infrastructure for Cooperative Work Support in Multi-Agent Systems. T. Wagner, O. Rana (eds.) Infrastructure for Agents, Multi-Agents, and Scalable Multi-Agent Systems. Springer Verlag, Lecture Notes in Computer Science, Volume 1887, 2001, pp.28-40
- Xuetao Niu, Boxun Zhang and Mihhail Matskin. Pluggable Architecture and System for Support of Cooperation in Multi-Agent Systems. SERP'07- The 2007 International Conference on Software Engineering Research and Practice. Las Vegas, June 2007.
- Khan, Basit.A; Matskin, Mihhail. AGORA Framework for Service Discovery and Resource Allocation. I: *The Fifth International Conference on Internet and Web Applications and Services ICIW 2010. IEEE Computer Society 2010 ISBN 978-0-7695-4022-1. p. 438-444*
- MAS infrastructure
- Cooperative work support
- Virtual shopping center example
- Some implementation details

The main idea of our approach

- We consider a cooperative work as a set of coordination and negotiation acts (cooperative points) between participants (agents).

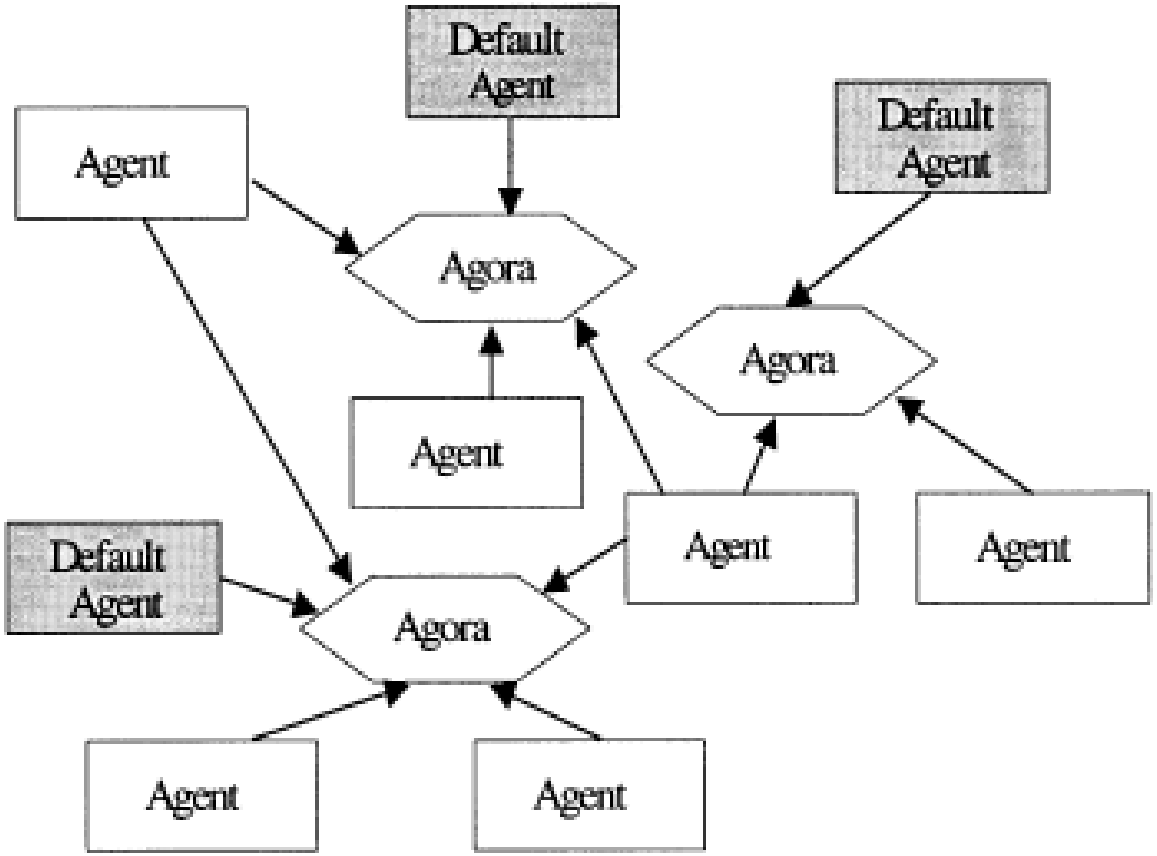
- Modelling of cooperative work assumes identification of cooperative points between participants in the work and providing support for them

Agora

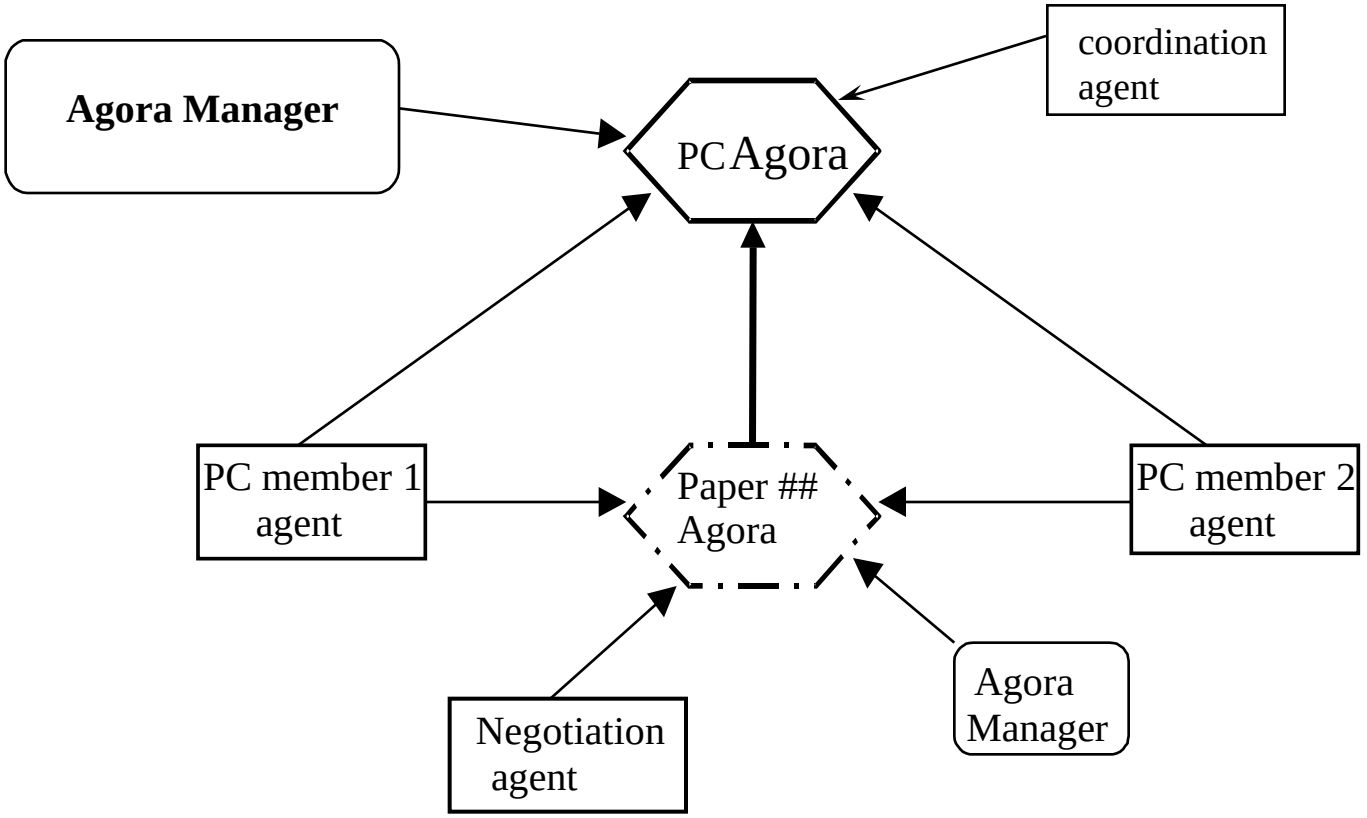
- In order to support work in the cooperative points we introduced a notion of Agora.

- Agora is a cooperative node or generic marketplace where agents can advertise themselves and establish a common context.

Agoras and agents



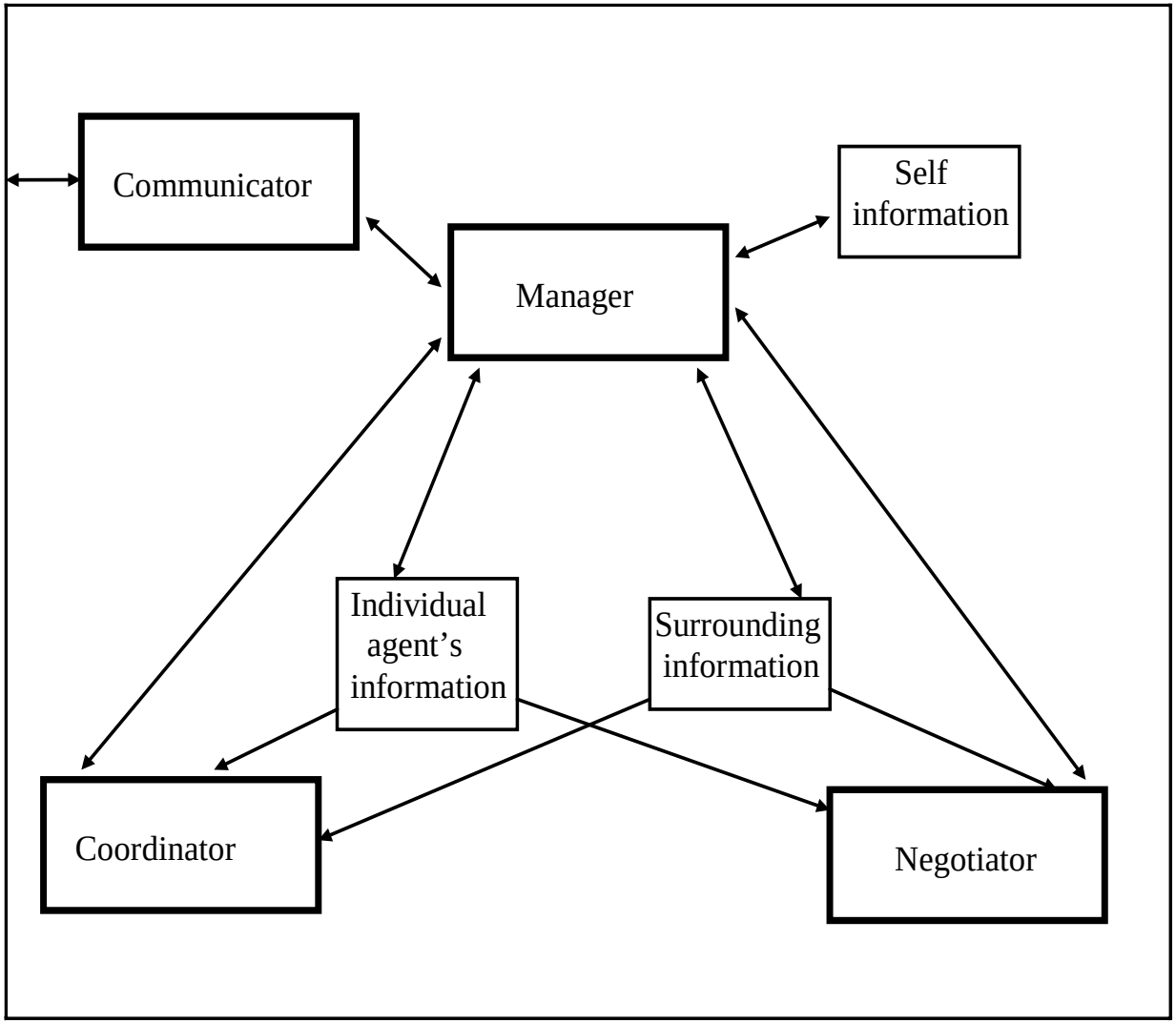
Long- and short-time Agoras



Agora contents:

- Information about registered agents (individual agent’s information),
- Information about Agora itself (group information),
- Information about other Agoras and common knowledge (surrounding information).

Agora Functional Architecture



Other possible manager functionality

- advance event handler – a complex decision procedure for starting matchmaking process, including filtering events, reasoning about necessity of matchmaking etc.
- advance matchmaking (using ontology and “non-precise” matching)
- registration/unregistration protocols handling
- decision about registration of particular negotiation and/or coordination agent
- processing queries about registered activities, ontology used and other general information
- pro-active reasoning with available knowledge
- history maintenance and analysis
- ….

Usage of Agora platform

- Having a problem first identify participants of cooperative work
- Identify cooperating points in the problem by finding a coordination and/or negotiation activities
- Map participants to agents
- Map cooperating points to Agoras
- Choose coordination and negotiation agents according to identified activities
- Create agents and Agoras

AGORA

Scenario: Virtual Shopping Mall

- Assume that there is a virtual shopping mall on the Internet containing different shops.
- There are customers who are looking for products
- Customers purchase goods by communicating with shops.
- Customers may negotiate about a price of a product.

AGORA
Scenario: Identify Participants

- Customer agent
- Shop agent
- Mall agent
- Marketing agent
- Bulk purchase agent
- Multi-product agent
- Price negotiation agent

AGORA
Scenario: Identify Cooperative Points

- Coordination of Mall activity
- Coordination of different Customer agents
- Coordination and negotiation between Shop and Customer agents
- Coordination of multi-product and bulk purchasing

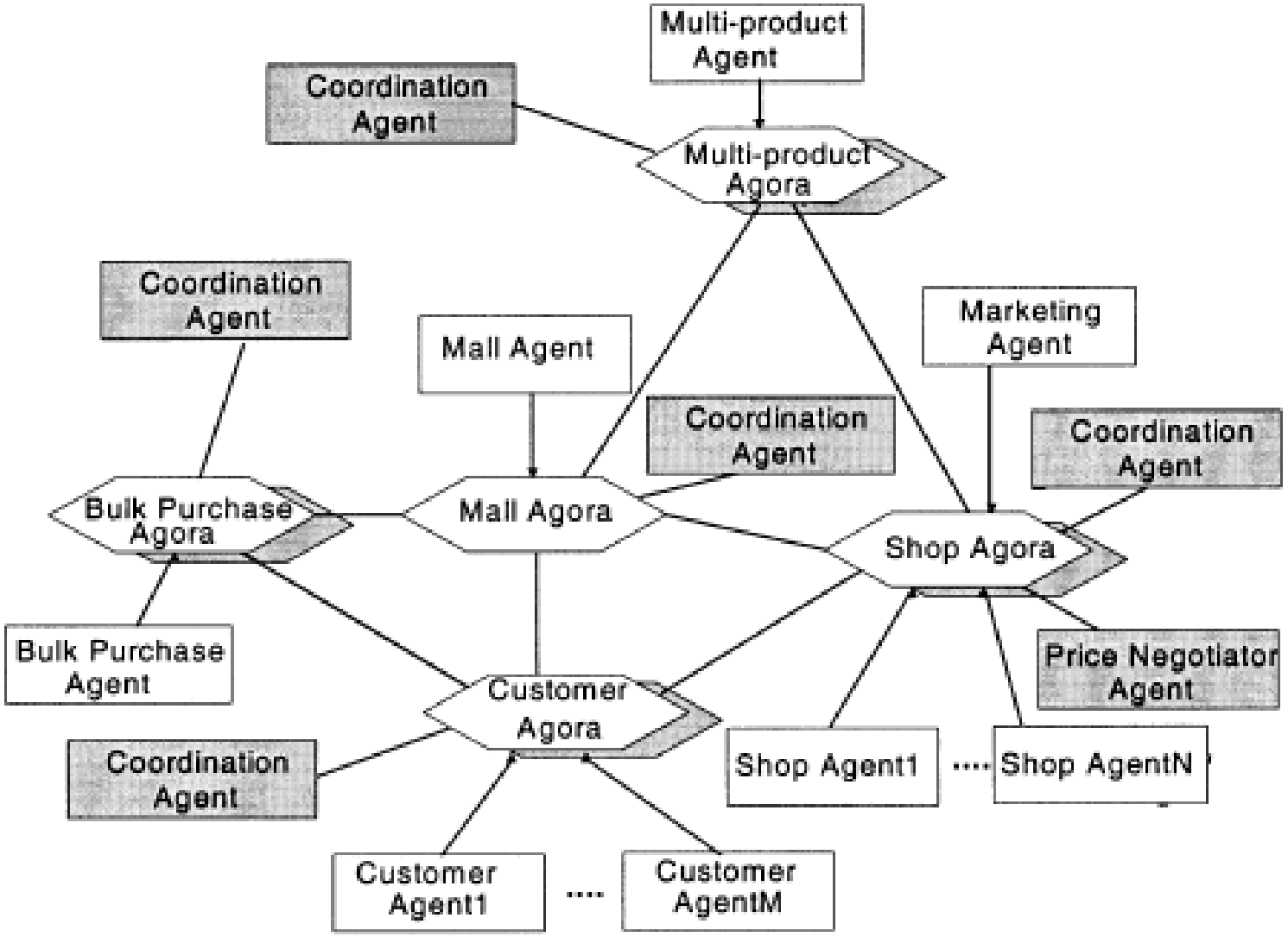
AGORA

Scenario: Virtual Shopping Mall

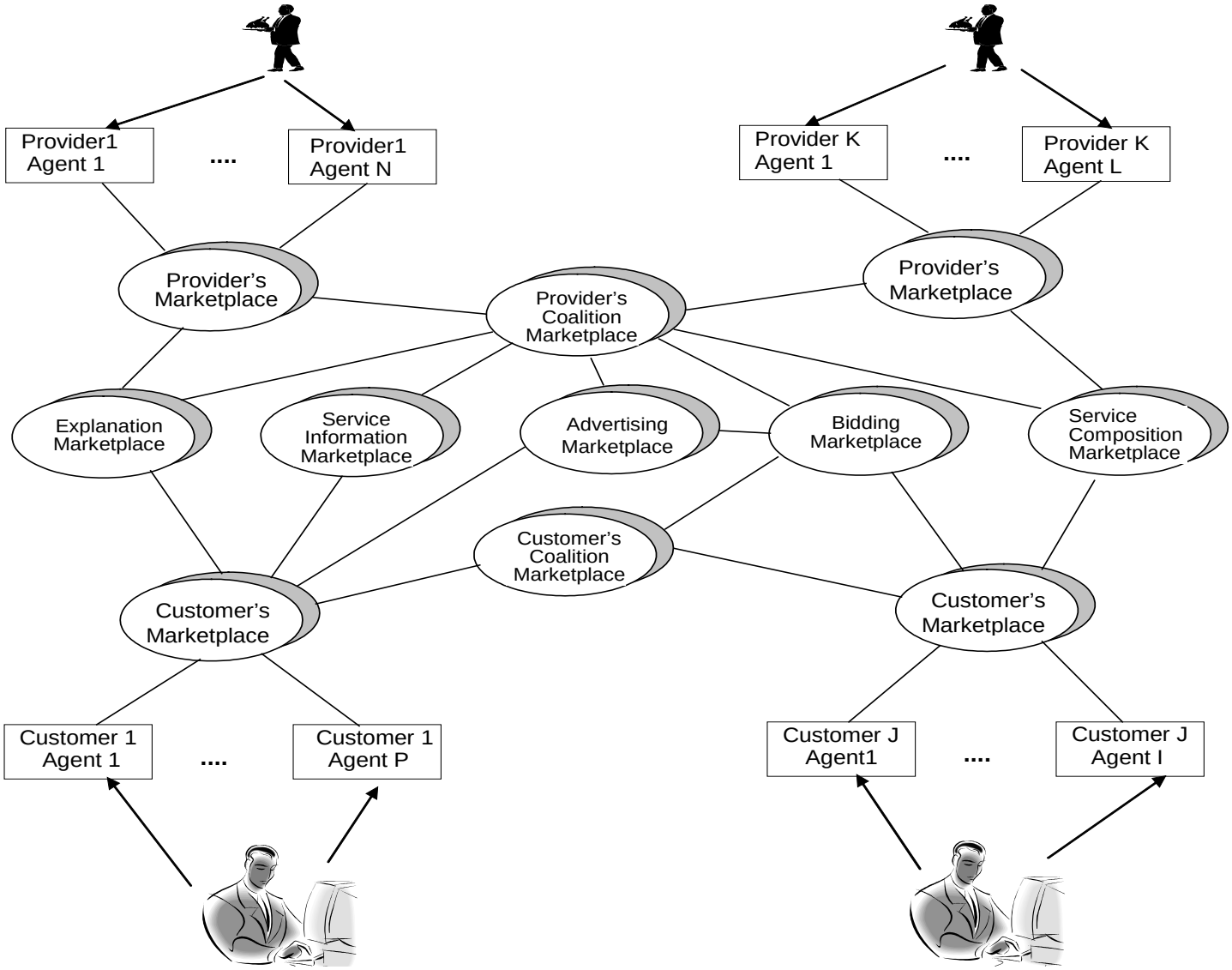
- Map participants to agents
- Map cooperative points to Agoras
- Choose coordination and Negotiation agents according to identified activities

AGORA

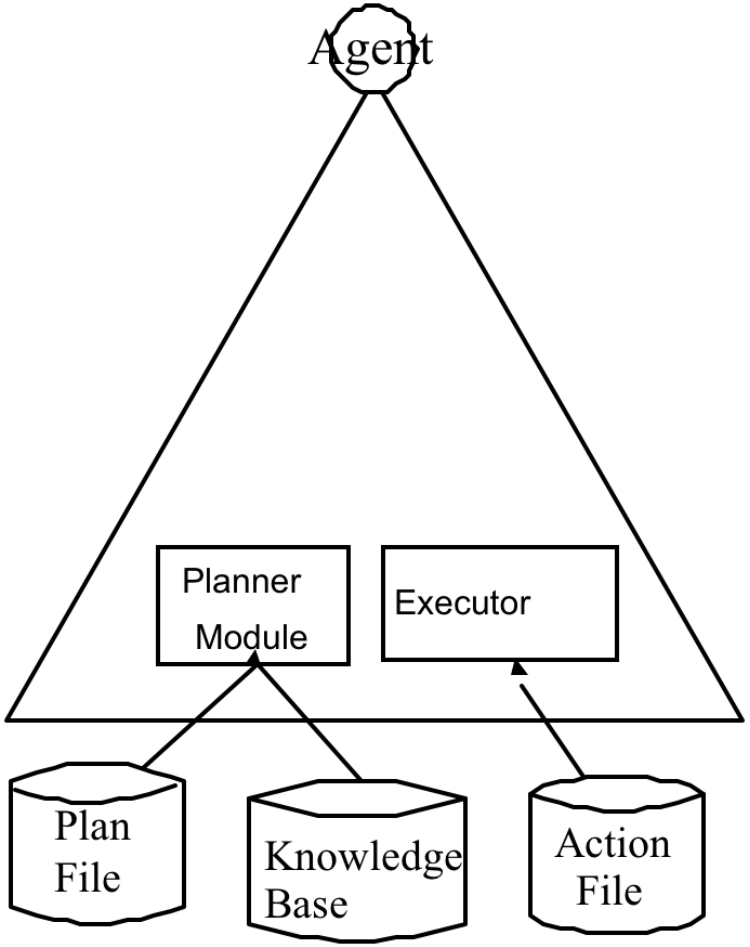
Scenario: Virtual Shopping Mall



Agora network



Easily configurable agent



A plan-file for CNP manager

```
<?xml version="1.0" encoding="UTF-8" ?>
   <agent>
      <function name="say">
         <in name="$words" />
         <println>concat(getNickname(getAID()),' says :',$words)</println>
      </function>
      <event name="onTerminate">
         <say words="'terminating'" />
      </event> <!-- search for a contractors  -->
      <df-search result="$contractors" type="'Fish'"
            protocol="'fipa-contract-net'" timeout="-1" />  <!-- define variables in outer context-->
      <assign var="$cheapest" value="max_long" />
      <assign var="$cheapest_provider" />
      <declare var="$cheapest_provider_index" />
      <assign var="$proposedAgents" value="array()" />  <!-- begin to ask all contractors  -->
      <for-each element="$contractor" iterate="$contractors">
         <cfp receiver="$contractor" protocol="'fipa-contract-net'" />
         <receive msg="$response" sender="$contractor" performative="'propose'" protocol="'fipa-
            contract-net'" timeout="3000" /><!-- check if timeout reached but no message received-->
         <if test="not($response)">
           <say words="concat( 'a contractor does not propose: ',$contractor)" />
           <continue />
         </if>
         <assign var="$dummp" value="append($proposedAgents,$contractor)" />
         <if test="smaller($response.content,$cheapest)">
            <assign var="$cheapest" value="$response.content" />
            <assign var="$cheapest_provider" value="$response.sender" />
            <assign var="$cheapest_provider_index" value="sub(length($proposedAgents),1)" />
         </if>
      </for-each> <!-- in case that no one propose  -->
      <if test="not($proposedAgents)">
            <say words="'no contractor proposed'" />
            <terminate />
      </if>
      <assign var="$dummy" value="remove($proposedAgents,$cheapest_provider_index)" />
      <reject-proposal receiver="$proposedAgents" protocol="'fipa-contract-net'" />
      <accept-proposal receiver="$cheapest_provider" protocol="'fipa-contract-net'" />
      <receive performative="'inform'" sender="$cheapest_provider" protocol="'fipa-contract-net'" timeout="-1" />
      <say words="concat('make the deal with ', getNickname($cheapest_provider),'at the price of ',$cheapest)" />
   </agent>
```

Negotiation and coordination agents

- Both Negotiation and Coordination agents are ordinary registered agents with some predefined functionality
- Their specificity lays in maintenance of corresponding negotiation and coordination protocols.
- The protocols are presented as plan files to be performed by the agents.

Ontology and performatives

Performative	Ontology	Meaning
Propose	Registration	Ask for registering new agent
accept-proposal	Registration	Inform if registering was accepted
reject-proposal	Registration	Inform if registering was denied.
Propose	AgoraNetwork	Ask for registering new Agora
accept-proposal	AgoraNetwork	Inform if registering was accepted
reject-proposal	AgoraNetwork	Inform if registering was denied.
refuse	Advertising	The Agora can't help the agent about a product, but may recommend another
failure	Advertising	The Agora can't help right now, but may help later.
inform	Advertising	Tell what s/he wants to sell
query-ref	Advertising	Asking for product information
inform	Consume	Give detailed information about product.
confirm	Consume	Continue with another client
confirm	UnRegistration	An agent sends this to unregister on Agora.
inform	UnRegistration	Deregistering accepted
disconfirm	UnRegistration	Deregistering denied

AGORA: an infrastructure for MAS

- a conceptual method for modelling cooperative work by identifying work participants and cooperative acts/points,
- we try to keep a proper level of abstraction in the multi-agent architecture,
- Agoras give a practical support for negotiation and co-ordination by providing an infrastructure and templates for cooperative points
- openness/extensibility is an essential feature of the approach

Conclusions

- We consider our MAS infrastructure as middleware on the top of system software
- cooperative work support is an important element of the infrastructure and it needs conceptual solutions
- extensibility and default components are important features
- we should be as much as possible close to “standards” (if they don’t exist then to general tendencies)

Next Lecture:

Agent Oriented Software Engineering

- Wooldridge: ”Introduction to MAS”,
  - Chapter 9 (except 9.4)